

**“Over the Top”: Building a Recommender System with Netflix Data using R**

Finally, we’ve reached to the end from this entire series. It’s been a long one, at least in terms of doing all this write up about it, but it’s cool that it’s finally here. So, before I go into what this would entail, here’s a quick recap.

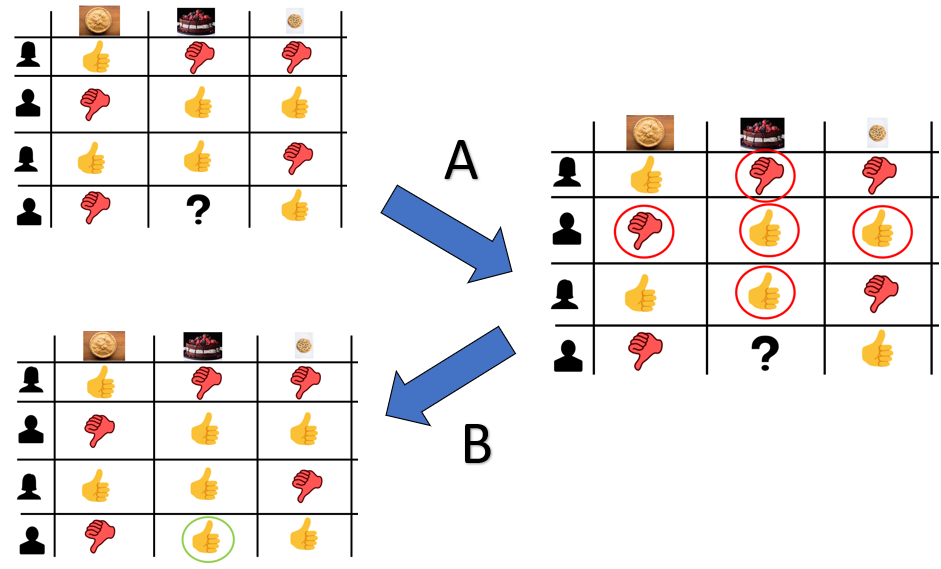
Originally, we’ve began with conducting some data wrangling and basic visualization in our exploratory analysis of the Netflix data as our Tier-1 project. Moving on from that, in the sample Tier-2 project, we’ve improved upon the visualization aspect with the use of the Shiny package by inputting a dynamic element in visualizing data, which in this case pertained to cast members as well as content with credited and uncredited cast. All-in-all, the stuff shown here aren’t really anything that’s too exotic in terms of skillset, just a lot of grunt work in getting things prepped.

Considering that I had explored the data as a whole as well as have a means to dynamically interact with the data, the next logical step would be to apply this data to some sort of real function. So, what better way to do so then by creating a recommender system, a staple machine learning application project.

**A LITTLE BACKGROUND**

You’ll probably are familiar with a recommender system principally given the rise of YouTube, Amazon, Netflix and other web services. However, a quick and dirty definition is that it’s a filter-based application of an algorithm that aims to predict the preference of the user for a given item (i.e., movies/clips to watch, songs to listen, products to buy).

Generally speaking, there are a few major types of recommender systems. One of the most common option are [collaborative-based model](https://en.wikipedia.org/wiki/Collaborative_filtering). It’s one where a model filters items based on the assumption that people have agreed upon in the past will have a similar agreement in the future. This model will only rely on recommendations from a historic rating profile from different users / items to make recommendations.



In this case, the rating history of past member about cakes/pies/cookies as dessert will be used to predict whether or not to recommend cake as a dessert in relation to said person’s opinions about pies and cookies.

A positive about using collaborative filtering is that it can working without having to analyze the influence/impact of the content in accurately predicting and recommending items. It’s just a matter of deciding which sort of machine learning algorithm to use really. However, this model does run into a few problems:

1. **Cold-start**: a problem where a system or a part of it is not working normally due to the lack of connection being made prior to using the model. In other words, if I don’t have anything to go off of, it’s a wild guess whether what I end up recommending would be any better than just randomly guessing.
2. **Sparsity**: Considering the vast amount of content/inventory that can exist, coupled with the fact that only a small subset is actually viewed/selected from the vast majority of active users, there may be few ratings for some of the more popular items.
3. **Scalability:** There may be millions of users and/or content that is available in a database, so the computational power to run this will need to be massive. So, unless you’ve got a dedicated building to hold all of your servers somewhere, this will be a limiting factor.

Considering that the Netflix data set we’re using has neither a rating/watching history component to work from. This method isn’t going to work for us. We do have an alternative solution to this, which is a [content-based recommender system](https://en.wikipedia.org/wiki/Recommender_system#Content-based_filtering). This method is based on the description of the item (i.e., all of the variables used to describe a particular item) to generate recommendation based on the item’s profile and how it compares to some profile.

This is something that is implicitly done whenever you go to a store and ask a salesperson to recommend you a product (say a car) based on some of your important purchasing qualities. However, you really just automate this process by filling out some kind of user profile.



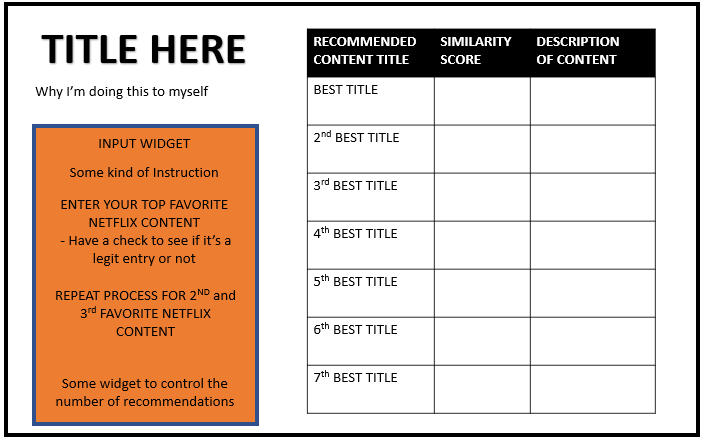
“*Is* t*his asshole really trying to sell me on getting the undercoating?*”

While this method does have its flaws too with respect to scalability as well as its limitation in recommending anything outside of your original preference setting, it does solve the “cold start” or “sparsity” issue as it does not rely on having existing connections available to make predictions. It just based on the weight of each feature, which denotes the importance of it to the user.

So that’s enough about the theory of recommender system. Let’s move onto how to make it happen using our Netflix data set.

**THE PROCESS**

Like all things, we’ll start with a quick visual with what I would like for this who project to look like. In my mind, it should look like this:



*So clearly, we’ll need to use the shiny package at some point*

To do that we first need to load a package called cluster which contains a number of different functions used for cluster analysis. Specifically, we’ll need to use the [daisy function](https://www.rdocumentation.org/packages/cluster/versions/2.1.2/topics/daisy) which is used to compute pairwise dissimilarities between observations in the data set according to some kind of distance-measuring metric like the geometric measure of Euclidean distance or Manhattan distance. However, since we’ll be using mixed data (i.e., different types of data), we’ll be using [Gower’s distance](https://stat.ethz.ch/education/semesters/ss2012/ams/slides/v4.2.pdf) as a metric. The details of Gower’s distance is outside the scope of this article post, but you can read more about it [here](http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.412.4155&rep=rep1&type=pdf), [here](https://rstudio-pubs-static.s3.amazonaws.com/423873_adfdb38bce8d47579f6dc916dd67ae75.html) and [here](https://cran.r-project.org/web/packages/gower/vignettes/intro.pdf). However, the TLDR rationale behind it is that the typical approach to clustering using K-means clustering can’t really be used here since I’m not using numerical data but rather categorical data in nature.

While this is great and all, the only kicker is that the daisy function only works on a matrix and not a data frame. So that’s something that we’ll need to knockout, but first let’s just isolate all of the variables within the existing data set that would be useful/needed for recommending movie/TV series.

Normally, you would do some sort of market research with a subsample of the target population to identify some sort of trend or patterns as it pertains to content and watchability (either in terms of some rating system or viewing frequency). I clearly don’t have that sort of data on my hands, so I would use my own best judgement in predicting what plays a role in recommending content. So, if you are doing this, play around with the sort of metrics that you will use or perform an exploratory analysis relating to Netflix content to pinpoint what sort of features in this dataset matters most in terms of content recommendation. In this case, I’ll be using the following:

* Show ID – need it for the recommender system to work
* Content type – because some people prefer watching TV series over movies or vice versa
* Title – going to need this for the recommender system to work
* Director – sometimes the clout associated with the director may sway someone to watch something (*usually if they are big time like Quentin Tarantino or Martin Scorsese*).
* Cast – self explanatory
* Listed in – contains a list of genres (3 max.) where we might be more inclined to watch one genre over the other
* Country – really going to be used to differentiate English and non-English films
* Rating – referring to content rating like PG-13 vs. TV-MA
* Description – will need this for the end product

From there, we’ll just separate out each element in a similar manner that we did in the Tier-1 project with the [separate()](https://www.rdocumentation.org/packages/tidyr/versions/1.1.3/topics/separate) and [pivot\_longer()](https://tidyr.tidyverse.org/reference/pivot_longer.html) functions

```

library(cluster)

netflix\_for\_recommend = netflix %>% select(show\_id, type, title, director, cast, listed\_in, country, rating, description)

netflix\_for\_recommend = netflix\_for\_recommend %>%

separate(

director, into = c("lead","assitant 1","assistant 2","assistant 3","assistant 4","assistant 5","assistant 6","assistant 7","assistant 8",

"assistant 9","assistant 10","assistant 11","assistant 12"), sep = ", "

) %>%

pivot\_longer(

lead:`assistant 12`, names\_to = "director\_type", values\_to = "director\_name"

) %>% filter(

!is.na(director\_name)

) %>%

mutate(

director\_type = ifelse(director\_type != "lead", "assistant/guest", "lead")

)

netflix\_for\_recommend = netflix\_for\_recommend %>%

separate(

country, c("main country", "secondary country", 'tertiary country', 'fourth country', "fifth country", 'sixth country', "seventh country",

'eighth country', 'nineth country', 'tenth country', 'eleventh country', 'twelfth country'), sep = ","

) %>%

pivot\_longer(

`main country`:`twelfth country`, names\_to = "country\_type", values\_to = "country\_name"

) %>%

filter(

!is.na(country\_name)

) %>%

mutate(

country\_type = ifelse(country\_type == "main country", 'main country', 'other country'),

country\_name = ifelse(country\_name == "", "Unknown Country", country\_name)

)

netflix\_for\_recommend = netflix\_for\_recommend %>%

mutate(

english\_or\_not =

ifelse(c(country\_type == "main country" & country\_name == "United States"), "English Speaking",

ifelse(c(country\_type == "main country" & country\_name == "United Kingdom"), "English Speaking",

ifelse(c(country\_type == "main country" & country\_name == "Canada"), "English Speaking",

ifelse(c(country\_type == "main country" & country\_name == "New Zealand"), "English Speaking",

ifelse(c(country\_type == "main country" & country\_name == "Australia"), "English Speaking",

ifelse(c(country\_type == "main country" & country\_name == "Ireland"), "English Speaking",

ifelse(c(country\_type == "main country" & country\_name == "Jamaica"), "English Speaking",

ifelse(c(country\_type == "main country" & country\_name == "Barbados"), "English Speaking",

ifelse(c(country\_type == "main country" & country\_name == "Belize"), "English Speaking",

ifelse(country\_type == "main country", "Non-English Speaking", NA))))))))))

) %>%

filter(

!is.na(english\_or\_not)

)

netflix\_for\_recommend = netflix\_for\_recommend %>%

separate(

listed\_in, into = c('principal', 'secondary', 'tertiary'), sep = ","

) %>%

pivot\_longer(

principal:tertiary, names\_to = "listing\_type", values\_to = "genre"

) %>%

filter(!is.na(genre)) %>%

mutate(

listing\_type = ifelse(listing\_type == "principal", "principal", 'secondary/tertiary')

)

netflix\_for\_recommend = netflix\_for\_recommend %>%

separate(

cast, into = c("headliner", 'cast member 1', 'cast member 2', 'cast member 3', 'cast member 4', 'cast member 5', 'cast member 6', 'cast member 7',

'cast member 8', 'cast member 9', 'cast member 10', 'cast member 11', 'cast member 12', 'cast member 13', 'cast member 14', 'cast member 15', 'cast member 16',

'cast member 17', 'cast member 18', 'cast member 19', 'cast member 20', 'cast member 21', 'cast member 22', 'cast member 23', 'cast member 24', 'cast member 25',

'cast member 26', 'cast member 27', 'cast member 28', 'cast member 29', 'cast member 30', 'cast member 31', 'cast member 32', 'cast member 33', 'cast member 34',

'cast member 35', 'cast member 36', 'cast member 37', 'cast member 38', 'cast member 39', 'cast member 40', 'cast member 41', 'cast member 42', 'cast member 43',

'cast member 44', 'cast member 45', 'cast member 46', 'cast member 47', 'cast member 48', 'cast member 49'), sep = ", "

) %>%

pivot\_longer(

headliner:`cast member 49`, names\_to = "cast\_type", values\_to = 'cast'

) %>%

filter(

!is.na(cast)

) %>%

mutate(

cast\_type = ifelse(cast\_type == "headliner", "headliner", "supporting cast")

)

```

With this data frame set up, the next move will be to finetune the filtering process by deciding which groups of each feature will be used in determining the appropriate recommendation. For example, I would expect the headlining cast member to play a way bigger role in determining what I would recommend in terms of a movie than some extra way down in the casting list. Similarly, I’m pretty sure that there wasn’t a single point in history that someone decided on something to watch based on the assistant director over the lead director (*unless that assistant director is in your social circle or something*). Thus, we’ll filter out this data frame to include only the following:

* Only include lead cast member (*aka. headliner*)
* Only include the lead director
* Only include the first listed genre since every row will have at least one, which isn’t guaranteed for the subsequently listed genre whereby the absence of this listing may significantly impact the results
* Do not include any content whereby the content isn’t confirmed to be English or Non-English speaking

and then changed their data type from a string to a categorical variable.

```

netflix\_for\_recommend = netflix\_for\_recommend %>%

filter(cast\_type == "headliner") %>%

filter(director\_type == "lead") %>%

filter(listing\_type == "principal") %>%

filter(!is.na(english\_or\_not)) %>%

select(show\_id, type, title, rating, director\_name, country\_name, english\_or\_not, genre, cast, description)

netflix\_for\_recommend = netflix\_for\_recommend %>%

mutate(

type = as.factor(type),

rating = as.factor(rating),

director\_name = as.factor(director\_name),

cast = as.factor(cast),

genre = as.factor(genre),

country\_name = as.factor(country\_name),

english\_or\_not = as.factor(english\_or\_not)

)

```

From here, we’ll work on creating that matrix, which begins with selecting only the variables in our newly created data frame that would be used for generating outcomes. In this case, it is type, content language, cast, director, rating and genre.

```

netflix\_select\_features = netflix\_for\_recommend %>% select(type, english\_or\_not, rating, genre, director\_name, cast)

```

The next step is using this subset as an input for the daisy() function and wrapping it with a function that makes the output a matrix called [as.matrix()](https://www.rdocumentation.org/packages/base/versions/3.6.2/topics/matrix). However, looking into the documentation, you’ll realize that it has an additional argument called “weight” which allows you to specify a weight for each variable (*in accordance to listed order*) instead of treating each variable having equivalent influence with one another. This actually will be first opportunity to optimize your algorithm and an excellent opportunity to utilize that hypothetical example of prospecting item features mentioned earlier. Now in my case, I’m just going to roll that:

1. Content type (i.e., movie vs TV series) and lead director name have comparable impact with each other but have half the impact as content rating and genre.
2. Both content rating and genre have equivalent impact with one another but have half the impact as the name of the lead cast member and whether the content is English-speaking or not.

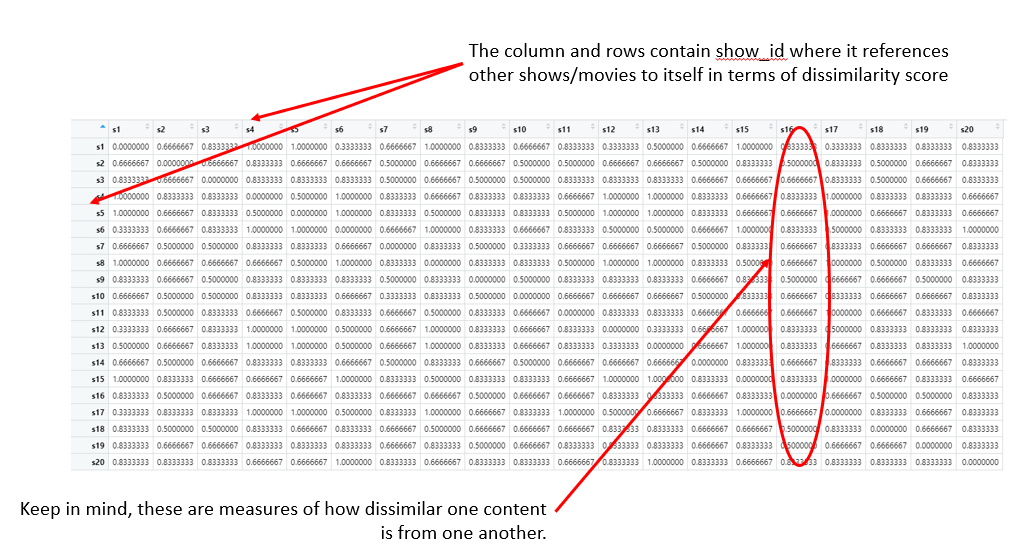
```

dissimilarity = as.matrix(daisy(netflix\_select\_features, metric = "gower"), weight(0.5, 2, 1, 1, 0.5, 2))

row.names(dissimilarity) = netflix\_for\_recommend$show\_id # To add row names for ID purposes

colnames(dissimilarity) = netflix\_for\_recommend$show\_id # To add column names for ID purposes

```



Now before going any further, it’s always best to do a mini-sample test to see if the process works before expanding it to a larger scale. So as long as this test works, I can later expand it to make it look closer to what I had originally planned in my figure above.

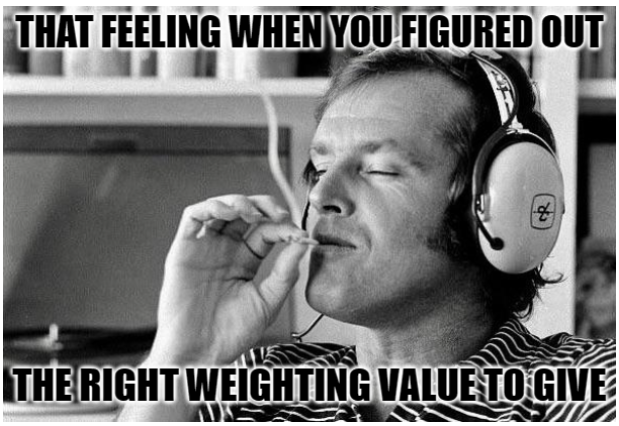
So, what you’ll need to do next is select three of your favorite Netflix content that exists in the data set. If you don’t have any, you can use whatever 3 entries in the data set like I’ll be doing. These three titles are “Sense8”, “Adam Ruins Everything”, and “The Waterboy”. These selected Netflix content will be used to subset the dissimilarity matrix to include only columns corresponding to these titles.

```

test\_selection = netflix\_for\_recommend %>% filter(title == "Sense8" | title == "Adam Ruins Everything" | title == "The Waterboy")

```

At this point we have our second opportunity to optimizing our recommender system as we’ll be introducing a secondary weighting score that corresponds to our ranking of our top 3 Netflix content. This will serve to put a greater emphasis in selecting titles more akin to our top choice as opposed to our second or tertiary choice. The key component to implementing this is to select the right weighted value that’s not too large that it totally skews your recommendations to your top pick but large enough where there these differences can be observed in the recommendations. Now this can vary depending on the number of inputs you plan on using for your selections, but in my case, I worked it out to have a multiple factor of 1.1 for my top choice, 1.05 for my second choice and 1 for the tertiary choice since the dissimilarity scores operate on the scale between 0 and 1.



```

test\_selection = test\_selection %>% mutate(ranking = ifelse(title == "Sense8", 1.1, ifelse(title == "Adam Ruins Everything", 1.05, 1)))

```

We now have everything we need to convert the dissimilarity matrix into a data frame. Now this conversion start by creating a list of show ID corresponding to our top content selection from the index column of the dissimilarity matrix. We can accomplish that by using the [which() function](https://www.r-bloggers.com/2017/03/which-function-in-r/#:~:text=The%20which()%20function%20will,others%20will%20give%20an%20error.). From there, you can use the data.frame() function to convert a filtered out matrix that contains only columns corresponding to the test selection along with a column listing the show ID for every Netflix content.

```

select\_data\_indices = which(colnames(dissimilarity) %in% test\_selection$show\_id)

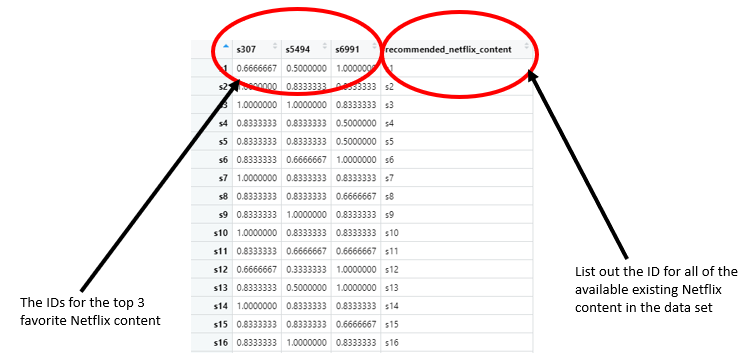
test\_selection\_result\_data = data.frame(

dissimilarity[, select\_data\_indices],

recommended\_netflix\_content = row.names(dissimilarity) # I need a way to list all of the existing content in reference dataframe for selection purpose

)

```



Using this data frame, we’ll have to first reorganize it in such a way where the dissimilarity scores are all listed in a single column that stacked together according to each of the test selected titles. To do this, you’ll just need to use the pivot\_longer() function.

```

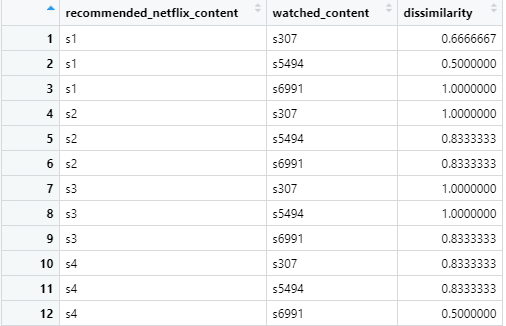
test\_selection\_result\_data %>%

pivot\_longer(cols = c(-"recommended\_netflix\_content"), # This is to reorganize the dataframe with columns for content\_id, dissimilarity metric,

names\_to = "watched\_content", # and what that particular metric score is related to the test selection (there should be like 3 for every content)

values\_to = "dissimilarity")

```



The next step is to join the data frame that exclusively contains features for the test selection that includes that weighting factor pertaining to the ranking of the “favorite” Netflix content.

```

test\_selection\_result\_data %>%

pivot\_longer(cols = c(-"recommended\_netflix\_content"),

names\_to = "watched\_content",

values\_to = "dissimilarity") %>%

left\_join(test\_selection, by = c(“watched\_content” = “show\_id”))

```



It’s important to note that this data frame still contains those “favorites” in the recommendation pool. So, you need to filter them out. Aside from this, you’ll also notice that the scoring used for determining recommendation is in a negative context. If you were to include the influence of the favorite weighting with this score, it’ll have the opposite effect than what is intended. To correct for this, you’ll need to convert this dissimilarity score to a similarity score, which really is a matter of flipping it around.

```

test\_selection\_result\_data %>%

pivot\_longer(cols = c(-"recommended\_netflix\_content"),

names\_to = "watched\_content",

values\_to = "dissimilarity") %>%

left\_join(test\_selection, by = c("watched\_content" = "show\_id")) %>%

filter(recommended\_netflix\_content != watched\_content) %>%

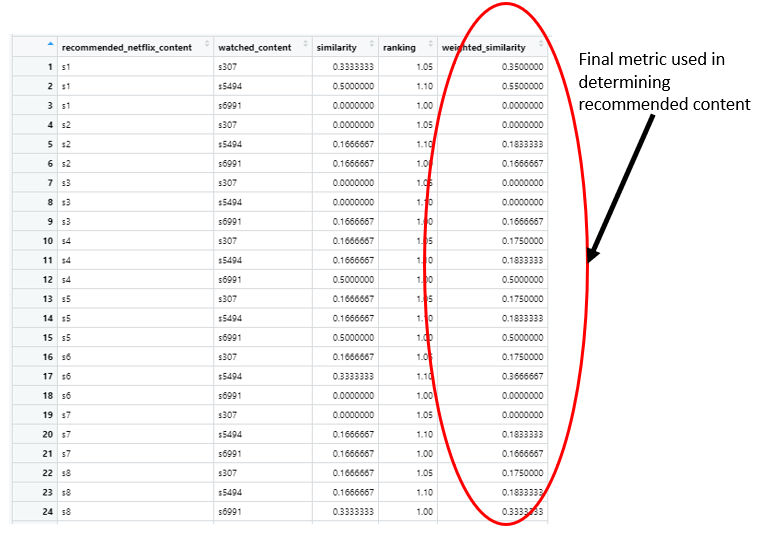
mutate(

similarity = 1-dissimilarity,

weighted\_similarity = similarity\*ranking) %>%

select("recommended\_netflix\_content", “watched\_content”, 'similarity', "ranking", 'weighted\_similarity')

```



The next steps are to group the available recommendation pool according to variable ID, then arrange it according to the weighted similarity score. This would be followed by selecting only the top similarity score for each of the available recommendation content and finally arranged it according to the weighted similarity score again to organize the entire data frame as a whole.

```

something\_here = test\_selection\_result\_data %>%

pivot\_longer(cols = c(-"recommended\_netflix\_content"),

names\_to = "watched\_content",

values\_to = "dissimilarity") %>%

left\_join(test\_selection, by = c("watched\_content" = "show\_id")) %>%

filter(recommended\_netflix\_content != watched\_content) %>%

mutate(

similarity = 1-dissimilarity,

weighted\_similarity = similarity\*ranking

) %>%

select("recommended\_netflix\_content", 'similarity', "ranking", 'weighted\_similarity') %>%

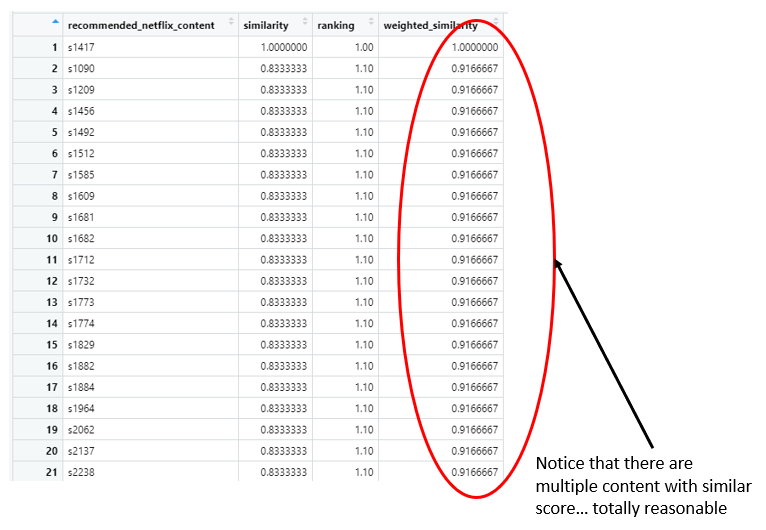
group\_by(recommended\_netflix\_content) %>%

arrange(desc(weighted\_similarity)) %>%

slice(1) %>%

arrange(desc(weighted\_similarity))

```



Finally, you will need to select some number of the top recommended content and then joining this data frame with the first data frame that we did at the beginning of this article. This will enable us to get the corresponding title to the ID, along with the other selection features.

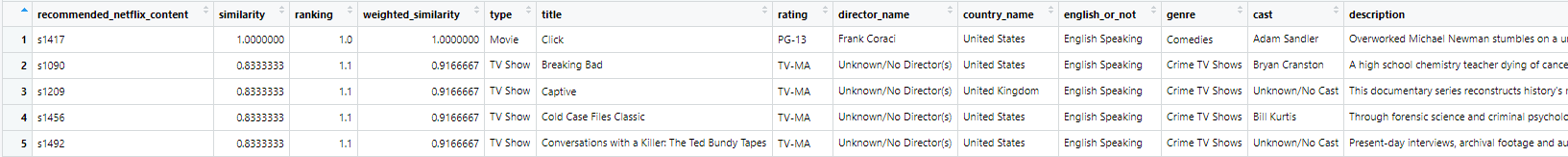
```

top\_recommendations = head(something\_here , 5)

top\_recommentations = top\_recommendations %>%

left\_join(netflix\_for\_recommend, by = c("recommended\_netflix\_content" = "show\_id"))

```



Boom we’ve got a working Netflix content-based recommender system. Pretty sweet right? Obviously, we can automate this by including each of the above steps into a singular function. You can check out that code [here](https://github.com/Vibe1990/Shiny-Project-Ideas/blob/main/Netflix%20Project/Tier%203A:%20Creating%20a%20Recommender%20System#L149). The next obvious step is making this whole process interactive, which we can do by using the Shiny package.

```

library(bslib) # For adding a theme element to the Shiny app

ui = fluidPage(

theme = bs\_theme(bootswatch = "simplex"),

navlistPanel(

"Tier 3 Project: A Content-Based Recommender System in R",

tabPanel(

title = "Find out what to watch next!",

tags$div(

HTML("<body>

<p style = 'font-size: 120%; text-align: justify'>

Ever wanted to have a complete stranger figure out what you should watch on Netflix based on your current favorites on Netflix.

Well here's the app for you. Who needs Rotten Tomatoes, IMDB or other vetted critique source. Type in some of your favorite TV series or movies on Netflix and this thing will try to put out some recommendations for you.

Give it a shot!

</p>

</body>")

),

tags$br(),

fluidRow(

column(width = 4,

wellPanel(

helpText("In this section, we'll need you to enter the title of 3 TV series or Movies on Netflix that you're into in a ranked fashion. We'll use this as a benchmark to figure out what sort of other stuff on Netflix to re-comment to you. As a reminder, you'll need to be mindful of spelling and case."),

tags$br(),

textInput(inputId = "entry\_1",

label = "Put down the title for your top-ranked Netflix content here."),

textOutput(outputId = "entry\_1\_check"),

tags$br(),

textInput(inputId = "entry\_2",

label = "Put down the title for your second-ranked Netflix content here."),

textOutput(outputId = "entry\_2\_check"),

tags$br(),

textInput(inputId = "entry\_3",

label = "Put down the title for your third-ranked Netflix content here."),

textOutput(outputId = "entry\_3\_check"),

tags$br(),

sliderInput(inputId = "n\_recommend", label = "How many recommendations do you want", min = 1, max = 100, value = 5, step = 1),

)),

column(width = 8, tableOutput(outputId = "Recommendations"))

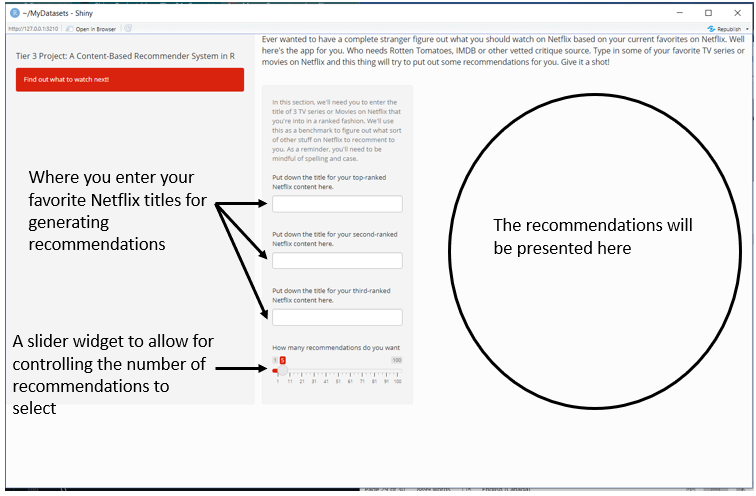
)

)

)

)

```



Considering that the inputs will be relying on text input, it’s probably likely that someone will probably enter the title incorrectly b/c of the lack of capitalization or a misspelling error or include some content that is not present in the data set (possible considering if it’s really new content). This can be checked within the widget below the text entry to denote whether the entry is good to go or not. This is accomplished at the server side.

```

server = function(input, output) {

# Used to check if the entries are valid or NOT

output$entry\_1\_check = renderText({

if (input$entry\_1 %in% netflix\_for\_recommend$title) {

as.character("Yes this title can be used")

} else {

as.character("Something went wrong here. Either there was a mistake in the entry (spelling or case-issue) or this isn't available in what I used in my current Netflix data set.")

}

})

output$entry\_2\_check = renderText({

if (input$entry\_2 %in% netflix\_for\_recommend$title) {

as.character("Yes this title can be used")

} else {

as.character("Something went wrong here. Either there was a mistake in the entry (spelling or case-issue) or this isn't available in what I used in my current Netflix data set.")

}

})

output$entry\_3\_check = renderText({

if (input$entry\_3 %in% netflix\_for\_recommend$title) {

as.character("Yes this title can be used")

} else {

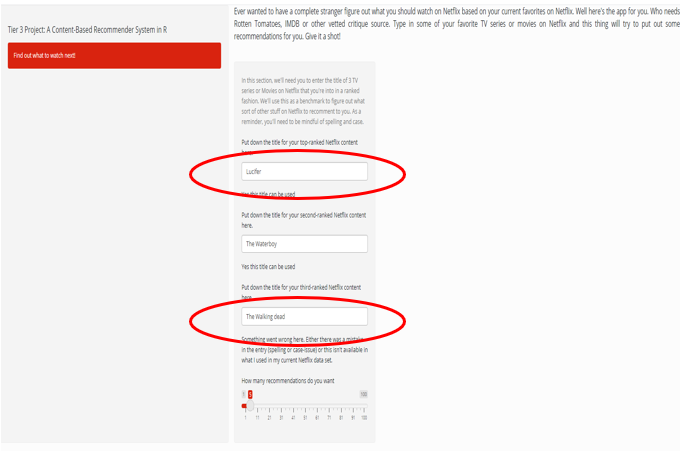
as.character("Something went wrong here. Either there was a mistake in the entry (spelling or case-issue) or this isn't available in what I used in my current Netflix data set.")

}

})

}

```



The last part is to input the table that describes the recommended content along with its weighted similarity score and variable ID.

```

server = function(input, output) {

# Used to check if the entries are valid or NOT

output$entry\_1\_check = renderText({

if (input$entry\_1 %in% netflix\_for\_recommend$title) {

as.character("Yes this title can be used")

} else {

as.character("Something went wrong here. Either there was a mistake in the entry (spelling or case-issue) or this isn't available in what I used in my current Netflix data set.")

}

})

output$entry\_2\_check = renderText({

if (input$entry\_2 %in% netflix\_for\_recommend$title) {

as.character("Yes this title can be used")

} else {

as.character("Something went wrong here. Either there was a mistake in the entry (spelling or case-issue) or this isn't available in what I used in my current Netflix data set.")

}

})

output$entry\_3\_check = renderText({

if (input$entry\_3 %in% netflix\_for\_recommend$title) {

as.character("Yes this title can be used")

} else {

as.character("Something went wrong here. Either there was a mistake in the entry (spelling or case-issue) or this isn't available in what I used in my current Netflix data set.")

}

})

recommending = reactive({

number\_of\_picks = as.numeric(input$n\_recommend)

placeholder\_selection\_data= subset\_dataset()

column\_1 = netflix\_for\_recommend$show\_id[netflix\_for\_recommend$title == as.character(input$entry\_1)]

column\_2 = netflix\_for\_recommend$show\_id[netflix\_for\_recommend$title == as.character(input$entry\_2)]

column\_3 = netflix\_for\_recommend$show\_id[netflix\_for\_recommend$title == as.character(input$entry\_3)]

selected\_indices = which(colnames(dissimilarity) %in% placeholder\_selection\_data$show\_id)

getting\_dataframe = data.frame(dissimilarity[, selected\_indices], recommended\_netflix\_content = row.names(dissimilarity))

recommendation\_data = getting\_dataframe %>%

pivot\_longer(

cols = c(column\_1, column\_2, column\_3),

values\_to = "dissimilarity",

names\_to = "watched\_content"

) %>%

left\_join(placeholder\_selection\_data, by = c("watched\_content" = "show\_id")) %>%

filter(recommended\_netflix\_content != watched\_content) %>%

mutate(

similarity = 1-dissimilarity,

weighted\_similarity = similarity\*ranking

) %>%

group\_by(recommended\_netflix\_content) %>%

arrange(desc(weighted\_similarity)) %>%

slice(1) %>%

arrange(desc(weighted\_similarity))

top\_recommendations = head(recommendation\_data, number\_of\_picks)

top\_recommendations = top\_recommendations %>%

left\_join(netflix\_for\_recommend, by = c("recommended\_netflix\_content" = "show\_id"))

top\_recommendations = top\_recommendations %>%

left\_join(netflix\_for\_recommend, by = c("recommended\_netflix\_content" = "show\_id")) %>%

select(recommended\_netflix\_content, title.y, title.x, weighted\_similarity, dissimilarity, english\_or\_not.x, english\_or\_not.y, cast.x, cast.y, type.x, type.y, genre.x, genre.y, rating.x, rating.y, director\_name.x, director\_name.y, description.y)

selection = top\_recommendations %>% select(title.y, recommended\_netflix\_content, weighted\_similarity, description.y)

colnames(selection) = c("Suggested Titles", "Show Id", "Weighted Similarity Score", "Description")

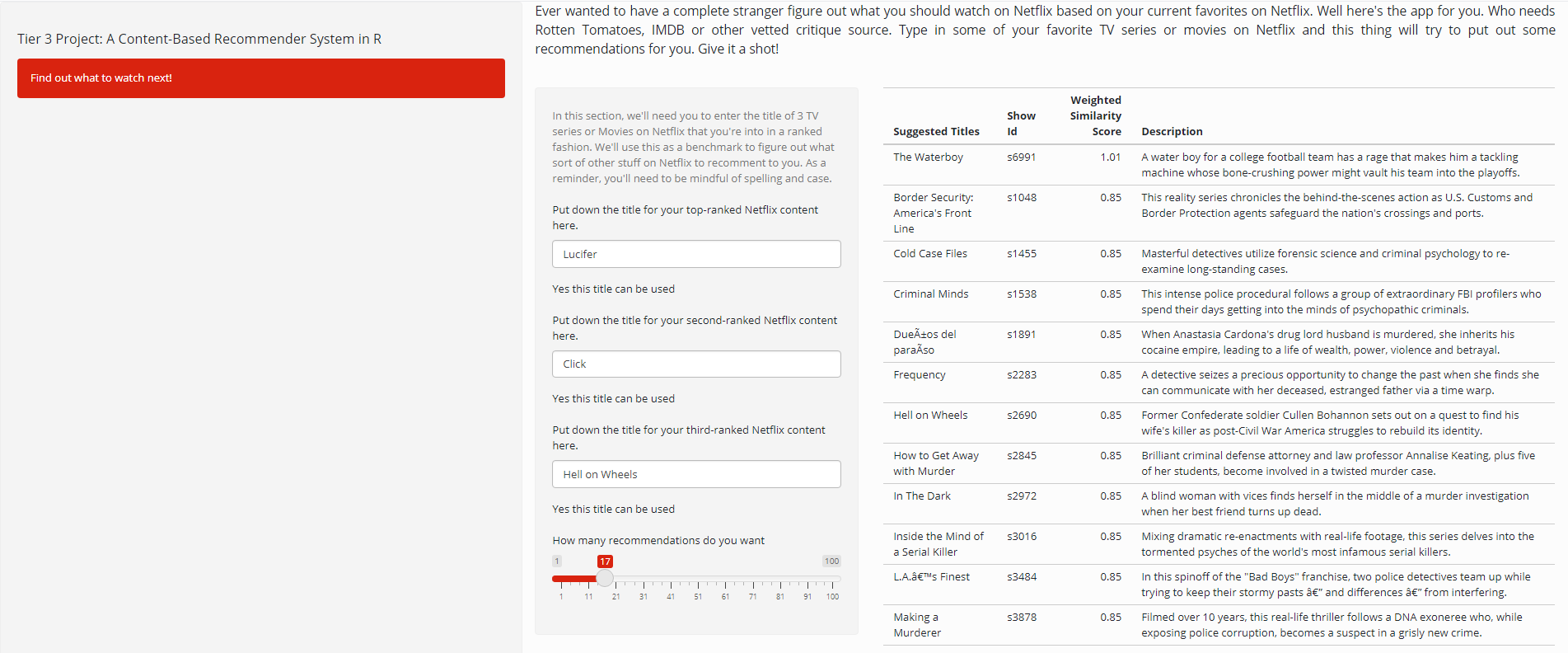
selection

})

output$Recommendations = renderTable({recommending()})

}

```



There we have it. We’ve got a fully functioning Shiny web app that provides a number of TV series/movie recommendations based on some of your favorites. Pretty cool, eh?

Now is this perfect, **ABSOLUTELY NOT**. There are a number of limitations such as:

* Number of features used whereby how the data frame definitely reduced the capacity to which we can include the features in the recommendation system,
* The totally unfounded use of weighting scale as it pertains to the favorite titles that functioned as a multiplier,
* The totally unfounded selection of item features that was thought to play the most significant role in determining items to recommend.
* We could have applied a different form of machine learning algorithm to be used in suggesting recommended title that produced better outcomes based on A/B testing between models.
* We’re missing a significant component in the form of rating/viewership history that could have been included in this dataset. Assuming we had that info, we can create a Hybrid recommender system that would combine the collaborative- and content-based recommendation system, which is something most companies use.

Nevertheless, we brought a proof-of-concept into practice. Hopefully, you can see the jump in both skill set and data science knowledge relating to this project which would make it an appropriate Tier-3 project.

Obviously, if you have the time and patience to do so, you can make an all-encompassing project that included a deep dive analysis of this data set with this app. If you’re interested in how this is coded, you can check out [my code here](https://github.com/Vibe1990/Shiny-Project-Ideas/blob/main/Netflix%20Project/Tier%203B:%20Putting%20it%20All%20Together%20(Server%20Side)) and [here](https://github.com/Vibe1990/Shiny-Project-Ideas/blob/main/Netflix%20Project/Tier%203B:%20Putting%20it%20All%20Together%20(UI%20side)). Ultimately, it will look something [like this](https://drive.google.com/file/d/1o8ui8Gm48cZRW66PXMIxNnMHHFVZNZd7/view?usp=sharing). Not going to lie, this literally took me a month putting in 8 hours/day to get done from start to finish. So, if you are planning on doing something that intensive, just be prepared for it.

There we have it. We finally reached the end of this series putting into action on some of advice I’ve mentioned earlier. Thanks for following along and I hoped that you can take some nuggets of knowledge and/or inspiration to help you on your own unguided projects.

If you’re interested in check out some of my other projects, you can head over to my [GitHub](https://github.com/Vibe1990) to check some of them out. Alternatively, if you’re got some idea on a collaborative project or just want to connect, hit me up on my [LinkedIn](https://www.linkedin.com/in/michael-hoang-3222a220/).

Happy projecting all.