*This post requires some familiarity with the Harry Potter books but I’m committed to making this blog friendly to everyone, even Muggles/Nomajes.*

Have you seen [Mark Sellors’ blog post series](http://blog.sellorm.com/2017/12/18/learn-to-write-command-line-utilities-in-r/) about writing command line utilities in R? It’s a great one but I was a bit puzzled by his using randomness to assign houses in his sorting hat example (he added a [new method based on name digest-ing](http://blog.sellorm.com/2017/12/21/command-line-utilities-in-r-pt-4/) in the meantime).

This is a really cool [#rstats](https://twitter.com/hashtag/rstats?src=hash&ref_src=twsrc%5Etfw) tutorial… but since when does the sorting hat assign a random Hogwarts house?!   
<https://t.co/Ff8CHR6jb9>

— Maëlle Salmon (@ma\_salmon) [19 de desembre de 2017](https://twitter.com/ma_salmon/status/943094156695670784?ref_src=twsrc%5Etfw)

This prompted a reply by David Hood who later came up with R code to assign you to a Hogwarts house based on your Twitter activity!

It should be possible to assign House on the basis of Twitter analysis (among R using tweeters). Quatitatively:   
Original posts – opinionated – Gryffindor  
Replies – social – Slytherin  
posts links out of Twitter – homework – Ravenclaw  
Retweets- keeping it all working – Hufflepuff

— David Hood (@Thoughtfulnz) [19 de desembre de 2017](https://twitter.com/Thoughtfulnz/status/943237103151849473?ref_src=twsrc%5Etfw)

I was thrilled to [see David Hood’s sorting hat Github repo](https://github.com/thoughtfulbloke/sorting_stat) and thought it’d be the perfect occasion to answer that fascinating question: are #python users more likely to get into Slytherin than #rstats users?

*Another note: I do not care about any Python vs. R fights except for Quidditch games, so go away trolls.*

**Getting Twitter users**

In his repo, David Hood used accounts sampled from their posting on the #rstats hashtag, and uses their scores as a scale against which to rate any user. I had to take a different approach because I wanted to compare R against Python posting on a basis that wouldn’t be R. I chose to use JavaScript posters as a reference set, because you can’t randomly search Twitter users and even that wouldn’t necessarily give me a “representative” set. Therefore, I had to assume something, and I chose to assume the Sorting hat would have chosen JavaScript tweeps as a reference set. I wanted more than 200 JavaScript tweeps and more than 100 R and Python tweeps, respectively, and asked for a bit more from Twitter API because I knew there’d be duplicates from people tweeting about more than one of these languages (like [Lynn Cherny](https://twitter.com/arnicas) for instance). I didn’t do any sample size calculation. Here is how I got users:

library("magrittr")

# get users

languages <- c("javascript", "python", "rstats")

# a bit more than what we want because of exclusion criterion: duplicate

no <- c(260, 130, 130)

get\_users <- function(language, no){

users <- rtweet::search\_users(q = paste0("#", language),

n = no)

users$language <- language

return(users)

}

# remove duplicates

users <- purrr::map2\_df(languages, no, get\_users)

users <- dplyr::group\_by(users, screen\_name) %>%

dplyr::mutate(duplicate\_user = n() > 1) %>%

dplyr::ungroup() %>%

dplyr::filter(!duplicate\_user)

Here is the number of accounts I got:

users %>%

dplyr::group\_by(language) %>%

dplyr::summarise(n = n()) %>%

knitr::kable()

| **language** | **n** |
| --- | --- |
| javascript | 258 |
| python | 126 |
| rstats | 128 |

I had actually overestimated the amount of overlap between accounts. Well this does not mean any of these accounts is exclusively related to one of the languages, I have for instance seen rtweet creator [Mike Kearney](https://twitter.com/kearneymw) himself tweet about Python and he’s still in my R sample. It’s because the rank in Twitter Python set might be different than the Twitter R rank, but I’ll go on and assume that if an account was returned for R rather than Python, then it’s not a bad representant of R.

**Computing the sorting metrics**

David Hood calculates a metric for each house, with the logic explained in his [Github repo](https://github.com/thoughtfulbloke/sorting_stat) (I feel a heart pinch every time I write that because it is such a great repo that its creator should *blog*, don’t you think?). I simply used his code after updating rtweet when I realized the output I got from the package had different column names. I probably should have cared a bit more about (Non) Standard Evaluation when wrapping the code in a function but anyway I’m relying on magic!

users <- dplyr::select(users, screen\_name, language)

# function to get scores

get\_scores <- function(screen\_name){

print(screen\_name)

data <- try(rtweet::get\_timeline(user = screen\_name, n = 1000),

silent = TRUE)

if(!is(data, "try-error")){

data %>%

dplyr::mutate(has\_external\_links = ![is.na](http://is.na)(urls\_url),

has\_reply = ![is.na](http://is.na" \t "_blank)(reply\_to\_screen\_name),

has\_reply\_to\_self = screen\_name == reply\_to\_screen\_name & ![is.na](http://is.na" \t "_blank)(reply\_to\_screen\_name),

has\_retweet = ![is.na](http://is.na" \t "_blank)(quoted\_status\_id) | ![is.na](http://is.na)(retweet\_status\_id)) %>%

dplyr::group\_by(screen\_name) %>%

dplyr::summarise(

slyth = sum(has\_reply & !has\_reply\_to\_self) / n(),

huffl = sum(has\_retweet) / n(),

raven = sum(has\_external\_links & !has\_retweet) / n(),

griff = (n() - sum(has\_reply |

has\_retweet |

has\_external\_links |

has\_reply\_to\_self)

) / n()

)

}else{

NULL

}

}

scores <- purrr::map\_df(users$screen\_name, get\_scores)

Let’s have a look at the distribution of scores in the different populations.

library("ggplot2")

users <- dplyr::select(users, screen\_name, language)

scores <- dplyr::left\_join(scores, users, by = "screen\_name")

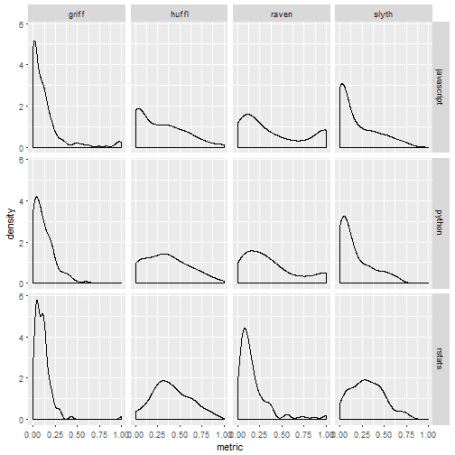
scores %>%

tidyr::gather("house", "metric", slyth:griff) %>%

ggplot() +

geom\_density(aes(metric)) +

facet\_grid(language ~ house)



The graph above gives an idea of what the results will look like…

**Applying the sorting hat**

For each R or Python account, the sorting hat as written by David Hood will compute a score by house obtained by sorting this account and all JavaScript users by the house metric, and then dividing the judged account’s rank by the total number of JavaScript users. The chosen house is the one with the highest score. Note that this script does not ensure that each house has the same size, but in real life the Sorting hat might use other information than Twitter like Instagram or Github activity, and other programming languages might help fill all houses.

I think copy pasting the same code for all houses was the quickest method for David Hood which is fine but I decided I’d re-format the code a little bit to 1) avoid repetitions, 2) getting acquainted with tidy eval (I know, I know, I’m really late) with [Omayma’s blog post](http://www.onceupondata.com/2017/08/12/my-first-steps-into-the-world-of-tidyeval/) as a guide. I also browsed [this issue](https://github.com/tidyverse/dplyr/issues/2944) to find an example of the correct syntax for arrange. I clearly need more practice, which I intend to get this year, using [Edwin Thoen’s reference](https://edwinth.github.io/blog/dplyr-recipes/) among other things.

library("rlang")

compare\_with <- dplyr::filter(scores, language == "javascript")

compare\_with$isSortee <- FALSE

candidates <- dplyr::filter(scores, language != "javascript")

candidates$isSortee <- TRUE

# function for getting the score for one candidate & one house

get\_score <- function(house, fullname, candidate, compare\_with){

result <- dplyr::bind\_rows(compare\_with, candidate) %>%

dplyr::arrange(!!sym(house)) %>%

dplyr::select(!!house,isSortee)

score <- round(which(result$isSortee)/nrow(result),3)

tibble::tibble(house = quo\_name(fullname),

score = score)

}

# function for getting the candidate house

# based on the house scores

sort\_user <- function(candidate, compare\_with){

# seed for the case of equal scores

set.seed(42)

result <- purrr::map2\_df(c("slyth", "huffl", "raven", "griff"),

c("Slytherin", "Hufflepuff",

"Ravenclaw", "Gryffindor"),

get\_score,

candidate = candidate,

compare\_with = compare\_with)

result %>%

dplyr::arrange(desc(score)) %>%

head(n = 1) %>%

dplyr::mutate(screen\_name = candidate$screen\_name,

language = candidate$language) %>%

dplyr::select(house, screen\_name, language)

}

houses <- candidates %>%

split(candidates$screen\_name) %>%

purrr::map\_df(sort\_user, compare\_with = compare\_with)

readr::write\_csv(houses, path = "data/2018-01-01-sortinghat-houses.csv")

**Answering the question**

So now has come the crucial moment of truth… Are Python users more likely to get sorted into Slytherin? I’ll leave statistical tests as an exercise to the reader and just produce a visualization.

library("hrbrthemes")

ggplot(houses) +

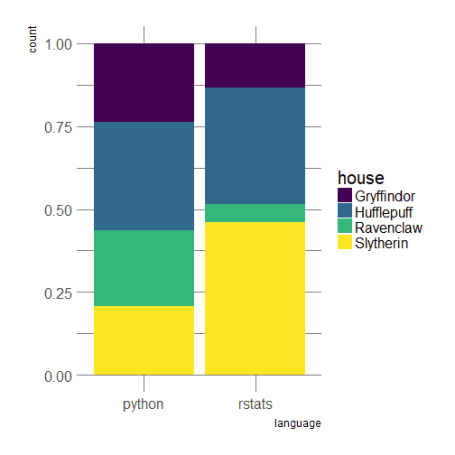
geom\_bar(aes(language, fill = house),

position = "fill")+

theme\_ipsum(base\_size = 20,

axis\_title\_size = 12) +

viridis::scale\_fill\_viridis(discrete = TRUE)



Oh, what a non-intuitive result! But maybe this means that the R community is very social on Twitter; or my sample is too small? In any case, I guess this is good news if the R Slytherin sample contain [parsselmouths](http://harrypotter.wikia.com/wiki/Parseltongue): quite a few useRs should then be able to talk to all the terrifying Australian snakes they’ll meet in Brisbane during useR! 2018.