Streaming the IMDb Top 250

Co-authored by
Rob Acheson and Jeff Fontas

Table of Contents

Project II Proposal

This is the proposal we submitted for our project. For our implementation journal, click here.

1. **Tentative project title**
   Streaming the IMDB Top 250

2. **Name, email, programming comfortability of each member of your group**
   Rob Acheson, racheson@fas.harvard.edu, comfort ⅘
   Jeff Fontas, jfontas@fas.harvard.edu, comfort ⅗

3. **Research questions and hypotheses**
   Provide the primary research question(s) and any secondary questions you are trying to answer with your data.

Questions

We want to answer the following research questions with our visualization:

1. Using IMDB's top 250 as the base group of films, what are the 10 most viewed movies on 1channel.ch?
2. How does the number of votes for a movie on IMDB compare to the number of views of that movie on 1channel.ch?
3. What sites serve the most films and where are they located?
4. What countries serve the most films?

Hypotheses

We anticipate that a couple of different things will be discovered from our data and visualization. First, we anticipate that there is going to be a positive correlation between the number of votes on IMDB for a particular movie and the number of views that movie has on 1channel.ch. And second, when it comes to sites and countries, our hypothesis is that there are really only a handful of sites and a handful of countries that serve most of the content.

4. **Motivation**
   Explain why you are interested in your research question(s).

Since the dawn of Napster in the late 90's, the web has changed the way the world exchanges and consumes content. Today, there are hundreds of different ways that people can legitimately. The content-creation industry has gone through a decade or more of adjustment in order to accommodate consumers in the transformed distribution landscape, and as such more content consumption services and products appear every day.

Even with legitimate platforms available, underground methods for sharing pirated content proliferate. BitTorrent, p2p networks like DC++, and online locker storage services all provide sophisticated ways to exchange content outside the normal distribution models. As with most illicit/taboo subjects, these pirated content networks exist in some of the darkest corners of the web.

Given the flux in the media distribution landscape, and because the untold dollars the content-creation industry reaps in revenue, online piracy is consistently one of the most publicly discussed issues of our time. In the past legislative session, an anti-piracy bill, SOPA, eventually stalled in Congress after a coordinated blackout from Google, Reddit, Wikipedia, and other popular websites. In the future, how governmental and non-governmental actors approach solving this problem is going to be deeply consequential for how information will be exchanged around the world.

If we are to draw informed conclusions about the issue of online piracy, we need to better understand the darker corners of the web that are contributing to it. For our project, we want to explore content-indexing websites, a method of illicit content distribution, to understand where and how content is being shared through this method, and who the key players are that are involved in making this distribution model work.

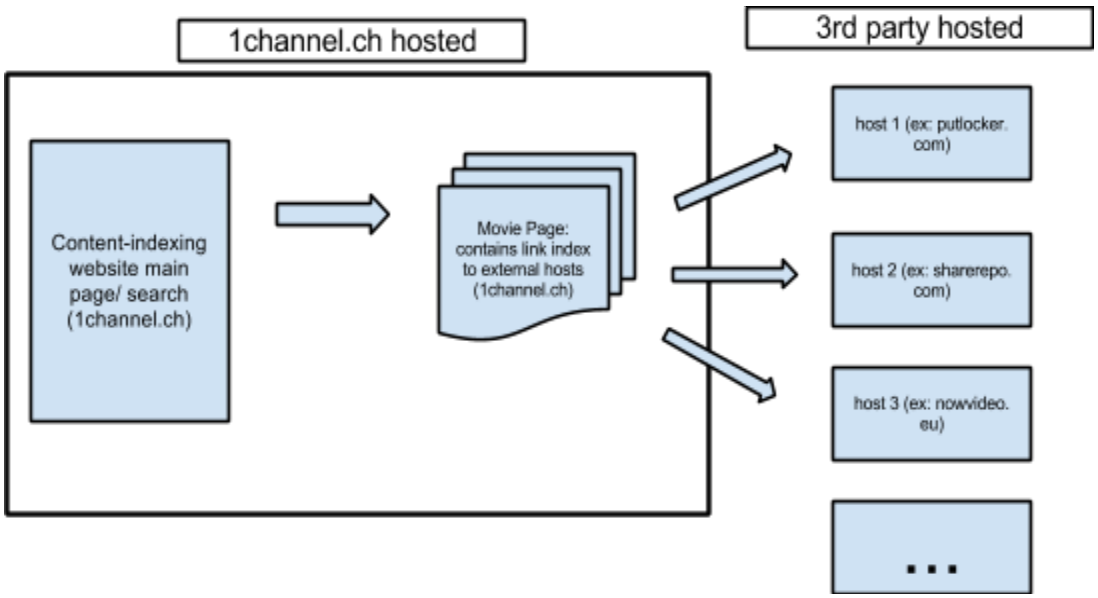
5. Data

What data will you use to construct your visualization? How is it obtained? How is it relevant to your research questions? If appropriate, provide a link to your data source.

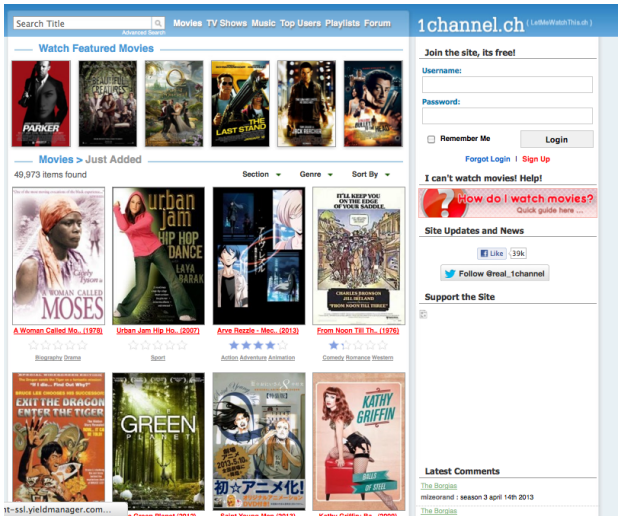
Dataset Description

To explore this issue, we want to focus on one particular online distribution model, content-indexing websites, and to limit our dataset, one website: 1channel.ch. Among the various content-indexing websites, 1channel had the highest Alexa rank (297, 167(!) in the US). Users on 1channel can access over 49,000 movies and over 6,000 TV shows, all for "free." Some of the most popular movies on the site have over 1,000,000 views.

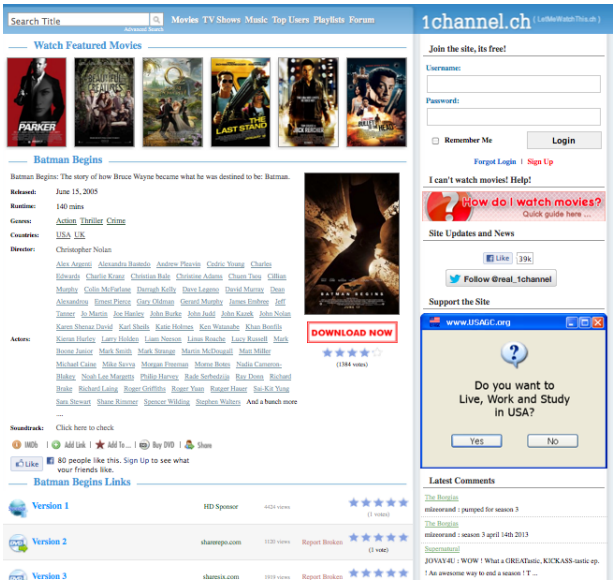
In order to explain the value in the scraping we will be doing of this website, it is worth it to take a step back and explain how content-indexing websites work.



The above diagram represents the general structure of 1channel.ch and other content-indexing websites. The websites are designed so that the indexing site itself does not directly host any content (i.e. intellectual property). This is similar to the way peer-to-peer networks and torrent sites like thepiratebay.se do not host any actual content, but instead provide a gateway to the content.



1channel.ch main page



1channel.ch content page

On the main page of the indexing website, users can browse and search for content and then open a page for the specific movie, TV show, etc. itself. This content page contains links to a bunch of externally hosted copies of the content. For example, if the content page was for the movie “Batman Begins”, there would be a list of direct links to a bunch of different third-party sources of the movie.

The external hosts are typically obscure “locker”-style websites. These websites provide users the ability to upload files and generate a publicly-facing, unique URL that anyone can use to access a stream of the content. All someone has to do is open the URL and stream away.

Scope of Dataset and Sources

We are going to be scraping 1channel.ch for the top 250 films on IMDB, using a Python script and some different Python modules. At a very high level, our script will do the following:

1. Retrieve the top 250 films from IMDB.
2. For each movie, open 1channel.ch and build a search string for the movie. 1channel.ch uses keys in the search string URL to prevent DDoS/webcrawling (at least, that is our guess), so we will need to open the main website every so often to retrieve the key as it is refreshed. It is typically refreshed every 10 minutes or so. We have already built this functionality in a script, along with some of the other steps.
3. Using the search string URL, open the search results page and grab the appropriate movie. We are going to perform some basic validation to make sure the year release in the results page matches the year released on IMDB. This will enable us to make sure we grab the correct content page.
4. Open the content page. On the content page, retrieve the list of third party links and the number of times they have been viewed.

During this whole process, we will be building a JSON object that contains the information we want to use for our visualization. The JSON object will look like the following:

```
{       FILM:
        {       Hosts:
                        [ Name of host:
                          {       Views for host
                                  Number of links for host
                                  IP address
                                  Location
                                          { Country
                                          City
                                          Country Code}
                        }
                        Name
                        …
                        Name
                        …
                        ]
        }
```

```
        {       IMDB:
                {       URL
                        RatingCount
                        Year
                        Directors
                        Genre
                        Image Link
                }
        }
}
```

IP Geolocation Data

One of the most interesting answers we want to gather from our dataset is where the hosts of the content and the indexers are located. We are going to capture this information using Python modules that interface with the MaxMind GeoLite Free database. This is a database of billions of IP address and their national original released for free by MaxMind. According to the FAQ on the MaxMind website, the accuracy rate for IP address by country from their internal tests is 99.8%.

Additionally, there is also a GeoLite city database that geocodes IP address by city. The city information is much less accurate, but the important piece of the city information for us is just to provide a degree of geographic disparity. This will help when multiple entities are rendered within one country, since there is less overlap.

Below is a snippet of code that retrieves the IP address for a given URL and the national original of that IP address:

```
>>> import pygeoip, urllib2, urlparse, socket
>>> gi =        #the geoip datasource
>>> url_sharerepo = urllib2.urlopen('http://sharerepo.com/cxzmsozjym8h')
        # A sharerepo.com link for “Batman Begins” retrieved from 1channel,
>>> socket.gethostbyname(urlparse.urlparse(url_sharerepo.geturl()).netloc)
        #get the ip address
'37.1.200.212'
>>> sr_ip = socket.gethostbyname(urlparse.urlparse(url_sharerepo.geturl()).netloc)
        #get the ip address but this time store it as a variable

>>> gi.country_code_by_addr(sr_ip)          #print the country code
'NL'
```

6. Potential Visualizations, Technical Process, and Three Sketches

Explain what potential types of visual encodings would be appropriate for visualizing your data and why they might be appropriate (note that this may change once you have your data in hand, but that's why this is just a proposal). Please describe what tools you might use and how you intend on using them to construct your visualizations. If you choose to keep your project I data and questions, explain how your new visualization will improve over the previous one, or how it will differ. Explicitly list and address the comments you got from your TF for project I.

For our visualizations, we are going to use D3.js to compose our visualization. Below is the most recent sketch of our visualization, with an explanation of what the individual labeled sections are and how they will interact.

V1: Top ten chart and all films checkbox

This the first interactive section that allows the user to update the other linked views. It displays a list of the top ten films and a checkbox that toggles the display of the remaining 240 titles. Individual title(s) can be selected to update all other charts. For example clicking on the first title in this list will:

- update V2 to only show results for this current film

- highlight the film's point in V3
- redraw the map only showing statistics for the current film

V2: Views by Site - Bar Chart

This is a bar chart displaying the number of views reported by 1Channel on the y-axis broken out by host site on the y-axis. Results will be ordered by most views. This will quickly allow a view to determine which sites serve the most content. This chart will update based on selections in V1 and redraw to only show views based on the selected current selections in V1. If no selections are made in V1 it will display top 10 or all films depending on the checkbox.

V3: 1Channel views vs number of IMBD ratings - Scatterplot

Because of the sheer number of data points, 250 to be exact, we believe a scatterplot will be most appropriate for this visual. Using positional encoding, the number of views reported by 1channel.ch will be plotted on the y-axis and the number of ratings in IMDB plotted across the x-axis. We will use three colors to differentiate between the data points. The top 10 results will be highlighted by a bright single color. The remaining 240 will be show in gray. If a particular title is selected in V1 the highlighted title will remain bright and the remaining will dim to somewhere in between the color and gray.

V4: The Map

The map is will offer insight into large trends quickly. Using a combination of positional and area encoding, one will quickly be able to see the number of views and where the content was hosted. The area of the circle will expand based on views by site. All sites will be shown in a similar semi-transparent color. Hovering over a circle will present a modal window close to the location of the circle. Within this modal site specific information will be displayed (country, city, total views, etc.).

V5: Basic Project Information

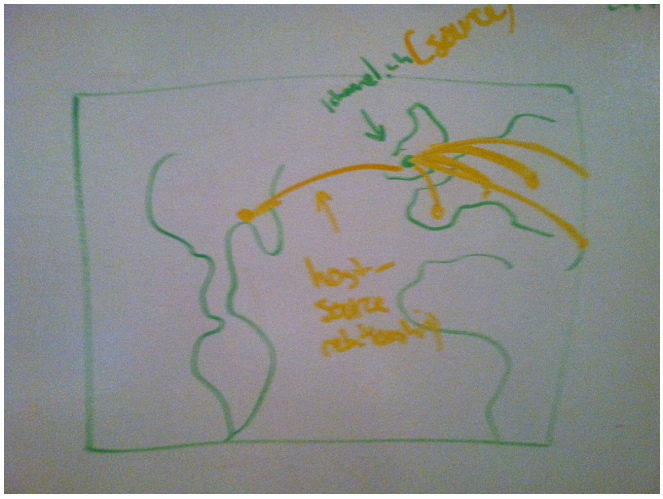
This section will just be static text. It will provide a brief description of our visualization, the source of the data, and some interesting findings.

V6: Map Controls

V6 is filed under our "nice to have" column. We are thinking of using this section for map controls, so that users can manipulate a zoomed in view of the V4 map by brushing over a smaller world projection within the V6 box.

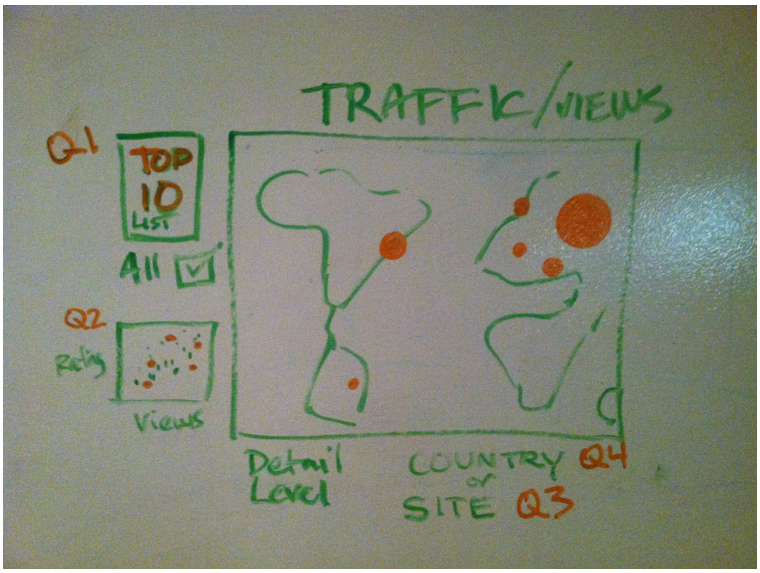
Sketches

Sketch 1



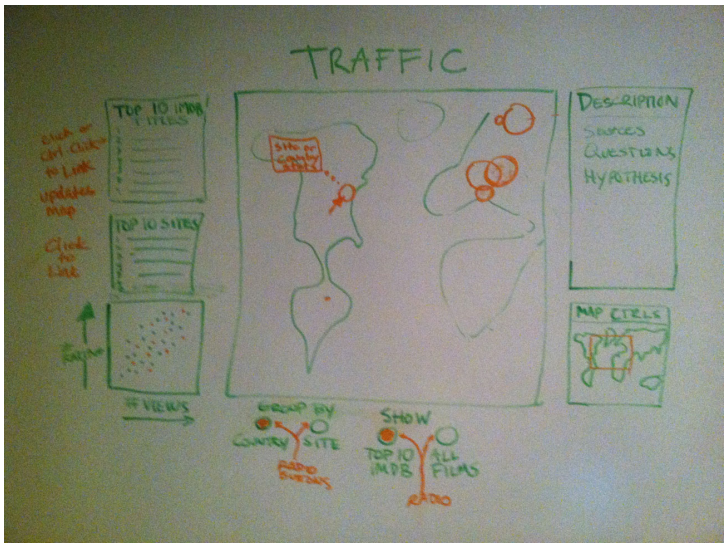
We knew when we began that needed to use a map of some sort in our visualization. At first we were thinking about using the map to show the "flow" of content from content-indexing site to all of the hosts. This was ultimately decided against because encoding the data as lines wouldn't enable as easy a method to compare different hosts or countries as using area or other methods.

Sketch 2



In this sketch, we got closer. Here we encoded number of views by country as the area of the circles. We also brainstormed about how to show different filters/linking. We started discussing using a list in the top left corner, then drilling down to agreeing that we wanted to use the top 10 highest rated movies from IMDB as a subset. We included the "All" checkbox and the country vs. site options to enable additional filtering of the data. We also talked about how to show the relationship between the number of ratings on IMDB for a movie vs. its views on 1channel.ch.

Sketch 3a



Building on the previous drawings the purpose of this sketch was to help determine what tools we would need to allow the viewer to interact with the chart. Specifically, how would a user use linking, generate details on demand, and filter the content. We wanted a list to link views, filtering by radio buttons and details on demand via tool-tip popover. Top 10 sites were considered as a list and radio buttons as a filter, but in the end these seemed to confuse the user experience and were removed in the last iteration.

Sketch 3b, Final Sketch

This is our final sketch we produced. It is described fully in the top part of this section.

Project II Implementation Journal

As we mentioned in our proposal, our goal from the outset was to scrape information from content-indexing and streaming websites that exist in the darker corners of the web so that we could learn more about them. In order to accomplish the objectives we set out in our proposal, we divided our workload into to two major chunks--designing our scraping script and creating our dataset, and then secondly, building the visualization.

The natural division of tasks among these two major tasks worked out well; it enabled one of us to predominantly work on scraping or visualization development while the other tackled the HW 6 problems. As lessons from the HW began percolating, we were then able to utilize that new knowledge about the realities of working with D3 to redefine our the structure of our dataset. At the end, we then teamed back up to finish development and work through other design issues that we hadn't resolve previously.

In the following sections, we walk through the development, setbacks, and design decisions we made to present information about the global infrastructure that enables millions of viewers around the world to freely consume digital media. We will start by describing the development of our scraping script, then discuss development of our visualization. After that, we will conclude with the answers our visualization provides to our research questions.

Design & Architecture of our Scraping Script

Overview

In order gather information from 1channel.ch for the top 250 movies on IMDB, we built a script in python designed to scrape the data from both sites and simultaneously create an object containing our dataset. During initial planning and design of the script, we used the structure outlined in our proposal to guide development. The initial modules we developed to scrape movie titles from IMDB conformed to this spec, but as time went on we had to rework the structure in order to accommodate additional scraping and using the data in D3. In all, the scraping itself was a very time consuming process--due to complications with our data source (1channel.ch's search functionality was broken the week we had set to begin working on the project) and navigating scraping issues, it took us about two weeks to scrape and organize the information the way we wanted. Our script predominantly used Pattern for scraping, and used a module called pygeoip to connect to the MaxMind GeoLite City database for geocoding.

Script Structure

Our script is made up of a collection of modules that were designed to accomplish three distinct

tasks. First, we needed to scrape IMDB for the top 250 movies, and build an object that could contain within it information about the hosts for all of these movies that exist on 1channel.ch. After this module and the initial object were built, all we needed to do was update films within this object with additional information we scraped from 1channel.

Next, we needed to use the data we grabbed from IMDB as fodder for our scraping of 1channel.ch. This module also needed to be capable of parsing and traversing the search results from 1channel so that the information about the particular movie we were searching for could be retrieved. Finally, using the scraped information from 1channel, we needed to geocode the host information in our object in order to plot it on a map.

In order to control the scraping process, we used a main script that called each of the individual modules we developed and also limited the information we were performing operations on (see diagram). This file, **1channel_scrape.py**, evolved significantly over time. Because of the fact that we only had a need for certain modules at certain times, we rewrote the code in this file constantly to turn on or off calls to particular functions or modules. For example, once the script ran through the IMDB scraping process, we commented that section of code out because the IMDB section of our object had already been built. Most of the work that each task required was performed almost entirely in the specialized modules we developed. The main script only consisted of a few lines of code related to the execution of the module in question--the rest of the execution and building of the object was contained within the module.

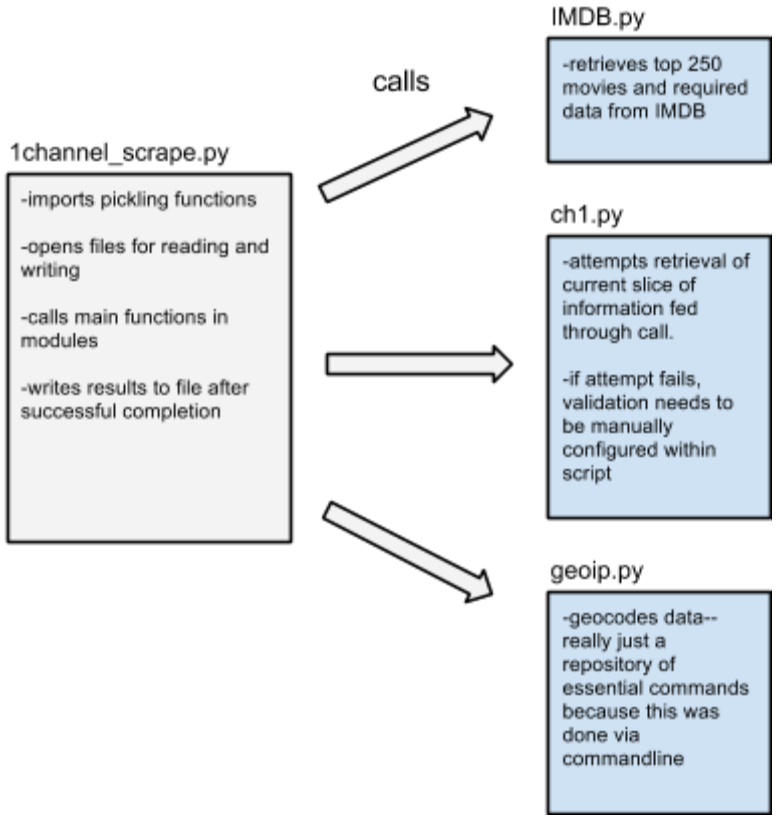


diagram of scraping script architecture

In addition to **1channel_scrape.py**, we built the following modules:

1. **imdb.py:** provides a series of IMDB functions that assign the IMDB values to that section of the object. This module was called first and originally built a dictionary of films and their attributes as it ran through the top 250 film list. We later on had to modify the structure of our dataset away from the dictionary setup, but regardless, this module and the dictionary of films it created formed the foundation for all our future data. The module was designed to compartmentalize all IMDB operations inside the module itself--only one line of code is called in the **1channel_scrape.py** in order to build the entire 250 element object. Because IMDB is such a big website, we did not limit the amount of movies we retrieved per attempt, we just created the script so that it retrieves them all.

2. **ch1.py:** provides a series of scraping functions for building the 1channel section of the object. The module is designed to scrape 1channel.ch by utilizing the site's search feature. 1channel uses a key in its search string, and our script retrieves this key from the main page prior to each search and builds the search url for each movie being retrieved. After it retrieves the movie search results with the search url, it identifies the correct movie in the search results by performing a validation and then retrieves that movie's content page. The content page was where we scraped all our information from--it contained information on all the hosts hosting a copy of the movie and how many times it had been viewed. Using some functions from the built-in socket and urllib2 modules, we retrieved the IP address of the hosts listed on the content page for geocoding later.

   The execution of this section of the scraping ended up being much less automated than in the IMDB module. This was a result of the validation we used to verify that we were selecting the correct movie from the 1channel.ch search results page. The IMDB title and year were used to search for and then validate within the search results the film we were looking for. This validation step posed significant challenges (the next section of the document is dedicated to explaining these difficulties). This module initially took the dictionary of films as input we created from **imdb.py** but was eventually refactored to support receiving an array of films instead.

   Whether it was necessary or not, one of our objectives during the scraping of 1channel was to conceal as much as possible the fact that we were scraping this information at all from the site itself. To this end, we configured the script to make all its requests through Tor so we could conceal their source. Tor is software that basically enables greater anonymity on the web. One version of the program runs locally on a user's machine and acts as a SOCKS 5 proxy which forwards all traffic through Tor relays all around the world. Because the requests to 1channel.ch are coming from Tor relays, and not directly from the machines we used to scrape, it would appear on its face that there wasn't

coordinated scraping going on.

3. **unescape.py:** this module encapsiulated Frederik Lundh's unescape function for unescaping html entities so that they appear as unicode. This module was used briefly during the IMDB scraping process to unescape titles with foreign characters.

4. **geoip.py:** this module contains most of the code we used for geocoding the hosts in our dataset. Because all it really took was one carefully crafted loop to geocode the whole dataset, the geocoding section was done largely through the command line.

Results from Scraping

Using these modules, we were able to scrape a total of 247 out of the 250 titles from IMDB off of 1channel.ch. Our dataset ended up containing 45 unique movie hosts in 15 different countries. All of the titles we scraped had been viewed a total of 18,856,069 times as of March 24, 2013.

All that said, our scraping was not smooth sailing. Some of the problems we faced are listed below.

Problems with Scraping

During the scraping process, we ended up encountered a number of issues and oversights in planning that needed to be addressed. Broadly, these issues related to two big problems in our approach to scraping our dataset.

The first big issue we discovered was that the structure of our dataset as it was originally developed was way too cumbersome to work with. As noted in our proposal, we originally intended to structure our dataset as a dictionary, using the title of each film as the root-level key. One of the big issues with this was that it made iterating over the dataset during the 1channel scraping stage incredibly difficult. Dictionaries are designed to not have any order at all, and instead of working around this fact, it simply made more sense for us to just restructure the dictionary as an array of all the films (stored as objects) that we were analyzing. Otherwise we would have had to use modules that sorted dictionaries... blegh!

The second reason we had to restructure our dataset was to enable easier manipulation of our data in D3. As our scraping progressed and we also made it through HW 6, we discovered that the data function in D3 took an array as input. This reality, in addition to the ease with which we could slice the dataset during the scraping process, made refactoring the object a no-brainer.

The second major issue we ran into was discovered after a few trial runs of scraping the information from 1channel.ch. As we were testing, it became increasingly clear that there were

major inconsistencies between the data we had scraped from IMDB and the data we were searching for on 1channel.ch. The script performs a validation on the search results retrieved from 1channel.ch in order to ensure that when we search for a vague title like “The Dark Knight” or “Star Wars” that could return multiple results that we are selecting the correct result. Our validation iterated through the search results, looking for both a match to the title of the movie in IMDB and a match for the year of that title’s release. If both those conditions were true for a particular result, the module would then retrieve that movie’s content page. Unfortunately, the validation didn’t always work perfectly and we ended up having to manually address all the inconsistencies we had in our dataset.

The typical validations issues we ran into were the following:

1. **Years not matching:** If the year of release on IMDB and the year of release on 1channel.ch were different, the validation would fail even though the movie did exist on both sites. The typical way this was resolved by adding conditional logic for that particular movie; for example, if the title of the movie being searched for was “Star Wars” and the year of release in IMDB and 1channel were different, before the validation was performed we would temporarily assign the year value for Star Wars from IMDB to what it was on 1channel.ch. This preserved this validation for other scenarios--there were cases where for movies with sequels where this validation did work and was helpful.

2. **Punctuation and special characters:** In some cases punctuation would cause issues during search or validation. In some searches, a search with an apostrophe would not produce the necessary results, so the search terms were manually modified prior to search to produce the correct movie result (for example, “Schindler’s List”’s search keywords were simply changed to “schindler”, which produced the necessary movie on the results page).

   In addition to apostrophes, special characters posed a much bigger problem because they typically broke the script at multiple stages in the scripts execution. For example, “Amélie” (as listed in IMDB) failed on both during the search stage and the results validation stage. This was because the title did not include the special character on 1channel.ch, and not only that, the title of the film was completely different than the one listed in IMDB (1channel listed the film as “Amelie from Montmartre”). In this case, we manually changed the search string to “Amelie”, and then, on the results validation step, we used conditional logic to address the naming inconsistency.

3. **Foreign film titles:** In many cases, foreign films were either listed in translated form on IMDB and in their original language on 1channel, or with the english title and the foreign translation in parentheses. This title inconsistency had to be addressed before being able to retrieve the proper movie content pages, and we used similar conditional logic to the above Amelie example to resolve the problem.

4. **Some titles just simply didn't exist on 1channel:** There were three titles that didn't exist at all on 1channel. These were lesser-known documentaries and foreign films, and even after aggressively searching manually for them, they were just nowhere to be found.

In total, there were about 40 titles where there were issues with inconsistency between the IMDB and 1channel.ch databases. Manually addressing them took some time because of the fact that when an issue was encountered, we had to manually review the problem and address the issue after the review.

Geocoding the data

Because we retrieved the IP address of each host listed for each movie during the 1channel scraping process, we had all the necessary data we needed to geocode the sites after retrieval. We decided during the course of retrieval to up the ante and use the GeoLite City database for our geocoding. Since the accuracy by country was already incredibly high (~99%), the added bonus of being able to possibly retrieve city data was a big help--the chances were pretty good that multiple sites could be hosted in the same country. Being able to geocode at the city level would (hopefully) enable us to map sites so that they weren't overlayed over top of each other.

The actually geocoding was done from the commandline in an interactive Python session. It was only about 6 lines of code to loop through all the films in our dataset. Most of the code is listed below:

```
import pygeoip
gic = pygeoip.GeoIP('GeoLiteCity.dat')
for x in mov_obj2:
    if "1channel" in x:
        for hosts in x['1channel']:
            x['1channel'][hosts]['Location']['country_code3'] =
gic.record_by_addr(x['1channel'][hosts]['IP'])['country_code3']
            x['1channel'][hosts]['Location']['city'] =
gic.record_by_addr(x['1channel'][hosts]['IP'])['city']
            x['1channel'][hosts]['Location']['Lat'] =
int(gic.record_by_addr(x['1channel'][hosts]['IP'])['latitude'])
            x['1channel'][hosts]['Location']['Lon'] =
int(gic.record_by_addr(x['1channel'][hosts]['IP'])['longitude'])
```

Building the Visualization

Overview

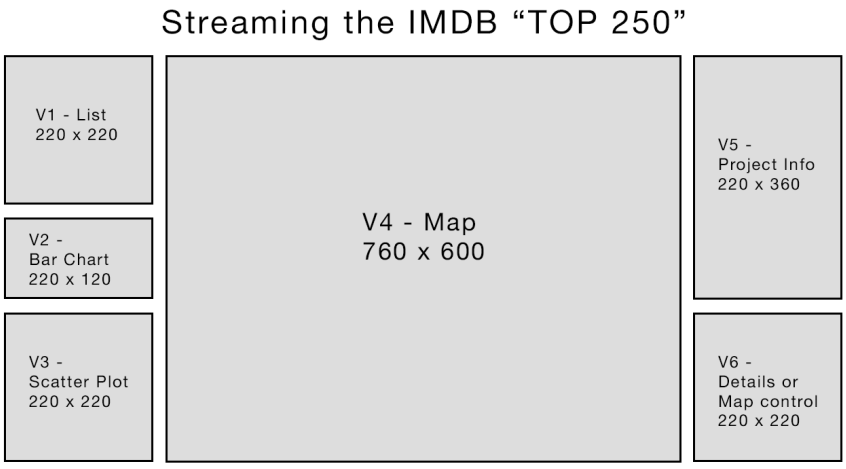
In order to build our visualization, we relied heavily on our sketches for our jumping-off point. Then, using photoshop, we built some basic mockups to explore the feasibility of the ideas we developed in our proposal. During this process, we made some adjustments to our objectives so that the visualization would be large enough to display a lot of information in a web browser yet also not be overcrowded.

After we had decisions made regarding layout, we took time to build some foundational code that would allow us to each branch off individually and develop subsections of the visualization on our own. Once we had these set up it enabled a lot of personal development time for each of us to assemble our individual sections. At the same time, we needed to be able to coordinate between ourselves so that when events happened within our individual visualizations they influenced the rest of the page.

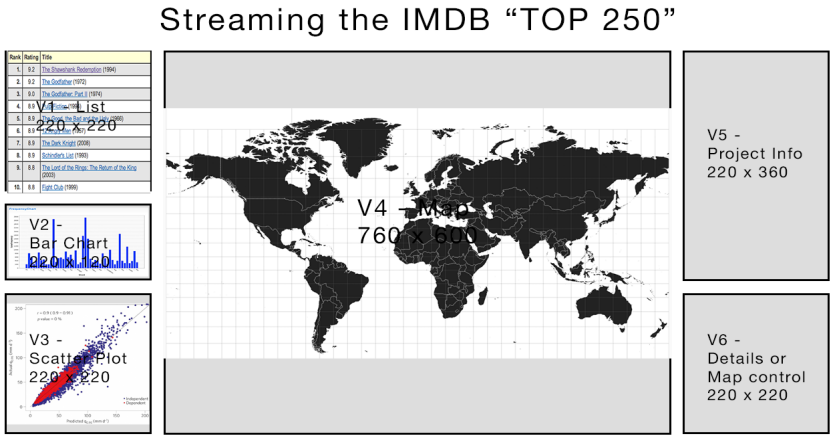
The following section walks through the progress we made in each of these areas and the design decisions we made along the way.

Layout Design

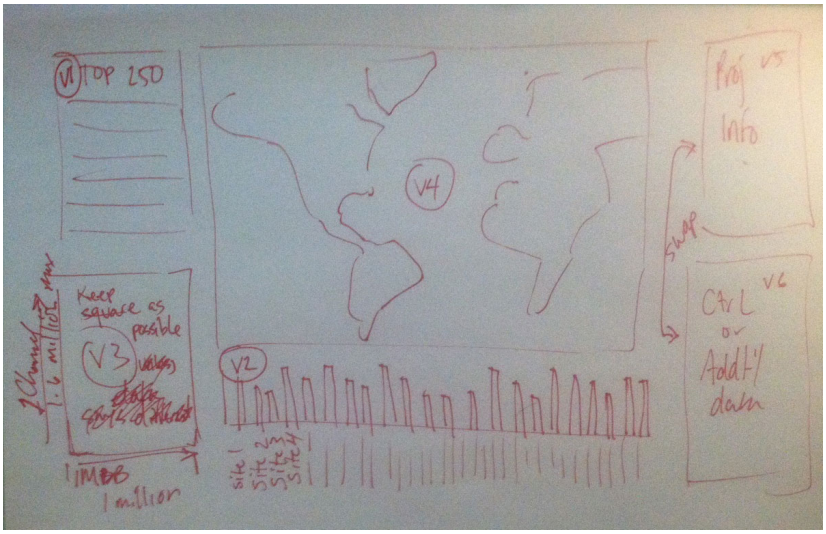
We started developing our layout by crafting an initial mockup based on the sketches we provided in our proposal. Using 1280 x 720 as our resolution, the initial mockup that we came up with looked like this:



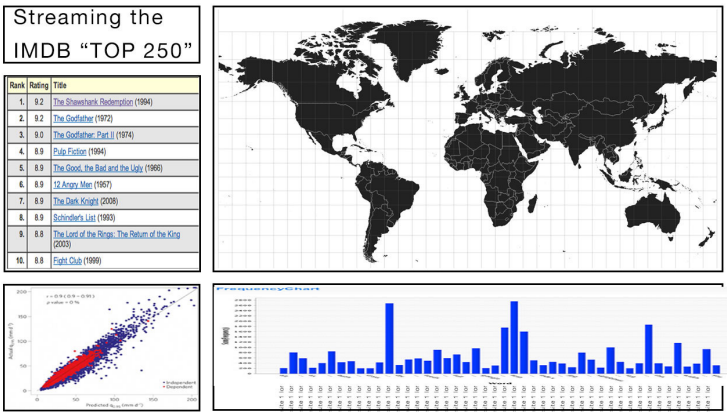
After we completed the mock-up, we searched for charts with similar data-density to the points we were expecting. During this time we also explored different projections to use for the map, and determined that the mercator projection would best suit our needs. After we plugged the images into our chart, we got this:



Looking at this iteration of the mock-up, it was clear to us that the map was taking up much more space than it actually needed. Additionally, the barchart appeared much too small to be of use to anyone viewing the visualization. If we were to move the bar chart beneath the map we could better show of the sites that we wanted to show in the bar chart and provide more space for the scatterplot. Here is a sketch we did during our deliberations:



After talking through some other issues regarding control of the map, we decided that the controls for the zooming would probably be irrelevant, and definitely fall in the “nice to have” category. We made another iteration of the mock-up to see if the visualization provided the view into where these streaming sites were and which sites were serving the most content.



This is the final mockup we created and largely resembles the visualization that became our end product. The overall visual design was meant to be simple, easy to navigate and clear to understand. We used a column layout both to separate the individual pieces and to create a natural alignment. As the viewer first scans the page, they will encounter the title of the visualization and the list of titles. This list was intentionally placed in proximity to the title so that one could quickly realize that these were the 250 titles that our visualization would focus on. Below this list, the 250 films are seen on a scatter plot. The right side of the visualization holds the map which is the centerpiece of the document. The bar chart below is large enough to display the data for all sites in our dataset and provides details that are not easily encoded in a proportional symbol map. The color palate was decidedly simple as well. Items selected in the list are a muted shade of red which mirrors the data the other charts. Deselected items are shown in a muted gray to contrast with the areas of red. Because all many elements are repeated in different manners, but with similar visual styles, parallels can quickly be drawn.

Designing the Core Functionality of our Visualization

The first step in actually creating the visualization was building the page template in HTML, **index.html** and styling with basic CSS, **style.css**. We had a solid plan for the types of charts we would use, so we did not use other tools such as Tableau or ManyEyes to create mockups. Instead, the iterative process of design happened as we were learning D3. Each section of the page was created within its own div and had its own Javascript file associated and was positioned with CSS.

Beyond the visual design, the first area of major development was **main.js** to manage the loading and parsing of the JSON input. As the JSON structure of our dataset was somewhat complex, it needed to be transformed into objects and arrays that D3 could easily ingest. There are two main arrays that store all of the data. The first, **films** stores film specific objects. This array is then used to build the **sites** array which held site ratings and views by input list of titles.

Because we needed a way to coordinate multiple linked views and the interaction that occurred within them, main.js was designed to serve as the intermediary between the charts and the data. As we mentioned, the main update function parsed the data to be used in all the views and then called an update function for each view to display on screen the data that had been parsed. This update method could also conveniently be called after users interacted and filtered the data; the parsing method could be passed a filtered list of titles which then was parsed and passed back to each view when it was redrawn

As we progressed further along, we integrated the interactions we made in our individual sections with the rest of the views . In particular, we created a link between the brushing feature in the scatterplot and the list view so that the list would highlight the selection made in the scatterplot. Simply using the update methods didn't generate this result, there was some additional coding that needed to happen to achieve this functionality.

Map.js

Overview

In our map we wanted to present a picture not only of where the sites were in the world that streamed the movie content we were studying, but we also wanted to display the proportion of views streamed by each site. In order to encode multiple variables, we decided on a bubble chart for our map because it would enable us to both encode location and proportion in a way that quickly enabled users to figure out who the big players were in the streaming game. Because the range of views for a given site could be anywhere in the double digits to the millions, we knew that the viewer would be able to quickly discern proportionality among the given sites. A color encoding was discussed as well, but because the range was so huge we felt that the proportionality simply would not get conveyed effectively, so we decided against it.

In order to overcome the basic flaw in the bubble chart--that users can't effectively discern smaller differences in area, we also decided to augment the bubblechart with a tool tip that would display the actual views. Not only that, the bar chart (discussed later) provided information in a different encoding, enabling easier comparisons for the user.

Deciding on Projection

After starting the initial page layout and visual design, we began considering what map type of map projection made the most sense. One of the things we realized during this process was that the map was really there to provide symbols of country boundaries for context--i.e. the shape/projection of the country didn't matter as much as the fact that it be *recognizable*. The

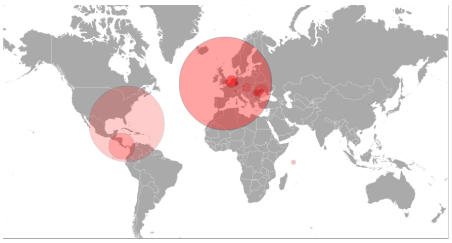
viewer's only task when looking at the map was to be able to identify the country or region of the globe the site was located in. Given the freedom we got from this realization, we ultimately decided on a Mercator projection because it was conveniently rectangular, and it is one of the most common global map projections. If for whatever reason the viewer couldn't identify a country, we country information was going to be included in the tooltip as well.

Development in D3

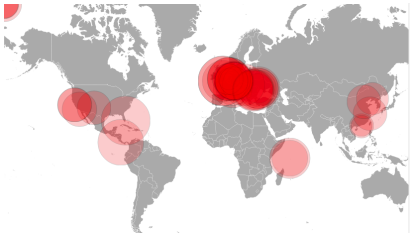
One of the tools we used to build our map was TopoJSON data provided by Mike Bostock. This enabled us to draw the map itself in D3. With data in place, correctly formatted, we was able to get a pretty decent map visualization working. The first iteration showed the locations of the sites, but did not convey the number of views.

A Question of Scale

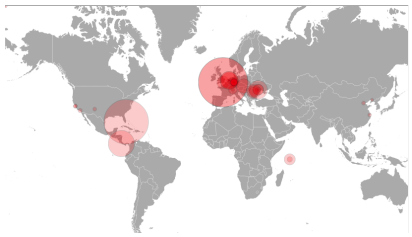
One of the issues we came back to time and again throughout development of our visualization was which scale we should use to present our data. Before we made a decision, we made sure to view the data in a variety of scales to understand the implications of each design choice. Fortunately, D3 makes doing so incredibly simple--changing one section of a line of code is all this necessary to present the information in an entirely new way.



Linear Scale



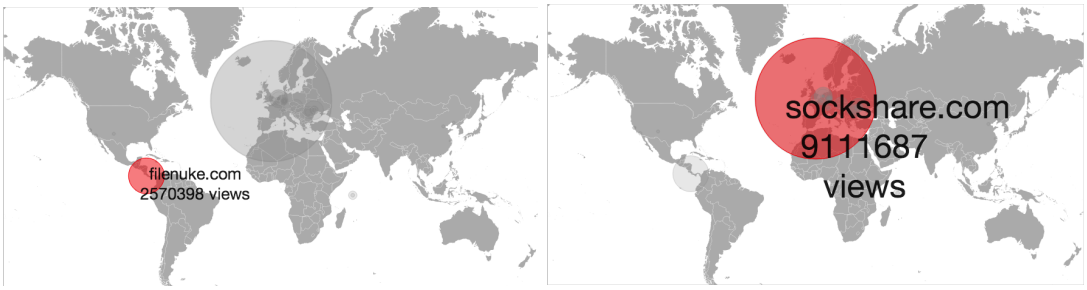
Log Scale



Sqrt Scale

The different scale encodings we tried are above. One of the things that was particularly difficult was that there was a huge range in the number of views for each site. This made a square root scale or similar exponential scale appealing, it frankly just looked better. But at the same time, we were incredibly weary of the “lie factor”. At the end of the day, accuracy was most important to us, so we went with the linear scale. As an added bonus, we also decided on a linear scale for our bar chart and scatter plot, so this also lends some consistency among our different views.

Tooltip



One of our final additions to the map was the tooltip. The tooltip is essential for providing specificity to our map visualization--without the tooltip viewers would have been able to clearly understand the proportional and geographic relationships in the map but they would not have been able to “drill down” to understand that values at play behind the scenes.

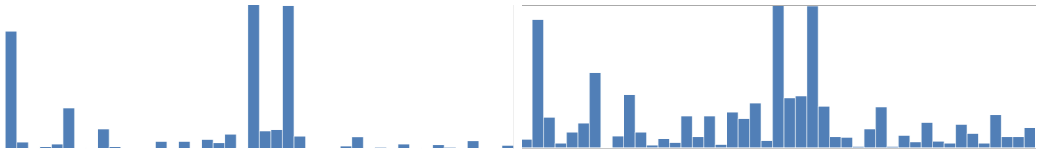
In the tooltip, one of the things we toyed around with was scaling text. Ultimately, as can be seen in the image above, this really wasn’t a viable option. The range between our maximum and minimum values was too large, and not only that, but viewers could already see the relationship between various entities in the visualization. Readability was the key here. With formatting, our final tooltip ended up looking like this.

Bar.js

Our bar chart worked hand-in-glove with our map visualization because the bar chart displayed the exact same information, sans the geographic data. The chart encoded the number of views positionally by site, giving the viewer the encoding most suited to the task of comparison. We were fortunate that the visualization left us with a wide and short section to display the chart--at it its widest, there are 45 sites being displayed on screen.

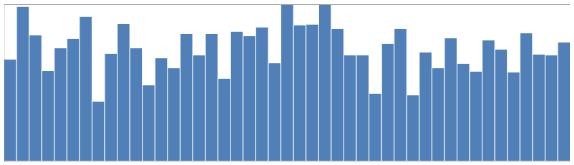
Scales

Even with the bar chart, there was no way we could escape another discussion on the scale with which to encode the data. As in the other scenario, we plotted these out in a very basic chart as our first step design step. Then, using some quick edits in D3, we changed the scale to observe how the different scales conveyed the information.



Linear Scale

Sqrt Scale



Logarithmic Scale

After looking at the different test scales, and considering other design choices we had made, we had no choice but to continue with the linear scale we had chosen in the map. First of all, it was important for us to show consistency across visualizations. If we didn't show the proportional relationships between the sites consistently, the viewer could easily suffer from the cognitive dissonance of seeing the same relationships portrayed one way in the bar chart and completely differently in the map. If that was the case, we would not be conveying the connection between the data on the map and the bar chart effectively.

The other reason we went with the linear scale is because it is the most truthful presentation of the information. As we can see from the chart, changing the scale completely changes the effect the chart has on the viewer at first pass. Additionally, exponentially-scaled charts are much harder to derive specific details from.

Site Names

our final bar chart

With the scale of the bar chart built, one final design decision we had to make was regarding the alignment of the text within bar chart. We ended up displaying the text at a 45 degree angle because that simply made the text most readable. Rendering it vertically required too much room to display all of the characters for a given site and at the same time was too difficult to read.

List.js

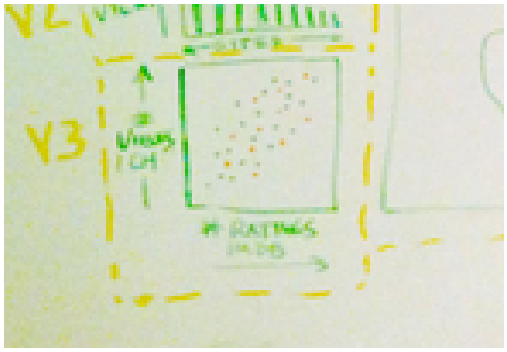
This is a simple and straightforward list for selection and filtering of film titles in the visualization. It was built using jquery. By default all titles are selected. Instead users can select multiple titles of interest and see the chart redrawn automatically. We had to include some conditional logic to enable the scatterplot to loop back around to the list and highlight the points selected in the scatterplot, but other than that, development here was pretty straight forward.

Early prototype *Our end result*

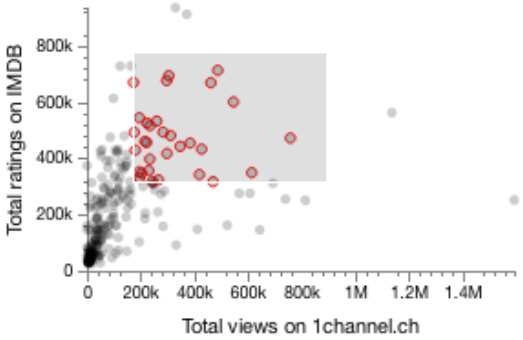
The only major implementation beyond our initial prototype was the building of the “select all”

button that basically enabled a “reset” of all the views in the visualization. Since selecting in the list filters the view, and unchecking the box would basically create an “empty” visualization, we decided to disallow users from unchecking select all.

Scatterplot.js



Our final sketch



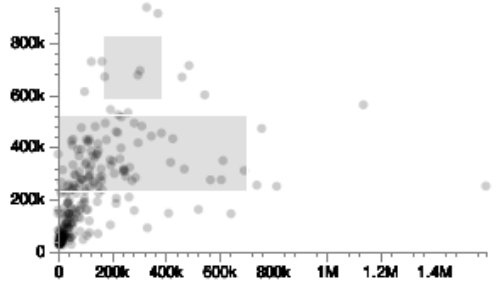
Our end result in d3

Overview

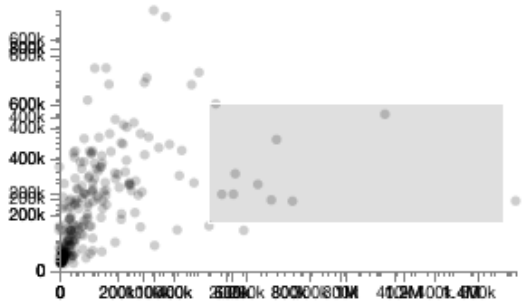
In our initial design process we decided we wanted to create a scatter plot that compared the number of ratings a movie received on IMDB with the number of times it was viewed on 1channel.ch. We felt this visualization would be important because these metrics most closely captured the level of engagement users of each community had with the films themselves. Plotting these values on a scatter plot most easily enabled an understanding of the relationship between these two metrics.

Axes in Peril

One of the most bugs we had to resolve came after the basics of the scatterplot were built. Because the scatter plot was refreshed every time data was refreshed in the visualization, the axes kept getting redrawn over and over again, becoming darker and darker with each update.



redrawing over the same axis



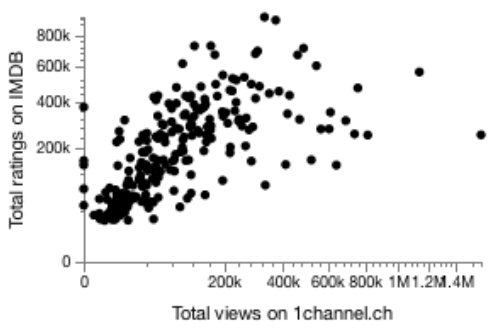
redrawing axis with different domain maximum

It took a long time to figure out the solution but ultimately the solution to the problem was two-fold--first, the maximum kept changing because the updateViews() call made in **main.js** redefined the maxViews and maxRatings values used for the domain with for the x and y axes. This was resolved by just initializing maxViews and maxRatings prior to calling updateViews().

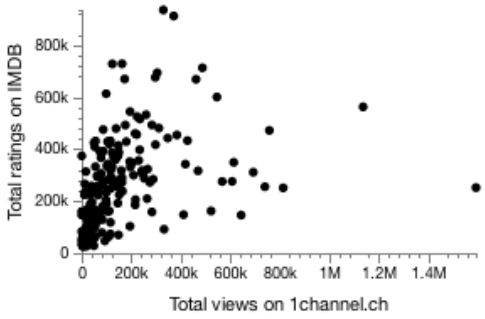
The second part of the solution was creating some logic so that the axis was only drawn once. The when the initial **scatter.js** file was loaded, we created variable called *axesDrawn* that was initialized to zero. Then, when the scatterplot is drawn for the first time, we used conditional logic to draw the axes only if *axesDrawn* == 0. After the first pass the value was then set to 1, and the axes were no longer redrawn over and over again. We also included the call to the brush function in this if statement block--this prevented calling the brushing function over and over again and preventing users from selecting data points after the first pass (you can see this problem in the example above).

Logarithmic vs. Linear Scale

One of the questions that arose during development was whether or not to use an exponential scale. Because our range of values was so large (the number of views for a movie, for example, could range from zero to 1.6 million), many of the points in the scatterplot were scrunched next to the y axis. We explored the possibility of using an exponential in order to bring these points off the axis. Here is side by side of the exponential scale and the linear scale we tested out:



exponential scale

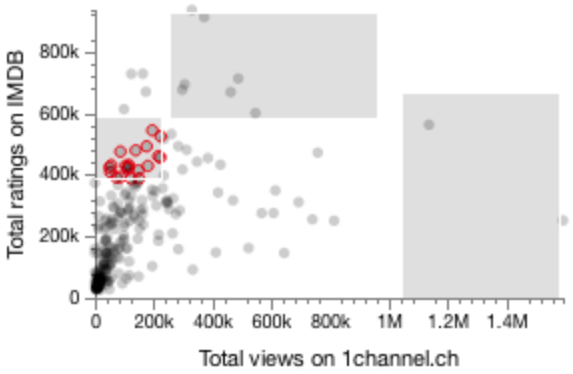


linear scale

One of the things that becomes clear from changing the scale is that the “lie factor” becomes a concern. If you look at the linearly-scaled rendering, the correlation between the number of views and the number of ratings on IMDB is apparent but not particularly strong. However, if we scale the axes exponentially, the relationship between the two variables appears much stronger even though it hasn’t changed a bit. Based on these observations we decided that the most appropriate scale to use was the one that spoke truth to the relationship between the two variables--the linear scale.

Brushing Implementation

After the scaling issues were resolved and the next big challenge was building the brushing functionality into the visualization. Using a tutorial and code we gathered from this site, simply adding the brushing functions and a few lines of code to call the brushing function was enough to get us up and running... initially. Soon it became clear that every time the views updated it was impossible to make a new selection using the brushing feature (see image). After some consternation trying to figure this piece out, it turns out the .call() method that called our brushing function was being called anew each time the views changed. This prevented the user from then being able to select any points.



an example of the brushing issue

In order to resolve this issue, it turns out that all that was needed was that we didn't have to keep redefining the brushing function with each time **main.js**. Simply defining it once in the same if block that drew the axes was enough. This was enough to kick off the brush event handler built-in to d3 that handles all the brushing events.

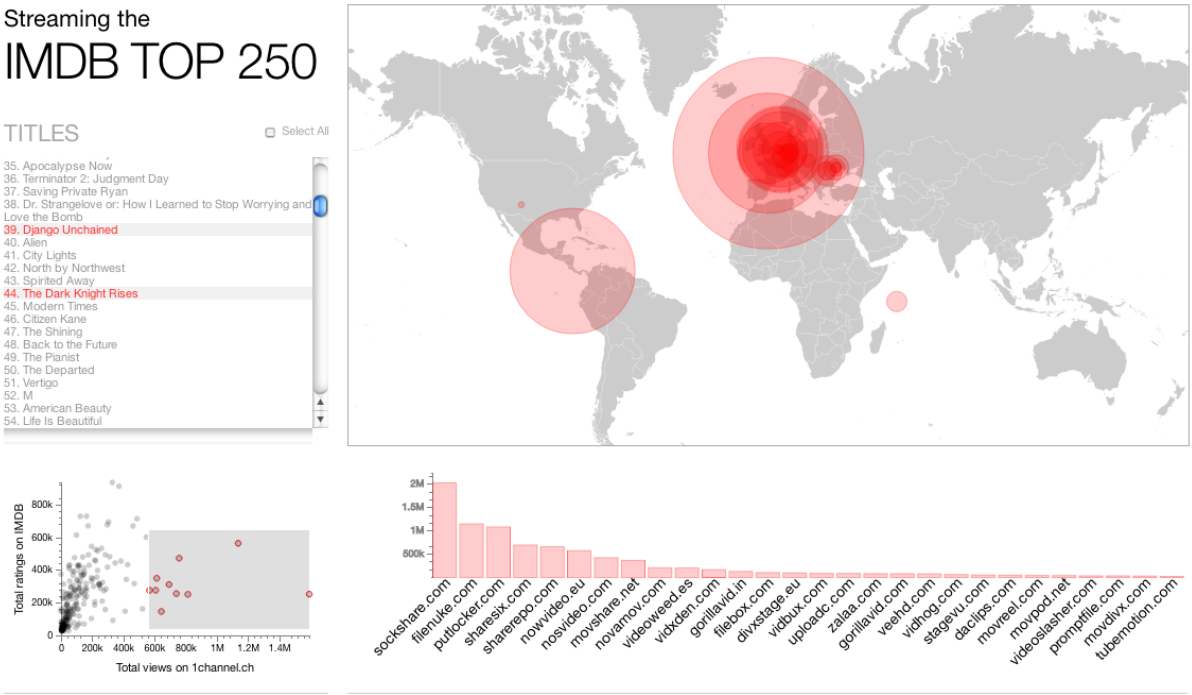
Conclusions

One of the reasons why we wanted to study content-indexing sites, and in particular, 1channel.ch, is because we knew so little about which sites were involved in making this operation work and how big of a role each site played in serving all of the content to viewers. After going through the scraping and visualization process, we can definitely say we have a much better grip on how the operation works, who the key players are, and who is delivering the content.

At the same time, when we look at our research questions, it seems like the answers to those questions don't tell the whole story of the data we have visualized. Below is a quick review of our research questions and some other major conclusions we can draw from our visualization.

Our Research Questions

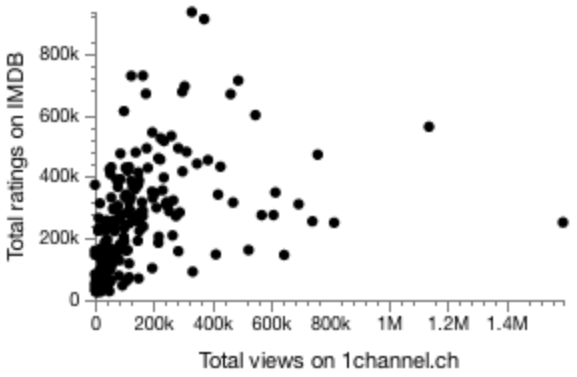
1. Using IMDB's top 250 as the base group of films, what are the 10 most viewed movies on 1channel.ch?



One thing our visualization did not do a good job of showing is what the top 10 most viewed movies in the dataset are. Since we focused primarily on presenting sites and views, in order to view the top 10 movies, users would have to use the scatter plot and list sections to select these titles manually (see the image above). In the future, adding some functionality to call this information out, or to sort the list by most viewed, would be a great idea.

One interesting thing that does come through is that most of the top 10 most viewed movies either a) have a cult following, like the *Lord of the Rings* trilogy, or b) are recent blockbusters (*The Avengers, the Dark Knight Rises, Harry Potter*). One of the most fascinating things is that *Django Unchained* was the #1 viewed movie with about 1.6 million views.

2. How does the number of votes for a movie on IMDB compare to the number of views of that movie on 1channel.ch?

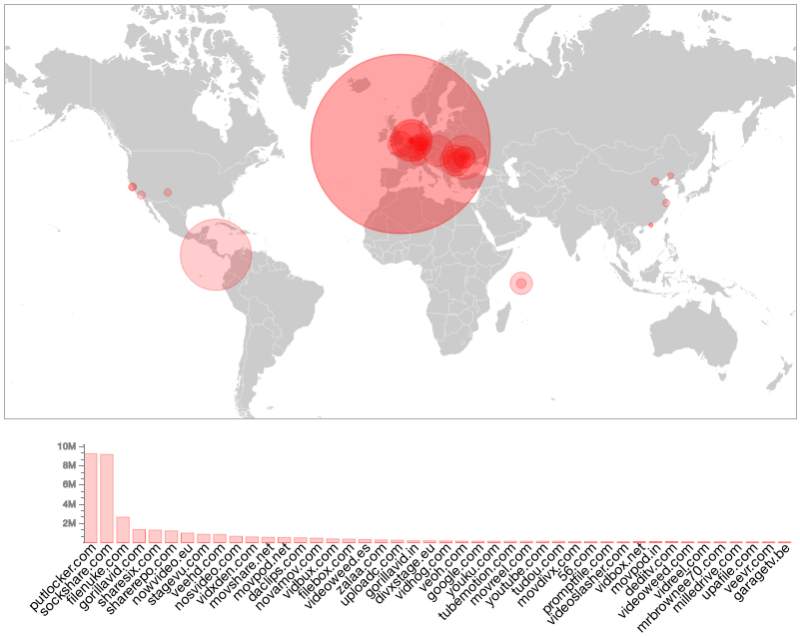


our scatter plot

As we mentioned earlier in this document, one of the things we were interested in exploring was the relationship between the number of ratings on IMDB for a movie compared to the number of views it received on 1channel.ch. We wanted to explore this information because these pieces of information were the closest proxies on each site for actual user engagement with the film.

After plotting everything out, we discovered that there is indeed a relationship, but that it is not a particular strong one. As the number of ratings goes up, it is true that there is a higher likelihood that that movie will be viewed. At the same time, there are a ton of outliers--there are a bunch of films that have been very popular with the 1channel.ch audience but not popular with users on IMDB. Part of the reason for this could be that some of the movies that are very highly rated have release dates from at least a few years ago. On the other hand, the two most popular movies, *Django Unchained* and *The Dark Knight Rises*, have yet to receive many ratings on IMDB. They also have yet to make it out on video. Coincidence? Probably not.

3. What sites serve the most films and where are they located?



Our visualization truly shines when it comes to answering this question. Using the bar chart, viewers have little difficulty understanding which site is providing the most views to users on 1channel.ch. If they feel so inclined, they can even use the map to complete this task. The only caveat is that the two largest servers of content, putlocker.com and sockshare.com, both serve traffic from the UK and thus overlap. Not only that, they are typically neck and neck when it comes to traffic, so the viewer typically only sees one circle when there really are two (the tooltips/mouseover on the bar come in handy here).

As we can see from the screenshot--hosts on 1channel.ch really have a global presence. There are even hosts that exist in the Seychelles, the island nation that now also hosts thepiratebay.se. Even with the global presence of all these content providers, it is abundantly clear that most of the traffic that is being served from these sites is coming from Europe, and within Europe (excepting the two largest providers), Eastern Europe in particular. The two largest providers, putlocker.com and sockshare.com, both appear to be serving their traffic from the United Kingdom.

4. What countries serve the most films?

Looking back to the previous section's screenshot, it is clear that our visualization can only partially answer this question. It is quite easy to tell what the top two countries are in terms of traffic--the host of the two leading sites is the UK, and the third place site is hosted in Costa Rica. But beyond those two, it is incredibly difficult to tell because of the lack of aggregation by

country.

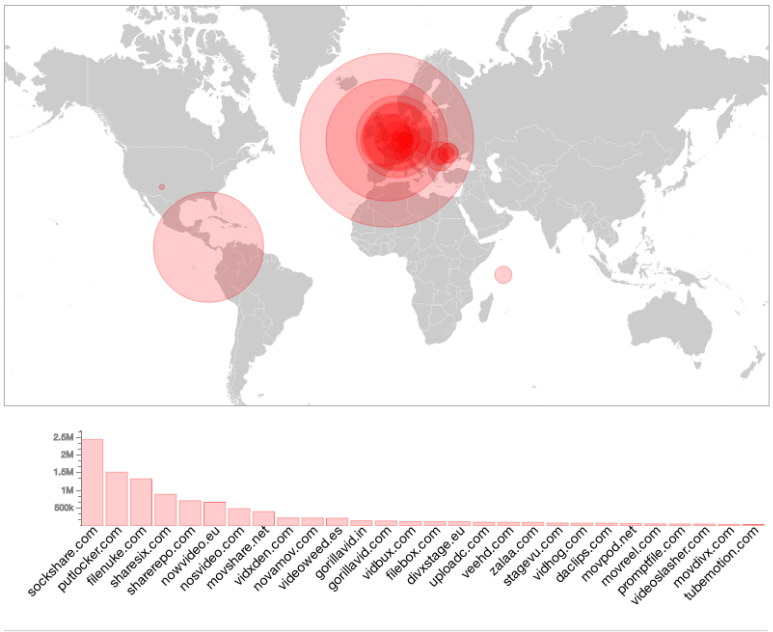
In order to fix this in the future, one thing that we could do was present a control that would enable toggling to display views by country or views by site. We initially outlined this in our spec but it was something we had to jettison because of time.

Questions That Were Answered but Not Asked

1. Do all the sites share the burden of serving these views or do a few key players dominate?

One of the most interesting answers we got out of our data was that almost all of the views provided in the dataset came from two hosts--putlocker.com and sockshare.com. There are only a couple of big players in the “marketplace” so-to-speak of the online streaming world, at least as it pertains to 1channel.ch, and there are a bunch of minor league-ers out there. The map and the bar chart both easily enable a viewer to complete this task and to draw the same conclusion.

2. Does the proportion of views for each site change using different slices of the data?



If a user spends a few seconds messing around with the scatter plot brushing, they will discover that depending on the movies that are selected, the marketshare of the individual seems to change pretty dramatically. If you look at those movies with over 500k views on 1channel.ch (above image), the domination by sockshare and putlocker diminishes significantly. They are still leaders, no question about that, but they don’t control the market like they do across the

entire 250-movie dataset. One reason why this is the case could be that the smaller players are doing a good job of getting their hosts in front of viewers of the content that is popular, but for the lesser viewed titles, the two top players are cleaning up.

Brainstorming / Discovery

3/19/13 - RA
Since a large portion of our project will be map based, I spent some time searching for SVG maps. In particular, I would like to use a map that can eventually be mapped to a sphere. Ideally we can find an up-to-date Equirectangular projection map that will look good flat and have added potential for PJ3.

Setup
3/7/13 - RA
Created Github repository for version control
Here is a nice cheat sheet for the command line.
Set Sublime Text 2 as the default GIT editor instructions here.

3/16/-3/17/13 - JF
I had some issues setting up github at first. I installed git but the osxkeychain helper was fatally erroring out. I decided to just use normal https login, but that provided 403 errors that I couldn't resolve. After some googling I thought it was that I didn't have write access to the git repository, but it turns out it sometimes HTTPS provides errors that are hard to decode. After I spoke with Rob, he gave me write access (which wouldn't have mattered) and he told me to use SSH. The following morning (3/17) I was able to get that configured in under 15 minutes. Now I can commit my changes!

3/17/13 - RA
I was able to pull the files that Jeff had committed and began running some initial tests. Jeff committed a few more changes and I tried to pull again. This time, I got the following error messages.

```
error: Your local changes to the following files would be overwritten by
merge:
      scraping/ch1.pyc
      scraping/imdb.pyc
      scraping/unescape.pyc
error: The following untracked working tree files would be overwritten by
merge:
      scraping/log.txt
```

I had a nice adventure trying to make sense of the conflicts. I'm new to Github so this is to be expected. This turorial was very helpful. I discarded all local changes and created a gitingnore file to prevent this in the future. Fortunately Github provides one preconfigured for python. Instructions on how to install it are here. I have already committed this "project level" gitignore file so no further action should be needed in that regard. I would, however, recommend installing a gitignore at the global level as well. I created one which can be found here. Save it

somewhere to disc and hide it in the home folder using

```
cp gitignore_global ~/.gitignore_global
```

Then add this file to your cross-repository configuration, by running

```
git config --global core.excludesfile ~/.gitignore_global
```

This should prevent these sort of errors moving forward. Additionally we need to be careful in the future to not commit compiled or generated files. One easy way to handle this would be for all log files to use the .log extension and data files (.json for example) could be saved in a folder that is never added to the project.

3/18/2013 - RA
Installed Atlassian SourceTree to manage GIT, based on some recommendations from co-worker.

Data Collection

3/16/2013 - JF
I compartmentalized the functions for IMDB, 1channel, and cleaning up html entity defs into different files. I also have a main file that just runs the functions from the various modules to build the json object. The json object is semi-complete, most of the imdb stuff is in there so far.

So far I have the following files compartmentalized into modules:

1. **imdb.py** provides a series of IMDB functions that assign the IMDB values to that section of the object. This function also needs to be called first because it builds the root level of the dictionary using the names of the movies as the keys.
2. **ch1.py** provides a series of 1channel functions for building the 1channel section of the object. This is semi-complete as of this writing, I still need to write in assignment and include the geoip functions.
3. **unescape.py** provides the Frederik Lundh unescape function for unescaping html entities so that they appear as unicode. Some characters are still showing as escaped so I need to do some further investigation there.
4. **geoip.py** geoip capture information

3/17/2013 - JF
I finally was able to build the IMDB section of the object, I am currently running the script to scrape the data for the Top 250 on IMDB. Once it finishes I am going to write the dictionary I am building to a file so that I don't have to retrieve it for here on out.

Just finished, I pickled the dictionary for later use. It is stored as “mov_obj.pickle” in our repository. To load it, that needs to be done is the unpickling:

```
f = open(‘mov_obj.pickle’, ‘r’)
mov_obj = pickle.load(f)
f.close()
```

type(mov_obj) will evaluate to “type dict” or some such thing

---
The next section I have to build is the 1channel.ch functions. Most of these are already built (searching, etc.) but I need to implement the assignment into the object, and incorporate the geoip stuff. I also need to grab the MaxMind geolite free city database so that I can incorporate city information into our object.

3/24/13 JF

I have finally finished scraping the necessary data from 1channel. It ended up being a pretty manual process because of inconsistencies between IMDB and the information in 1channel’s database. I ended up having to manually enter conditional logic for about 20% of the titles in 1channel because of various issues. In order to explain how the logic worked, and why I had to add it, I should probably explain the architecture of the script first and the evolution of our dataset structure.

3/24/2013 - RA
I created a script that could count items in the data set. Here are some interesting points that I have found that will control the direction of our visualization.

250 Films
45 Unique sites
15 Unique countries
935,000 Ratings on #1 IMDB film
1,611,000 Views of the #1 1Channel film

3/20/2013 - RA
Started searching for relevant map visualizations created in D3 that could be used as examples and or a starting point for the project.
http://mbostock.github.com/d3/talk/20111018/choropleth.html
http://bl.ocks.org/mbostock/4330486

http://bl.ocks.org/mbostock/3734333
http://bl.ocks.org/mbostock/2374239

Visualization Creation
3/25/13 - RA
Created initial site structure with new dimensions

Basic Overview

3/27/13 - RA
I spend quite a bit of time creating the functions to load and parse the JSON data. I wanted to transform the source JSON into arrays that D3 could easily ingest. The overall site is architected so that the data, or subset thereof, will be passed into the "parse" function. This function will automatically reformat the data and then call "updateViews" Each chart has its own update method which will cause the sections to be re-rendered with the new data.

main.js
- Loads JSON data into memory stores unmodified as dataSource and calls parse()
- Parse function creates array of Site objects and Array of Film objects
    - These will be used for all charts
    - Parse is set-up so that it can be reused again with a filtered subset of the original JSON object
    - Parse complete - calls initList() updateMap, updateScatter(), updateBar()

list.js
- populated by parse
- references **films** array
- creates a node for each film object
- filtering creates a list of indices - which are then filtered from films array
- parse is then called again on this subset which in-turn updates the map and bar chart

map,js
- populated by parse
- refereces **sites** array
- creates element for each site object
- mouse over map elements highlights current dims others
- finds matching element in bar chart by the site's name and manages highlighting of bar chart as well

bar.js
- populated by parse

- refereces **sites** array
- creates element for each site object
- mouse over bar elements highlights current dims others
- finds matching element in map by the site's name and manages highlighting of bar chart as well

scatter.js

- populated by parse
- remains consistent regardless of other interactions
- refereces **films** array
- creates element for each site object
- brushing elements highlights current dims others
- creates an array storing indices of selected points
- indicies are passed to list to automatically select
- selection in list calls parse with subset of data and then updates bar and map

Page Layout

2/20/2013 - RA

Started initial page layout and visual designs. Began considering map type to use. Miller seemed the most familiar to me in terms of how I picture a map. We had already decided to use area to encode our data, so choosing a map with accurate depiction of area was not important.
https://github.com/mbostock/d3/wiki/Geo-Projections

I crafted an initial layout in photoshop at 1280x720 resolution to see how it actually would work.

I wasn't really happy with the first iteration and thought that the scatterplot (V3) and bar chart (V2) were too small. I started looking deeper into these two sections to determine how much space is needed. After determining the approximate number of data points I searched for charts online that had a similar data density and placed them in the current layout.

It looks like there will be a lot of wasted space with the map. Also I am not comfortable with the size of the bar chart. How will we label the 45 sites at that size?

Redesign 3/24

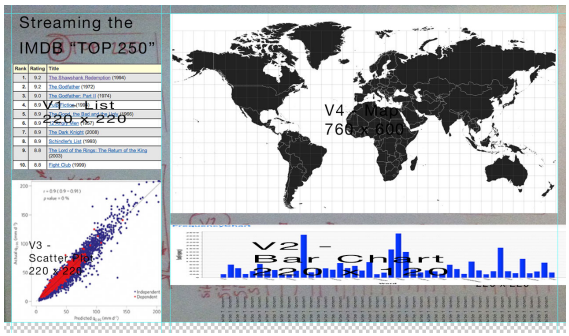
I like the direction this is moving.

The bar chart at the bottom

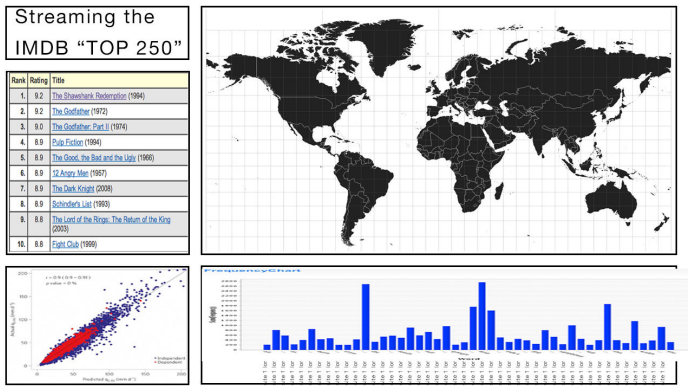
could now show all 45 data points for shares by site. Perhaps these could be encoded with color by country, since there are only 15. TBD. This also creates a nice space for sorting and filtering within V2.

Control of zoom and position on the globe may be irrelevant. Probably going to lose this. Project info can move to the bottom. Additional info can appear as a tool-tip or modal. This will allow for wider V1 and V3.

Here is the potential design after a quick layout in photoshop.



Properly sized version of above.



Page Layout

I crafted an initial layout in photoshop at 1280x720 resolution to see how it actually would work.

I wasn't really happy with the first iteration and thought that the scatterplot (V3) and bar chart (V2) were too small. I started looking deeper into these two sections to determine how much space is needed. After determining the approximate number of data points I searched for charts online that had a similar data density and placed them in the current layout.

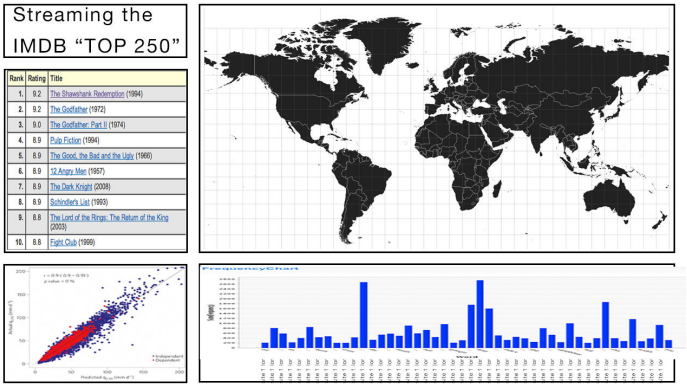
It looks like there will be a lot of wasted space with the map. Also I am not comfortable with the size of the bar chart. How will we label the 45 sites at that size?

Redesign 3/24
I like the direction this is moving.
The bar chart at the bottom could now show all 45 data points for shares by site. Perhaps these could be encoded with color by country, since there are only 15. TBD. This also creates a nice space for sorting and filtering within V2.

Control of zoom and position on the globe may be irrelevant. Probably going to lose this. Project info can move to the bottom. Additional info can appear as a tool-tip or modal. This will allow for wider V1 and V3.

Here is the potential design after a quick layout in photoshop.

Properly sized version of above.



README.txt content

Each chart has its own js file. There is a single html and single css.
Main.js manages the data for the entire visualization and updates the views accorgingly

I spend quite a bit of time creating the functions to load and parse the JSON data. I wanted to transform the source JSON into arrays that D3 could easily ingest. The overall site is architected so that the data, or subset thereof, will be passed into the “parse” function. This function will automatically reformat the data and then call “updateViews” Each chart has its own update method which will cause the sections to be re-rendered with the new data.

When developing locally, note that your browser may enforce strict permissions for reading files out of the local file system. **If you use d3.xhrlocally (including d3.json et al.), you must have a local web server.** For example, you can run Python's built-in server:

```
python -m SimpleHTTPServer 8888 &
```

Once this is running, go to http://localhost:8888/.