Final Reflection

David Mosack

8 December 2021

Shiny App and GitHub Repository Links

https://damosack.shinyapps.io/BechdelMovies/?_ga=2.142585394.1024157967.1639000711-2034314723.1639000711

https://github.com/damosack/STA518_1finalproject

Course progress

When I began this semester, I had already taken several classes that necessitated using R a lot. Biostatistics, computational biology, and regressional analysis (GVSU) spring to mind. While we did a lot of work in R, and I think that learning R was one of the main objectives, it was so incidental that I got out of these courses knowing how to say a few phrases but not how to make my own, if you'll excuse a bad analogy. Basically, it was a lot of copy and paste, and while I knew how to do some pretty specific high level stuff, no awareness of basics was crippling when trying to write something from scratch. My point, though, is that by the time I started this course I really didn't know how to do anything in R, and I had no idea where to start learning.

Looking back on the past semester, I'm confident I've never learned so much coding language (if you would consider R to have its own language) in such a short amount of time. While I may be better at doing statistical analysis in SAS, I'm certain I know how to work in R with much greater ease, particularly when it comes to data manipulation. There are still things we covered which I'm a bit shaky on, such as for-loops and the creation of elaborate shiny apps (shaky is perhaps an understatement with that one), but I think that overall the amount of effort I put into the course each week was richly rewarded, and when other commitments forced me to focus my attention elsewhere, my understanding of a given week's topic suffered noticeably. Regardless, to the best of my creativity, I always tried to experiment with new code we learned, and incorporate it into other work as much as possible. To be honest, experimentation with web scraping and, later, APIs motivated my topic for the final project perhaps more than anything. Ultimately, the code I wrote for that project was neither lengthy nor extremely complex, but it was the product of more than a few late nights, and much trial and error in R's console. Even if it is far from the most impressive project, and even though I needed help with bugged functions and basic strategy more than once, I'm still proud of the work I've done up to this point.

But speaking in generalities only gets you so far, so I suppose now is a good opportunity to start discussing the learning objectives. I demonstrated fulfillment of the majority of the course learning objectives in my final project, with a sole exception: "Write R progrms for simulations from probability models and randomization-based experiments." So, first I'll discuss how my final project demonstrates I have completed the others, in the order they appear on the course objectives list, and then I'll include a small section of code that shows awareness of the remaining simulation and probability objective.

Import, manage, and clean data

If I were to honestly estimate, I would have to say 85% of the time spent on my project was in some way involved with the importation, management, and tidying of my data. As I mentioned above, one of the foremost reasons for pursuing my project was that I had never performed web scraping before, and wanted to teach myself how to do it. Not only did I scrape thebechdeltest.com for nearly 10,000 movie titles and whether or not each passed the bechdel test, I also searched TMDB data for additional variables associated with each movie using the site's API; in other words importing and combining data from multiple sources to create a novel dataset that suited my purposes. Due to the nature of this process, and the significant iteration necessitated by the size of my original list of movies, the data was prone to contain many errors, and so needed much cleaning.

First I removed all movies where the title I got from thebechdeltest.com didn't match the title I got from querying TMDB's API, then I removed a couple entries which broke later functions (I discovered these to not be movies, which was the problem), next, all movies which were documentaries or recorded concerts (genre = music) since they were irrelevant to my analysis, and finally all movies where budget or revenue were missing or set equal to zero, to get rid of entries with obviously incomplete data.

Management of the data mostly involved the modification of categories for categorical variables, such as recoding bechdel_pass as 0 == fail rather than 0 == pass and later making it simply "pass" or "fail" when I created the shiny app (my original intention was logistic regression, so I wanted it to be an indicator variable early on), and the adaptation of Language to be "English" or "Foreign language" to make it a simple binary variable.

```r
MovieData <- MovieData1 %>%
    filter(primary_genre != "Documentary" & primary_genre != "Music") %>%
    filter(revenue != 0) %>%
    filter(budget != 0) %>%
    mutate(bechdel_pass = ifelse(BT_score.1_is_Pass == 0, "fail", "pass")) %>%
    mutate(spoken_language = language) %>%
    mutate(language = ifelse(language == 'en', 'English Language', 'Foreign Language')) %>%
    mutate(year = year(year)) %>%
    mutate(primary_genre = str_replace_all(primary_genre, "Science Fiction", "SciFi")) %>%
    mutate(logbudget = log(budget), logrevenue = log(revenue)) %>%
    select("title", "TMDB_ID", "bechdel_pass",
           "budget", "revenue", "primary_genre",
           "popularity", "year", "language",
           "runtime","TMDB_rating", "votes",
           "spoken_language", "logbudget", "logrevenue")
```

Create graphical displays and numerical summaries of data for exploratory analysis and presentations

In the shiny app I made after obtaining my data, I included a scatterplot/boxplot (changes depending on whether the x variable is categorical) with marginal histograms and a selectable color input, as well as a separate bargraph when x is categorical and a datatable showing the title, ID, bechdel test result, inputted x, and inputted y variables as the five columns displayed for ten movies at a time. I think the creation of the shiny app itself is a demonstration of this learning objective, since its entire purpose is the display of data, and particularly to facilitate basic exploratory data analysis. Taken together, the shiny app necessitated ggplot2 for basic graphing (I can do a fair amount more than simple scatterplots, boxplots, and histograms, but I doubt it's needed to show here; ggplot2 was the primary thing I learned in other R instruction), ggextra for the incorporation of marginal distribution plots, an understanding of reactivity and conditionals (to show different options according to user selection), and an understanding of shiny's fluidPage formatting. I could easily elaborate on the content of the app, making new tabs for different analyses and the resulting graphs, but given the time constraints of the project, this appeared a good place to stop.

Use source documentation to identify and correct common errors in R programs.

For this project I made much use of source documentation and R's help function to learn the details of unfamiliar packages, functions, and errors. To complete the project I had to learn a fair amount about Rvest (html_element, html_text, html_attr), purrr (map and map_df, mainly: how they work and incidentally also a bit about the nature of lists as opposed to vectors), TMDB (search_movie and movie), and "try".

There's not really a good way to demonstrate this learning objective, that I can think of, other than to say that I hadn't heard of any of these packages prior to starting my project (except purrr), and reading source documentation was the primary way I learned how to use them. I also made extreme use of source documentation when working on my final project for another class this semester, which involved the creation of species distribution models from geolocated observation data and raster formatted predictors for the state of MI. I won't include a link here since my understanding is that I'm not allowed to use work from one class for another, but I recieved zero instruction as to how to do a single thing with raster data for that class, and relied entirely on my own research and troubleshooting to complete it. Packages I learned about for that work included raster, sf, rgdal, and AUK, among a few others.

As for identication and correction of common errors, I did the majority of my experimentation in the console, because I wanted to keep my r script neat, organized, and free of broken code. But if you recall, I struggled a great deal with the first implementation of the search_movie function, and you helped me implement a try-catch to prevent the function from breaking down by forcing it to supply NA values rather than an error. After I used your advice to successfully assemble a list of movies and their TMDB IDs, I needed to perform a similar search function with these IDs for the other variables. As before, my code kept breaking, so I implemented a try-catch similar to the one you showed me. Still, the code broke. I implemented a progress bar to see roughly where the code was breaking, which was at 27%. I don't recall the error now, but it was something to the tune of: "can't slice a list of length zero" when prior to the use of the try-catch, it was "can't bind"adult" (length=3) to "language" (length=4)". At first, I poured over documentation for various functions and couldn't find anything which explained the error. It didn't help that it made no sense to me why the try-catch had resolved one but not the other. Eventually, I found the first movie that broke it by applying my function to narrowing subsets of the data frame until it was breaking for a single movie. I checked to title in my data, then looked up its TMDB page, and saw it was missing all genre data. The same problem arose later when runtime data was missing so I made it that my function returned NAs when runtime or genre as a list of length zero. Turns out the try-catch didn't even help this time, but after fixing it this way my function worked, and I finally had my (very untidy) data. The final function is shown below for reference.

```r
pg <- progress_bar$new(
  format = " downloading [:bar] :percent eta: :eta",
  total=7187, clear = F, width = 60)
 movie_att_function <- function(api_key, id) {
  pg$tick()
  r <- movie(api_key, id)
   if(length(r$genres) == 0 || length(r$runtime) == 0) {
    nope <- tibble("Title" = r$original_title,
                   "ID" = r$id,
                   "Budget" = NA,
                   "Language" = NA,
                   "Popularity" = NA,
                   "Year" = NA,
                   "Revenue" = NA,
                   "Runtime" = NA,
                   "Rating" = NA,
                   "Votes" = NA,
                   "Primary_Genre" = NA)
    return(nope)

    } else {

    return(tibble("Title" = r$original_title,
                    "ID"= r$id,
                    "Budget" = r$budget,
                    "Language" = r$original_language,
                    "Popularity" = r$popularity,
                    "Year" = r$release_date,
                    "Revenue" = r$revenue,
                    "Runtime" = r$runtime,
                    "Rating" = r$vote_average,
                    "Votes" = r$vote_count,
                    "Primary_Genre" = r$genres$name[1]))

    }
  }
```

Write clear, efficient, and well-documented R programs

Again it's difficult to point to specifics here, so I suppose I'll just refer to the organization of the project as a whole, which I separated into three basic tasks:

1. Scrape bechdeltest.com for titles and pass/fails
2. Use TMDB to obtain additional variable data for each movie
3. Write a shiny app that assists with basic EDA

The first two basically comprise the same fundamental task: obtain data and assemble a tibble to use for analysis in the app. Because the goal was the unification of the data from tasks one and two, I performed them both in the same r script (ScrapingMoviesandPasses.R). The script is well annotated with comments that explain what is happening at each step, in such a way that I believe it reasonable for anyone with a fairly basic understanding of R (i.e. me) to figure out what's going on in each section of the document, and why each chunk of code is necessary to achieve the end product. The third step has to be in its own R script since it's the code to run the app, and also I think the intent behind it is distinct enough from steps 1 and 2 that it makes intuitive sense to separate them. So the basic organization is one script for obtaining and processing the data, and one script for analyzing and outputting the data. In both scripts I did my best to remove all code which wasn't strictly necessary for some purpose or explanation. Originally there were also lengthy explanations in the data processing code that summed up every error I got and what I tried already. These were very helpful for my own process, but once I had a working solution, they made for a convoluted and confusing script, so they were removed.

Write R programs for simulations from probability models and randomization-based experiments

Since there was no opportunity to demonstrate this in my project, I'll include a bit of code to show that I can do some basic simulation.

```
#using RNG to pick a seed. Probably completely unnecessary but picking a random no. is hard.
sample.int(10000, size=1)

#RNG gave 2129 so that's the seed to make this reproducible
set.seed(2129)

#picking a random mean from a unif. dist. between -50 and 150
runif(1, min= -50, max= 150)

#mean is random no. from runif above, rounded down. Picked the n and stndev myself to make it fun.
a <- rnorm(500, mean= 63.810, sd= 30)

#creating an extra predictor for variation in y that won't be in the model
b <- rnorm(500, mean= 0, sd= 8)
```

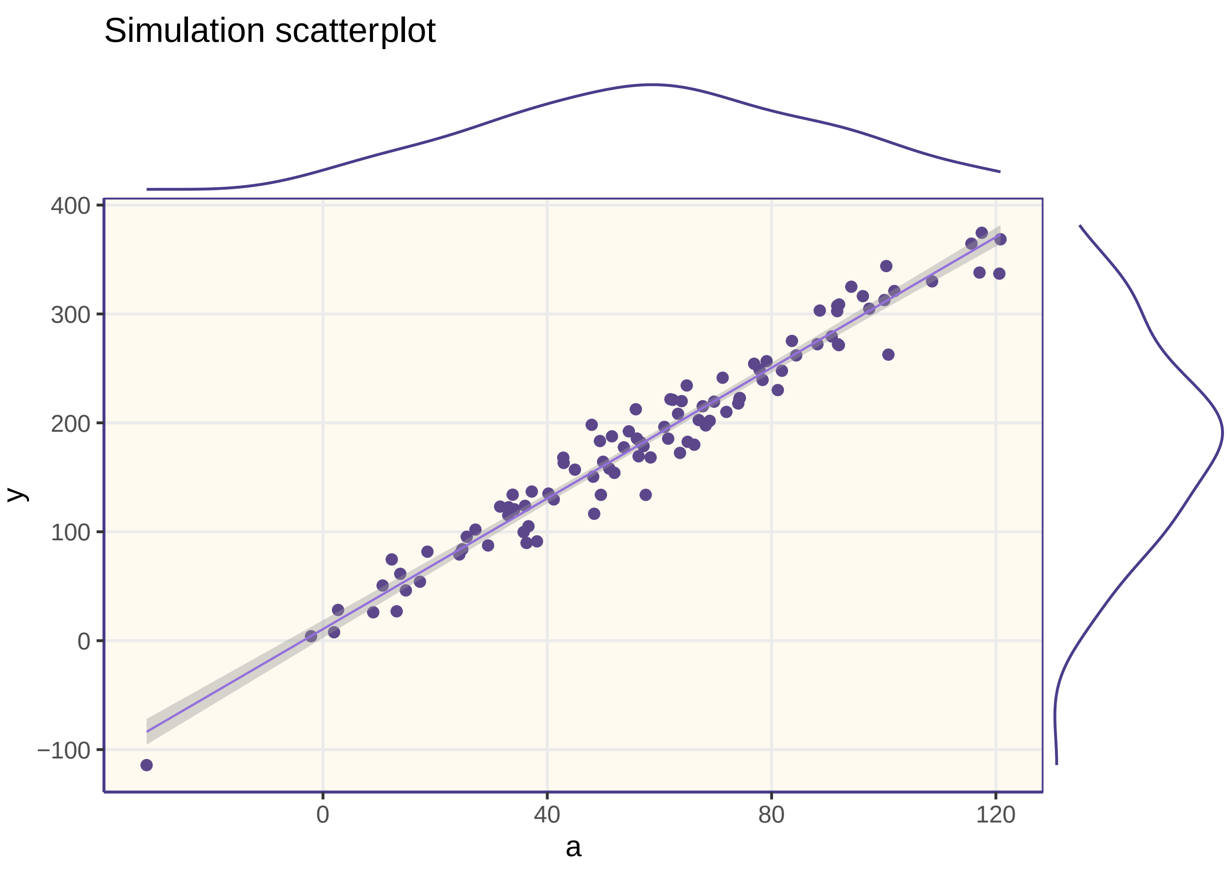
```
#creating error term with mean zero and fairly small stndev
e <- rnorm(500, mean= 0, sd= 20)


#making a response variable according to a simulated model
y <- 12 + 3*a + .5*b + e

#assembling a dataframe df with columns y, a, and e
df <- data.frame(y, a, e)
#randomly sampling 100 obs from df
sample <- sample(1:nrow(df), 100)
#selecting sampled rows and assigning to df_sample
df_sample <- df[sample, ]
```

After I set the seed in the above code, I simulate the random generation of two variables and an error term, which I used to simulate a response variable by creating a function to relate the predictors to the response. For the purpose of further random sampling, I bound these into a data frame, from which I randomly selected 100 observations. This is fairly basic, I'm only simulating normally distributed variables because it makes a nicer linear regression below, and the difference for a Poisson or binary distribution is not big. There's certainly a lot more to simulation, but I think I'm more limited by my imagination and need than ability at this point.

In the code below I create a graph with the simulated data, which I tried to customize to be a little less professional and a little more pleasant than the typical ggplot. I figured I might as well plot the data if I went through the trouble to simulate it, plus this shows a bit more for the learning objective concerning the greation of graphical displays and the learning objective concerning the documentation of R programs.

```
#plotting a and y for the sampled obs. with a little unnecessary customization. I've always been a fan
ggExtra::ggMarginal((
ggplot(df_sample, aes(x= a, y= y)) +
  geom_point(color="mediumpurple4") +
  stat_smooth(method= lm, color= "mediumpurple", size= 0.4) +
  theme_bw() +
  theme(axis.line = element_line(color= "darkslateblue"),
    panel.background = element_rect(fill= "floralwhite"),
    plot.background = element_rect(fill= "white"),
    panel.grid.minor = element_blank(),
    panel.border = element_rect(color="darkslateblue")) +
  labs(title = "Simulation scatterplot")),
type= "density",
color= "darkslateblue")
```

```
## `geom_smooth()` using formula 'y ~ x'
## `geom_smooth()` using formula 'y ~ x'
## `geom_smooth()` using formula 'y ~ x'
```



```r
#checking regression equation
lm(y ~ a)
```

```
##
## Call:
## lm(formula = y ~ a)
##
## Coefficients:
## (Intercept)            a
##      11.521        2.984
```

Final Grade

I would guess I'm really pushing the length limit on this reflection, so I hope it's okay to be pretty brief here. When I started this course, I knew how to do almost nothing in R, and I didn't know where to start if I wanted to learn more. I think the amount of progress I've made is enormous, especially relative to that starting point. Of course, I realize there's a lot of really complex stuff that I have absolutely no idea how to do, but I have a solid enough mastery of basics now, that I'm pretty confident I'd be able to figure out most coding challenges I set my mind to (maybe given unlimited time). When I hit walls with my project, the problem was almost always my patience, and not my ability.

Whether or not I've managed to justify it in this reflection, I've learned a ton about R over the course of this semester. Even my work for other courses (really I'm mainly thinking about CIS 661 here) would not have been possible without the stuff I learned in 518. In my midterm reflection I said something like "everyone could also do more and they could always do less", and I do believe that, but when I look back on the semester I really don't know how much more effort I could have afforded before I burned out. I would like to think I've earned an A.