Analyzing IMDb Data The Intended Way, with R and ggplot2

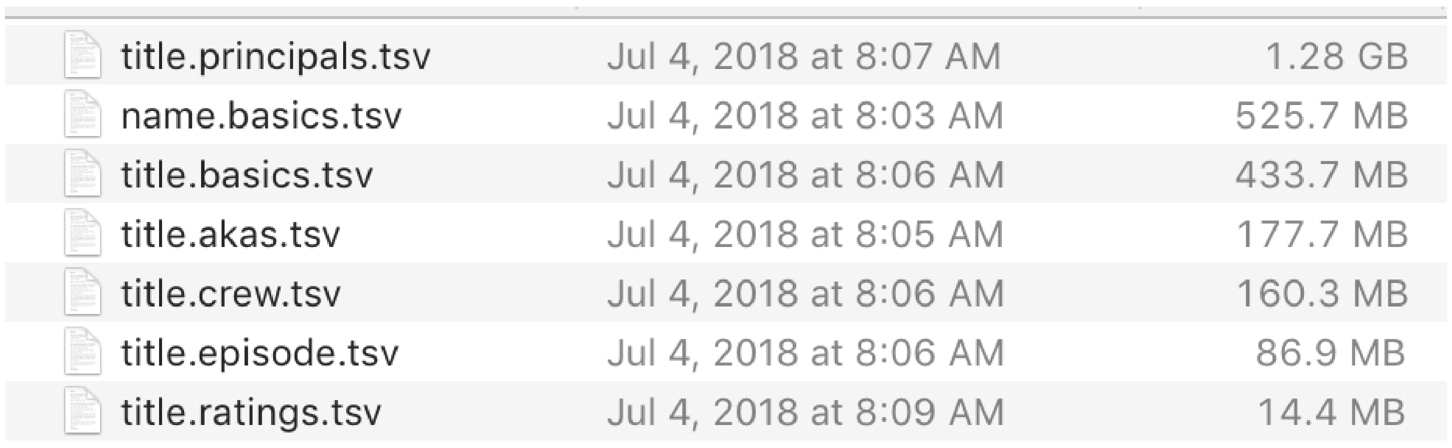
July 16, 2018 · 11 min read · Data Visualization, Data Science, Big Data

IMDb, the Internet Movie Database, has been a popular source for data analysis and visualizations over the years. The combination of user ratings for movies and detailed movie metadata have always been fun to play with.

There are a number of tools to help get IMDb data, such as IMDbPY, which makes it easy to programmatically scrape IMDb by pretending it's a website user and extracting the relevant data from the page's HTML output. While it *works*, web scraping public data is a gray area in terms of legality; many large websites have a Terms of Service which forbids scraping, and can potentially send a DMCA take-down notice to websites redistributing scraped data.

IMDb has data licensing terms which forbid scraping and require an attribution in the form of a **Information courtesy of IMDb (http://www.imdb.com). Used with permission.** statement, and has also DMCAed a Kaggle IMDb dataset to hone the point.

However, there is good news! IMDb publishes an official dataset for casual data analysis! And it's now very accessible, just choose a dataset and download (now with no hoops to jump through), and the files are in the standard TSV format.



title.principals.tsv	Jul 4, 2018 at 8:07 AM	1.28 GB
name.basics.tsv	Jul 4, 2018 at 8:03 AM	525.7 MB
title.basics.tsv	Jul 4, 2018 at 8:06 AM	433.7 MB
title.akas.tsv	Jul 4, 2018 at 8:05 AM	177.7 MB
title.crew.tsv	Jul 4, 2018 at 8:06 AM	160.3 MB
title.episode.tsv	Jul 4, 2018 at 8:06 AM	86.9 MB
title.ratings.tsv	Jul 4, 2018 at 8:09 AM	14.4 MB

The uncompressed files are pretty large; not "big data" large (it fits into computer memory), but Excel will explode if you try to open them in it. You have to play with the data *smartly*, and both R and ggplot2 have neat tricks to do just that.

First Steps

R is a popular programming language for statistical analysis. One of the most popular series of external packages is the `tidyverse` package, which automatically imports the `ggplot2` data visualization library and other useful packages which we'll get to one-by-one. We'll also use `scales` which we'll use later for prettier number formatting. First we'll load these packages:

```
library(tidyverse)
library(scales)
```

And now we can load a TSV downloaded from IMDb using the `read_tsv` function from `readr` (a tidyverse package), which does what the name implies, at a much faster speed than base R (+ a couple other parameters to handle data encoding). Let's start with the `ratings` file:

```
df_ratings <- read_tsv('title.ratings.tsv', na = "\\N", quote = '')
```

We can preview what's in the loaded data using `dplyr` (a tidyverse package), which is what we'll be using to manipulate data for this analysis. dplyr allows you to pipe commands, making it easy to create a sequence of manipulation commands. For now, we'll use `head()`, which displays the top few rows of the data frame.

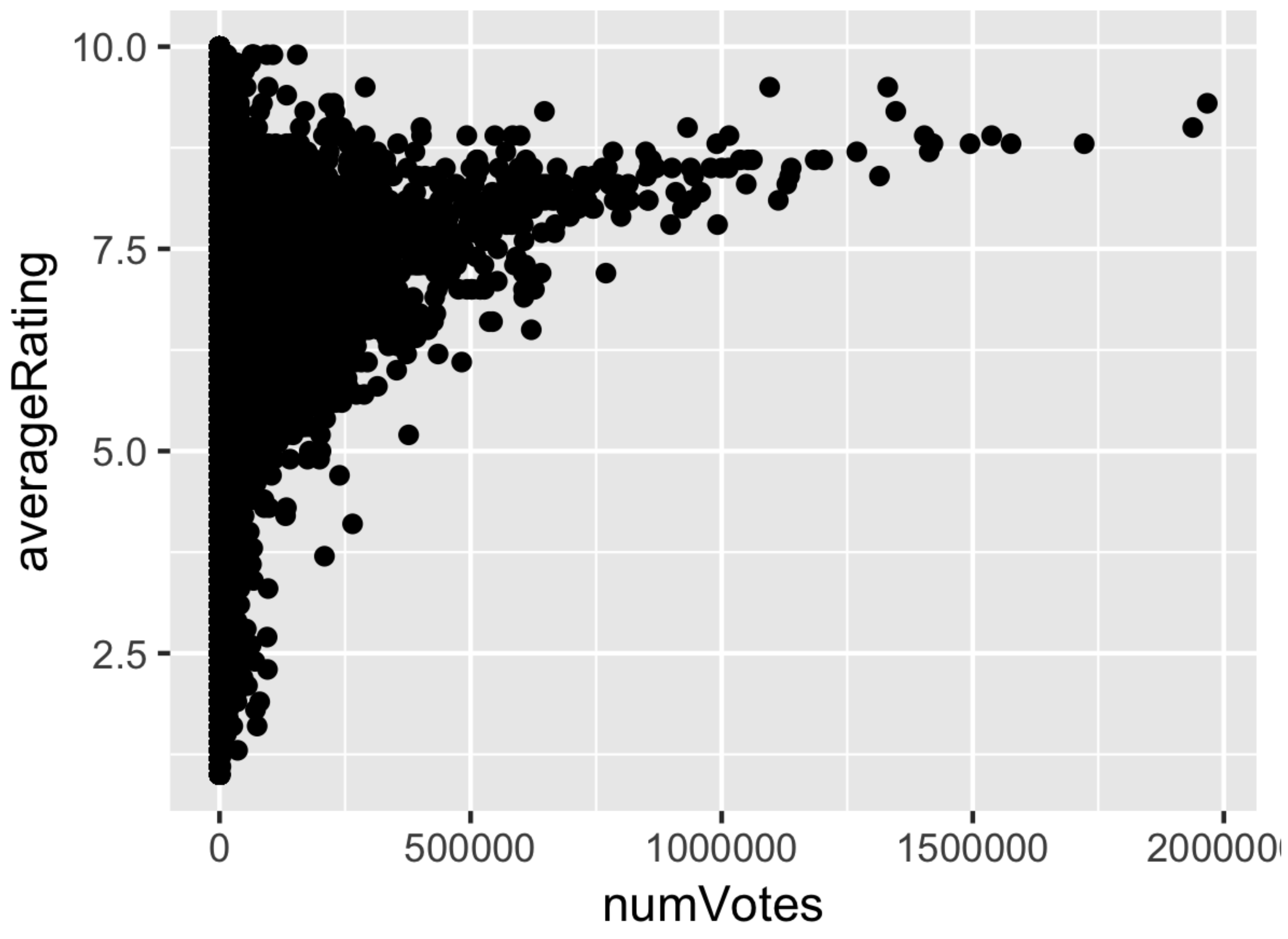
```
df_ratings %>% head()
```

tconst <chr>	averageRating <dbl>	numVotes <int>
tt0000001	5.8	1385
tt0000002	6.5	162
tt0000003	6.6	972
tt0000004	6.4	98
tt0000005	6.2	1666
tt0000006	5.6	86

Each of the **873k rows** corresponds to a single movie, an ID for the movie, its average rating (from 1 to 10), and the number of votes which contribute to that average. Since we have two numeric variables, why not test out ggplot2 by creating a scatterplot mapping them? ggplot2 takes in a data frame and names of columns as aesthetics, then you specify what type of shape to plot (a "geom"). Passing the plot to `ggsave` saves it as a standalone, high-quality data visualization.

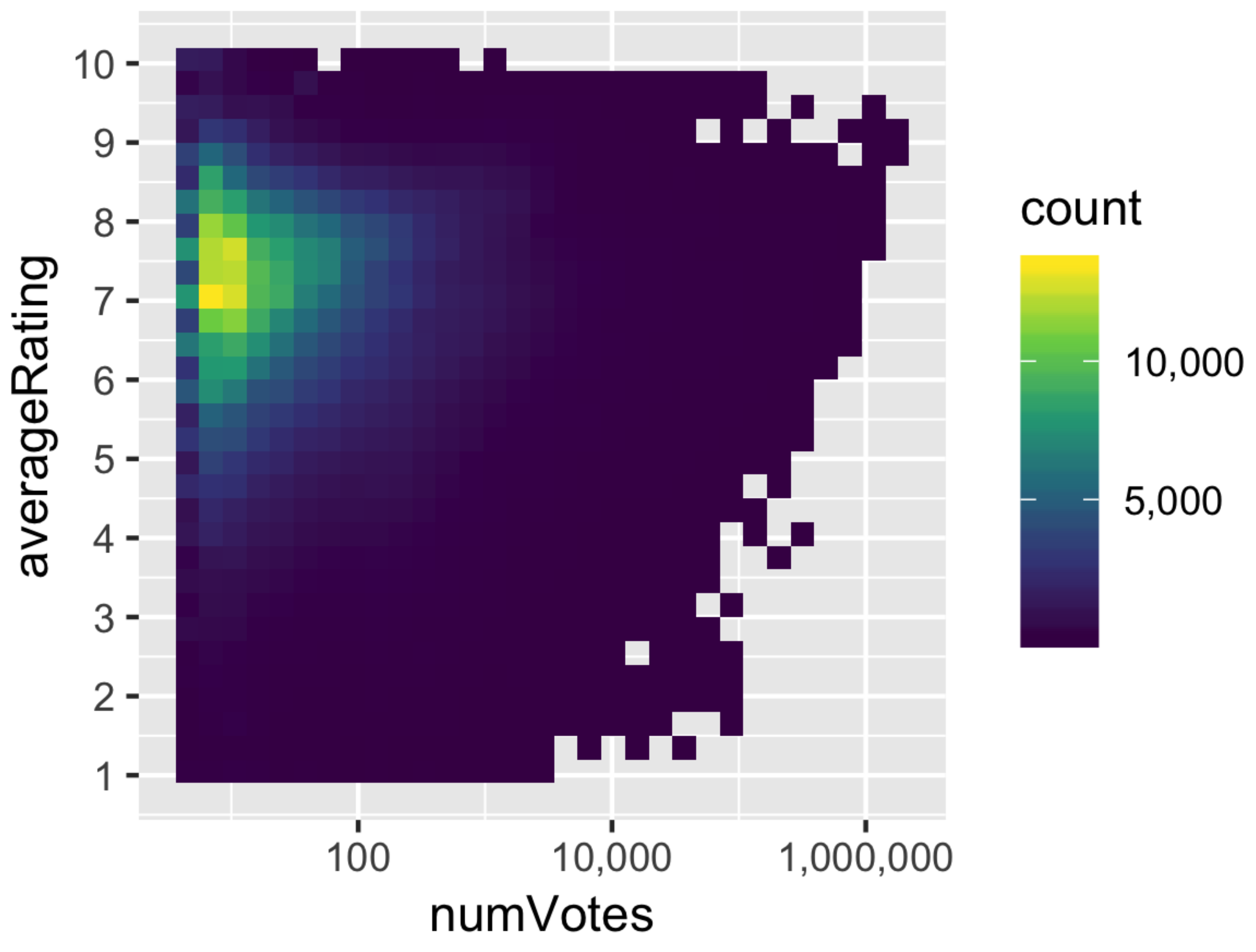
```
plot <- ggplot(df_ratings, aes(x = numVotes, y = averageRating)) +
          geom_point()

ggsave("imdb-0.png", plot, width = 4, height = 3)
```



Here is nearly *1 million* points on a single chart; definitely don't try to do that in Excel! However, it's not a *useful* chart since all the points are opaque and we're not sure what the spatial density of points is. One approach to fix this issue is to create a heat map of points, which ggplot can do natively with `geom_bin2d`. We can color the heat map with the viridis colorblind-friendly palettes just introduced into ggplot2. We should also tweak the axes; the x-axis should be scaled logarithmically with `scale_x_log10` since there are many movies with high numbers of votes and we can format those numbers with the `comma` function from the `scales` package (we can format the scale with `comma` too). For the y-axis, we can add explicit number breaks for each rating; R can do this neatly by setting the breaks to `1:10`. Putting it all together:

```
plot <- ggplot(df_ratings, aes(x = numVotes, y = averageRating)) +
          geom_bin2d() +
          scale_x_log10(labels = comma) +
          scale_y_continuous(breaks = 1:10) +
          scale_fill_viridis_c(labels = comma)
```



Not bad, although it unfortunately confirms that IMDb follows a Four Point Scale where average ratings tend to fall between 6 — 9.

Mapping Movies to Ratings

You may be asking "which ratings correspond to which movies?" That's what the `tconst` field is for. But first, let's load the title data from `title.basics.tsv` into `df_basics` and take a look as before.

```
df_basics ← read_tsv('title.basics.tsv', na = "\\N", quote = '')
```

tconst <chr>	titleType <chr>	primaryTitle <chr>	originalTitle <chr>
tt0000001	short	Carmencita	Carmencita
tt0000002	short	Le clown et ses chiens	Le clown et ses chiens
tt0000003	short	Pauvre Pierrot	Pauvre Pierrot
tt0000004	short	Un bon bock	Un bon bock
tt0000005	short	Blacksmith Scene	Blacksmith Scene
tt0000006	short	Chinese Opium Den	Chinese Opium Den

isAdult <int>	startYear <int>	endYear <chr>	runtimeMinutes <int>	genres <chr>
0	1894	NA	1	Documentary,Short
0	1892	NA	5	Animation,Short
0	1892	NA	4	Animation,Comedy,Romance
0	1892	NA	NA	Animation,Short
0	1893	NA	1	Short
0	1894	NA	1	Short

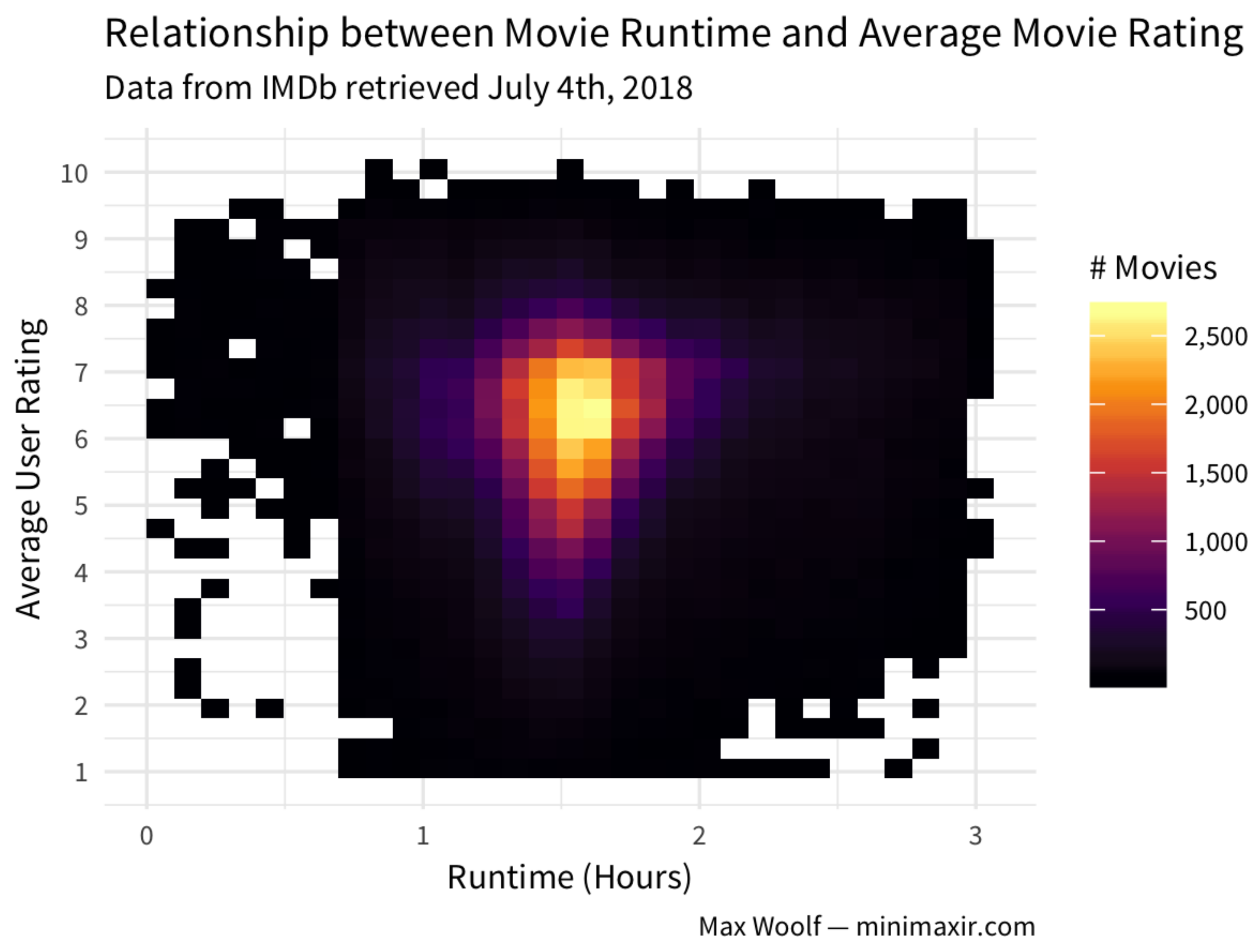
We have some neat movie metadata. Notably, this table has a `tconst` field as well. Therefore, we can *join* the two tables together, adding the movie information to the corresponding row in the rating table (in this case, a left join is more appropriate than an inner/full join)

```
df_ratings ← df_ratings %>% left_join(df_basics)
```

Runtime minutes sounds interesting. Could there be a relationship between the length of a movie and its average rating on IMDb? Let's make a heat map plot again, but with a few tweaks. With the new metadata, we can `filter` the table to remove bad points; let's keep movies only (as IMDb data also contains *television show data*), with a runtime < 3 hours, and which have received atleast 10 votes by users to remove extraneous movies). X-axis should be tweaked to display the minutes-values in hours. The fill viridis palette can be changed to another one in the family (I personally like `inferno`).

More importantly, let's discuss plot theming. If you want a minimalistic theme, add a `theme_minimal` to the plot, and you can pass a `base_family` to change the default font on the plot and a `base_size` to change the font size. The `labs` function lets you add labels to the plot (which you should *always* do); you have your `title`, `x`, and `y` parameters, but you can also add a `subtitle`, a `caption` for attribution, and a `color`/`fill` to name the scale. Putting it all together:

```
plot ← ggplot(df_ratings %>% filter(runtimeMinutes < 180, titleType == "movie", numVotes ⩾ 10), aes(x =
          geom_bin2d() +
          scale_x_continuous(breaks = seq(0, 180, 60), labels = 0:3) +
          scale_y_continuous(breaks = 0:10) +
          scale_fill_viridis_c(option = "inferno", labels = comma) +
          theme_minimal(base_family = "Source Sans Pro", base_size = 8) +
          labs(title = "Relationship between Movie Runtime and Average Mobie Rating",
               subtitle = "Data from IMDb retrieved July 4th, 2018",
               x = "Runtime (Hours)",
               y = "Average User Rating",
               caption = "Max Woolf — minimaxir.com",
               fill = "# Movies")
```

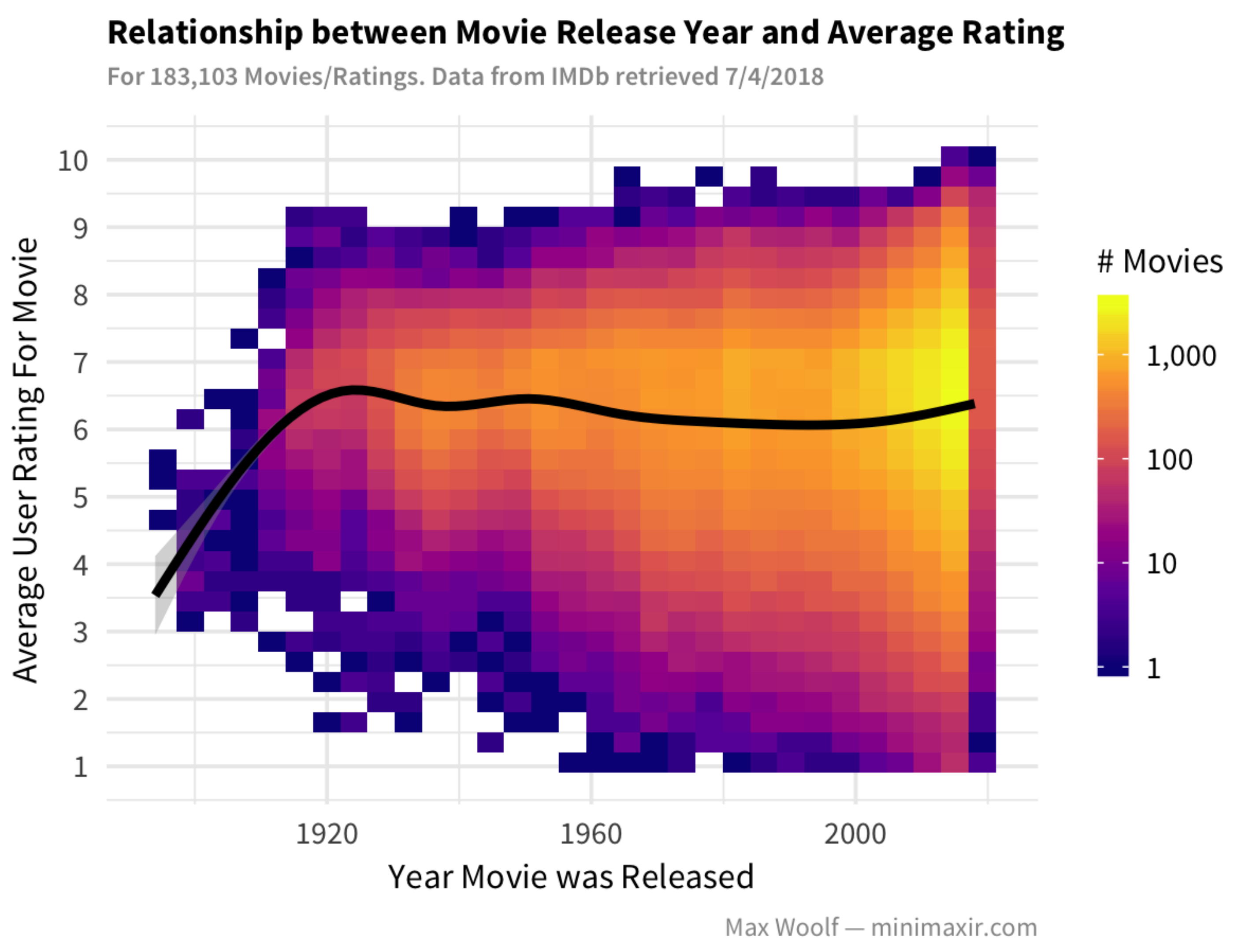


Now that's pretty nice-looking for only a few lines of code! Albeit unhelpful, as there doesn't appear to be a correlation.

(Note: for the rest of this post, the theming/labels code will be omitted for convenience)

How about movie ratings vs. the year the movie was made? It's a similar plot code-wise to the one above (one perk about `ggplot2` is that there's no shame in reusing chart code!), but we can add a `geom_smooth`, which adds a nonparametric trendline with confidence bands for the trend; since we have a large amount of data, the bands are very tight. We can also fix the problem of "empty" bins by setting the color fill scale to logarithmic scaling. And since we're adding a black trendline, let's change the viridis palette to `plasma` for better contrast.

```
plot <- ggplot(df_ratings %>% filter(titleType == "movie", numVotes >= 10), aes(x = startYear, y = averag
          geom_bin2d() +
          geom_smooth(color="black") +
          scale_x_continuous() +
          scale_y_continuous(breaks = 1:10) +
          scale_fill_viridis_c(option = "plasma", labels = comma, trans = 'log10')
```



Unfortunately, this trend hasn't changed much either, although the presence of average ratings outside the Four Point Scale has increased over time.

Mapping Lead Actors to Movies

Now that we have a handle on working with the IMDb data, let's try playing with the larger datasets. Since they take up a lot of computer memory, we only want to persist data we actually might use. After looking at the schema provided with the official datasets, the only really useful metadata about the actors is their birth year, so let's load that, but only keep both actors/actresses (using the fast `str_detect` function from `stringr`, another tidyverse package) and the relevant fields.

```
df_actors ← read_tsv('name.basics.tsv', na = "\\N", quote = '') %>%
                filter(str_detect(primaryProfession, "actor|actress"))  %>%
                select(nconst, primaryName, birthYear)
```

nconst <chr>	primaryName <chr>	birthYear <int>
nm0000001	Fred Astaire	1899
nm0000002	Lauren Bacall	1924
nm0000003	Brigitte Bardot	1934
nm0000004	John Belushi	1949
nm0000005	Ingmar Bergman	1918
nm0000006	Ingrid Bergman	1915

The principals dataset, the large 1.28GB TSV, is the most interesting. It's an unnested list of the credited persons in each movie, with an `ordering` indicating their rank (where `1` means first, `2` means second, etc.).

tconst <chr>	ordering <int>	nconst <chr>	category <chr>
tt0000001	1	nm1588970	self
tt0000001	2	nm0005690	director
tt0000001	3	nm0374658	cinematographer
tt0000002	1	nm0721526	director
tt0000002	2	nm1335271	composer
tt0000003	1	nm0721526	director

For this analysis, let's only look at the **lead actors/actresses**; specifically, for each movie (identified by the `tconst` value), filter the dataset to where the `ordering` value is the lowest (in this case, the person at rank `1` may not necessarily be an actor/actress).

```
df_principals <- read_tsv('title.principals.tsv', na = "\\N", quote = '') %>%
  filter(str_detect(category, "actor|actress")) %>%
  select(tconst, ordering, nconst, category) %>%
  group_by(tconst) %>%
  filter(ordering == min(ordering))
```

Both datasets have a `nconst` field, so let's join them together. And then join *that* to the ratings table earlier via `tconst`.

```
df_principals <- df_principals %>% left_join(df_actors)
df_ratings <- df_ratings %>% left_join(df_principals)
```

Now we have a fully denormalized dataset in `df_ratings`. Since we now have the movie release year and the birth year of the lead actor, we can now infer *the age of the lead actor at the movie release*. With that goal, filter out the data on the criteria we've used for earlier data visualizations, plus only keeping rows which have an actor's birth year.

```
df_ratings_movies <- df_ratings %>%
                        filter(titleType == "movie", !is.na(birthYear), numVotes >= 10) %>%
                        mutate(age_lead = startYear - birthYear)
```

tconst <chr>	averageRating <dbl>	numVo... <int>	titleType <chr>	primaryTitle <chr>
tt0111161	9.3	1967026	movie	The Shawshank Redemption
tt0468569	9.0	1938195	movie	The Dark Knight
tt1375666	8.8	1721909	movie	Inception
tt0137523	8.8	1576246	movie	Fight Club
tt0110912	8.9	1537603	movie	Pulp Fiction
tt0109830	8.8	1494533	movie	Forrest Gump
tt0120737	8.8	1420782	movie	The Lord of the Rings: The Fellowship of the Ring
tt0133093	8.7	1413319	movie	The Matrix
tt0167260	8.9	1403443	movie	The Lord of the Rings: The Return of the King
tt0068646	9.2	1346770	movie	The Godfather

category <chr>	primaryName <chr>	birthYear <int>	age_lead <int>
actor	Tim Robbins	1958	36
actor	Christian Bale	1974	34
actor	Leonardo DiCaprio	1974	36
actor	Brad Pitt	1963	36
actor	John Travolta	1954	40
actor	Tom Hanks	1956	38
actor	Elijah Wood	1981	20
actor	Keanu Reeves	1964	35
actor	Elijah Wood	1981	22
actor	Marlon Brando	1924	48

Plotting Ages

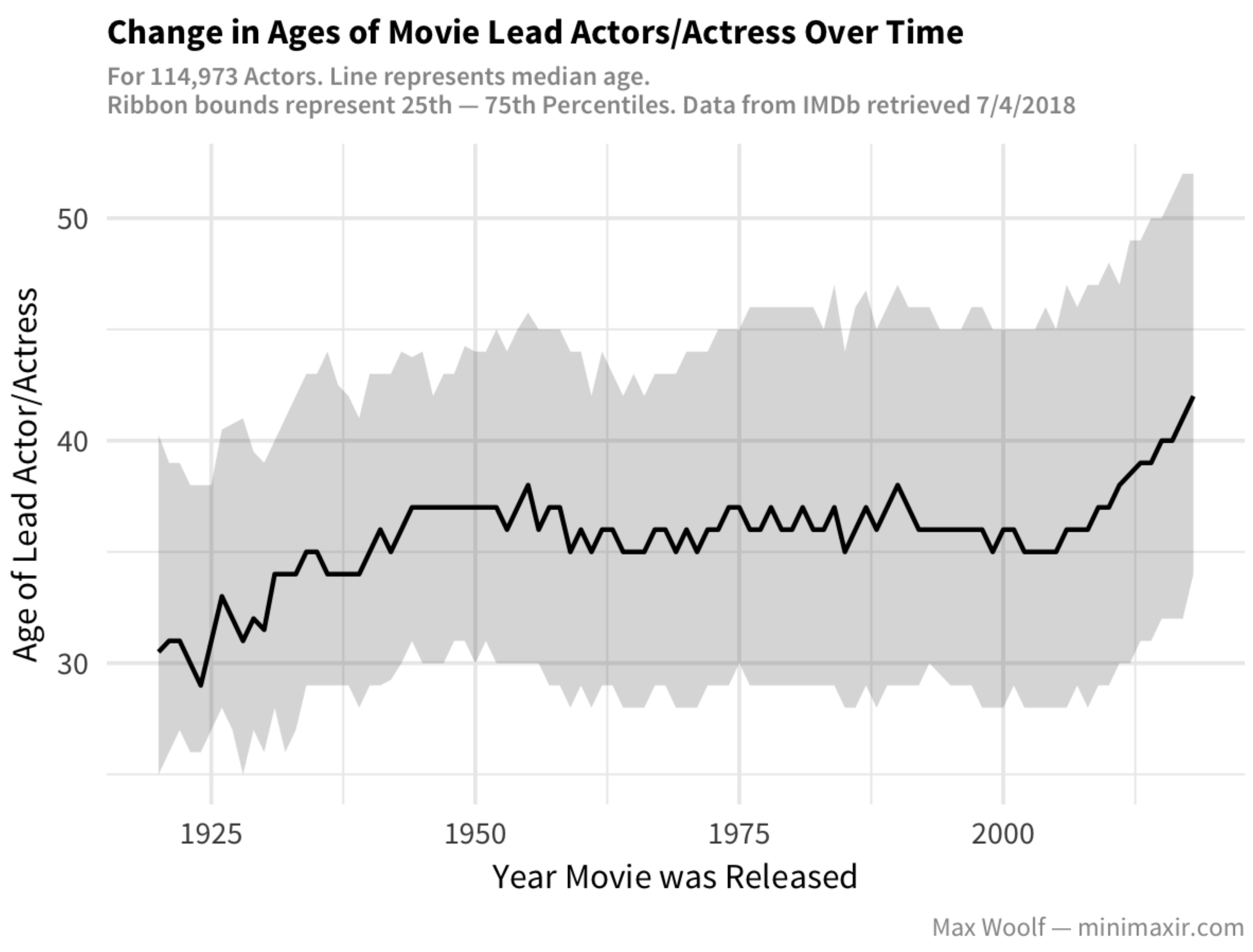
Age discrimination in movie casting has been a recurring issue in Hollywood; in fact, in 2017 a law was signed to force IMDb to remove an actor's age upon request, which in February 2018 was ruled to be unconstitutional.

Have the ages of movie leads changed over time? For this example, we'll use a ribbon plot to plot the ranges of ages of movie leads. A simple way to do that is, for each year, calculate the 25th percentile of the ages, the 50th percentile (i.e. the median), and the 75th percentile, where the 25th and 75th percentiles are the ribbon bounds and the line represents the median.

```
df_actor_ages ← df_ratings_movies %>%
                  group_by(startYear) %>%
                  summarize(low_age = quantile(age_lead, 0.25, na.rm=T),
                            med_age = quantile(age_lead, 0.50, na.rm=T),
                            high_age = quantile(age_lead, 0.75, na.rm=T))
```

Plotting it with ggplot2 is surprisingly simple, although you need to use different y aesthetics for the ribbon and the overlapping line.

```
plot ← ggplot(df_actor_ages %>% filter(startYear ⩾ 1920) , aes(x = startYear)) +
          geom_ribbon(aes(ymin = low_age, ymax = high_age), alpha = 0.2) +
          geom_line(aes(y = med_age))
```

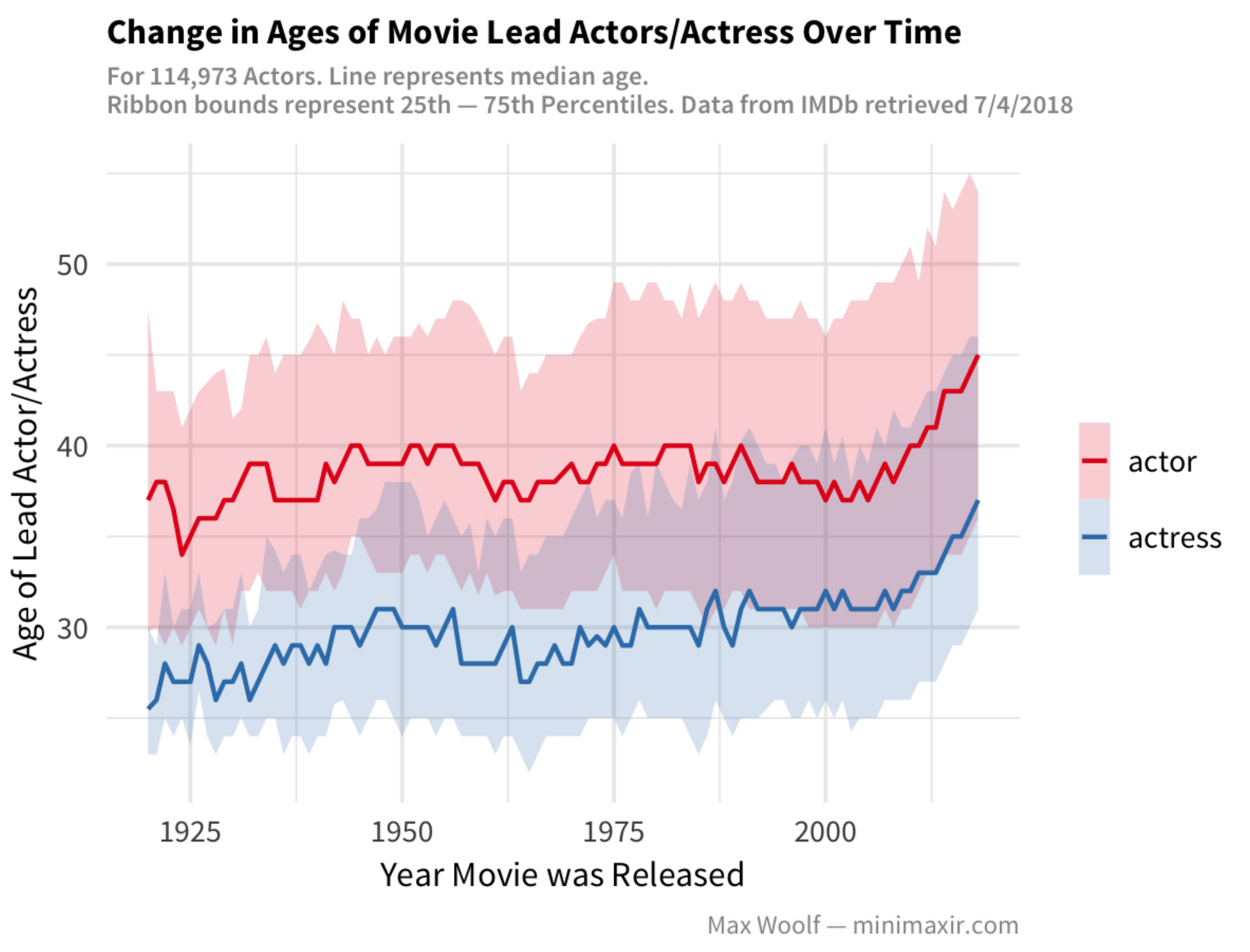


Turns out that in the 2000's, the median age of lead actors started to *increase*? Both the upper and lower bounds increased too. That doesn't coalesce with the age discrimination complaints.

Another aspect of these complaints is gender, as female actresses tend to be younger than male actors. Thanks to the magic of ggplot2 and dplyr, separating actors/actresses is relatively simple: add gender (encoded in `category`) as a grouping variable, add it as a color/fill aesthetic in ggplot, and set colors appropriately (I recommend the ColorBrewer qualitative palettes for categorical variables).

```
df_actor_ages_lead <- df_ratings_movies %>%
                  group_by(startYear, category) %>%
                  summarize(low_age = quantile(age_lead, 0.25, na.rm = T),
                            med_age = quantile(age_lead, 0.50, na.rm = T),
                            high_age = quantile(age_lead, 0.75, na.rm = T))

plot <- ggplot(df_actor_ages_lead %>% filter(startYear >= 1920), aes(x = startYear, fill = category, colo
          geom_ribbon(aes(ymin = low_age, ymax = high_age), alpha = 0.2) +
          geom_line(aes(y = med_age)) +
          scale_fill_brewer(palette = "Set1") +
          scale_color_brewer(palette = "Set1")
```



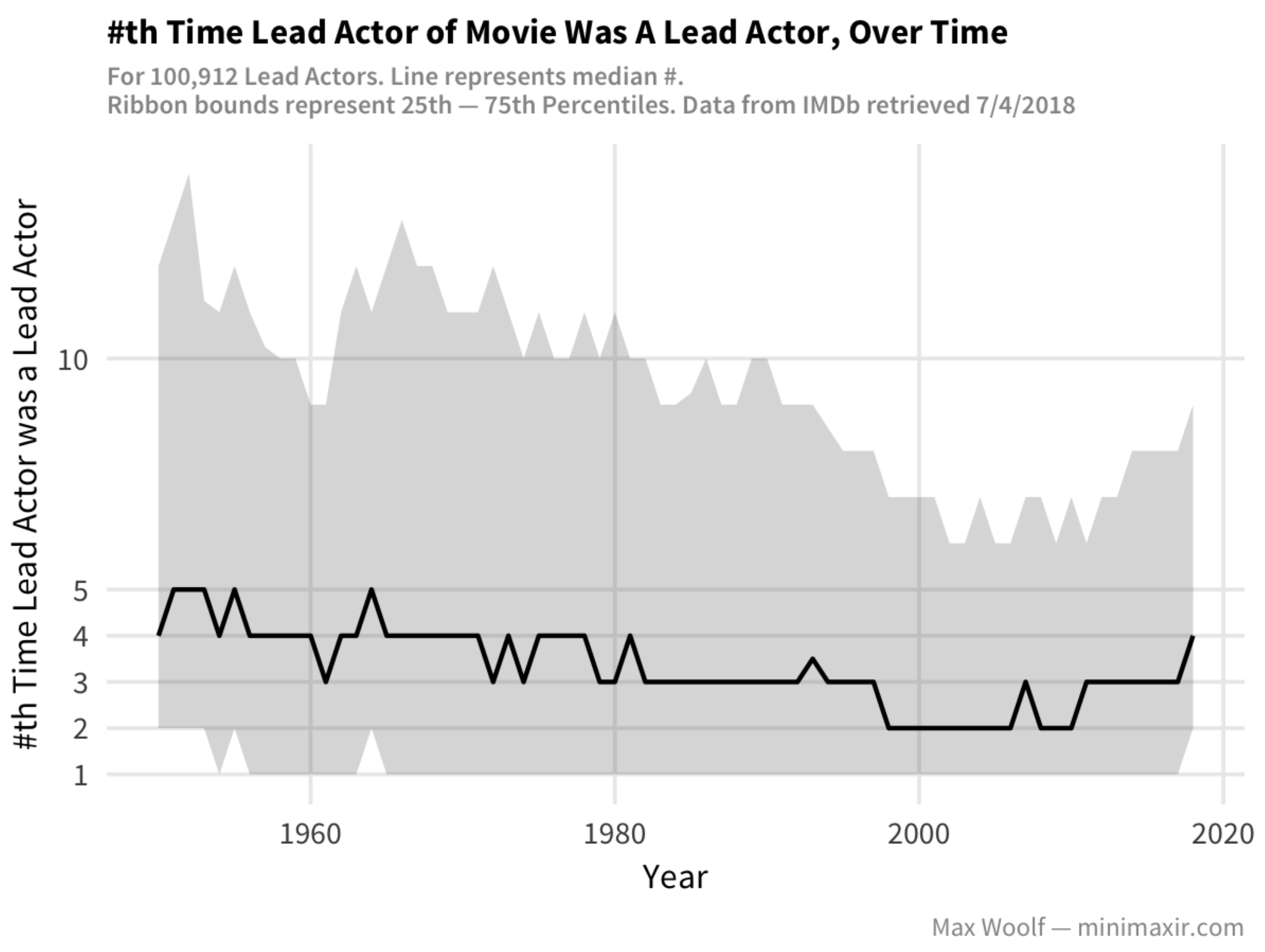
There's about a 10-year gap between the ages of male and female leads, and the gap doesn't change overtime. But both start to rise at the same time.

One possible explanation for this behavior is actor reuse: if Hollywood keeps casting the same actor/actresses, by construction the ages of the leads will start to steadily increase. Let's verify that: with our list of movies and their lead actors, for each lead actor, order all their movies by release year, and add a ranking for the #th time that actor has been a lead actor. This is possible through the use of `row_number` in dplyr, and window functions like `row_number` are data science's most useful secret.

```
df_ratings_movies_nth <- df_ratings_movies %>%
                      group_by(nconst) %>%
                      arrange(startYear) %>%
                      mutate(nth_lead = row_number())
```

primaryTitle <chr>	primaryName <chr>	nth_lead <int>
Avengers: Infinity War	Robert Downey Jr.	20
Black Panther	Chadwick Boseman	6
Deadpool 2	Ryan Reynolds	22
Ready Player One	Tye Sheridan	3
Annihilation	Natalie Portman	14
A Quiet Place	Emily Blunt	7
Solo: A Star Wars Story	Alden Ehrenreich	1
Tomb Raider	Alicia Vikander	12
Game Night	Jason Bateman	15
Red Sparrow	Jennifer Lawrence	12

One more ribbon plot later (w/ same code as above + custom y-axis breaks):



Huh. The median and upper-bound #th time has *dropped* over time? Hollywood has been promoting more newcomers as leads? That's not what I expected!

More work definitely needs to be done in this area. In the meantime, the official IMDb datasets are a lot more robust than I thought they would be! And I only used a fraction of the datasets; the rest tie into TV shows, which are a bit messier. Hopefully you've seen a good taste of the power of R and ggplot2 for playing with big-but-not-big data!

You can view the R and ggplot used to create the data visualizations in this R Notebook, which includes many visualizations not used in this post. You can also view the images/code used for this post in this GitHub repository.

You are free to use the data visualizations from this article however you wish, but it would be greatly appreciated if proper attribution is given to this article and/or myself!

If you liked this post, I have set up a **Patreon** to fund my machine learning/deep learning/software/hardware needs for my future crazy yet cool projects, and any monetary contributions to the Patreon are appreciated and will be put to good creative use.

Video R ggplot2

Max Woolf

Data Scientist at BuzzFeed in San Francisco. Creator of AI text generation tools such as aitextgen and gpt-2-simple. I am the data.

4 Comments minimaxir Disqus' Privacy Policy 1 Login

Favorite 3 Tweet Share Sort by Best

Join the discussion…

LOG IN WITH

OR SIGN UP WITH DISQUS ?

Name

Emmanuel Goldstein • a year ago

Great tutorial. Do you have some code on how to retrieve movies matching criteria?

• Reply • Share ›

Aman Tripathi • 3 years ago

The client wants you to build a real time ML service to identify the most important parameters - which, when provided as an input
- gives an idea whether they should invest in it or not.?

• Reply • Share ›

Elias Oziolor • 4 years ago

Hey man, thank you for making so many cool posts! I learn a ton about processing datasets and visualizing data from you. Big fan of your webpage!

• Reply • Share ›

Shabby Chef • 4 years ago

Some years ago I wrote a system to create a mirror of imdb based on their FTP data which runs within docker, see https://github.com/shabbych... . After pulling the data, it is imported (with filters for TV shows, porn, straight-to-video) into a MYSQL db running in docker. There is also an option for an R/shiny frontend to interact with the data. With the data in MYSQL, it is easy to query with dplyr, as I write here: http://www.gilgamath.com/mo... and http://www.gilgamath.com/bl... and elsewhere.

(I would not be surprised if the docker solution has some bitrot, however: i haven't re-scraped in over a year.)

• Reply • Share ›

Subscribe Add Disqus to your siteAdd DisqusAdd Do Not Sell My Data

Related

- Visualizing Airline Flight Characteristics Between SFO and JFK
- Problems with Predicting Post Performance on Reddit and Other Link Aggregators
- Visualizing One Million NCAA Basketball Shots
- A Visual Overview of Stack Overflow's Question Tags
- How to Make High Quality Data Visualizations for Websites With R and ggplot2