

# Assignment 7

*Jake Moody*

*February 19, 2017*

For this assignment I scraped data from a popular beer recipe website called beerrecipes.org. For any given beer recipe id, I wanted my script to scrape the beer's name, the beer style, the beer type, the beer rating (out of 5 stars) and the number of votes associated with that rating. I also wanted the script to scrape the user's description of the recipe. All of this information can be scraped from the HTML on the recipe's unique page. This can be accessed directly in the URL using the recipe's ID, which is a unique integer value. Here's a screenshot of a typical page we can scrape. You can see the unique recipe ID (339) in the URL as well all the info we want to scrape for the recipe.

The screenshot shows a web browser window with the address bar displaying `https://beerrecipes.org/Recipe/339/rainy-day-porter.html`. The page title is "Rainy Day Porter". Below the title, there are two buttons: "Back to List" and "Style Details". A rating section shows 5.0 stars based on 5 votes, with a "Vote or Review" button. Below the rating, there are social media sharing icons for Facebook, Google+, Pinterest, Twitter, Tumblr, LinkedIn, and Email. The page also displays the following information: Beer Style: Robust Porter (12B), Recipe Type: extract, and Yield: 5 US gallons. A glass of dark beer is shown to the right. The "Description:" section contains the following text: "I used two types of yeast pitched simultaneously for this brew. One was 5 grams (rehydrated) Doric Ale yeast, and the other was a 'large' sample taken from a previous (cherry ale) brew a few weeks earlier, originally Whittbred Ale yeast. Obviously, this is a very heavy ale, almost like a stout. I'd liken the flavor to Sierra Nevada's porter, but heavier, a little sweeter, and with (delicious) ginger. After about 3 weeks in the bottle, it was, uh, WOW!!! Delicious!! What a combination of flavors! I'd say that this is the correct amount of ginger for such a dark, heavy ale (for my taste). I've had (lighter) ales with too much ginger, but this was just right."

Note the information for the Rainy Day Porter as we will confirm this in our data frame output to ensure the script is working properly:

- Beer name: Rainy Day Porter
- Beer style: Robust Porter
- Beer type: extract
- Beer rating: 5.0
- No. of Ratings: 5

To approach this problem, I used Hadley Wickham's `rvest` package to search and sort the HTML code by node. I then used the `stringr` package to manipulate and clean the HTML. I then saved all the info I was looking for as a data frame. My `getBeer()` function, which returns a data frame of beer ratings from beerrecipes.org, is designed to handle a vector of multiple recipe IDs. The scraping is parameterized so it will take each individual ID in the vector and scrape the HTML for that page before continuing to loop through the rest of the vector. I also made the `getBeer()` function more robust by using the `try()` function to catch errors. In the example, I purposefully added a fake recipe ID '1111111', which does not exist and produces a 404 Error if you try to look it up. The `getBeer()` function is designed to recognize this and skip it if it comes across it. You will see in the final output that it is ignored. Below is the R code showing how the function works; comments next to the code explain each step:

```
getBeer <- function(id) {
  require(rvest) # required for webscraping
  require(stringr) # required for parsing
  beer_data_new <- data.frame() # Clear up this data frame
  beer_data_all <- data.frame() # Clear up this data frame
  # For loop to get the data runs through every recipe id for
  # each id Get URL to Scrape
  for (i in 1:length(id)) {
    url <- paste("https://beerrecipes.org/Recipe/", id[i],
      sep = "") # parameterized URL builder
    test <- try(read_html(url), silent = TRUE) # try the URL
    if ("try-error" %in% class(test)) {
      # if it returns an error (doesn't exist, for example)...
      next #...then skip it
    } else {
      # otherwise Scrape the data from various nodes in the HTML
      scrape_1 <- url %>% read_html() %>% html_nodes("p span") %>%
        html_text() # used to get description
      scrape_2 <- url %>% read_html() %>% html_nodes("p") %>%
        html_text() # used to get style/type
      scrape_3 <- url %>% read_html() %>% html_nodes("a , .mtop-5") %>%
        html_text() # used to get rating
      scrape_4 <- url %>% read_html() %>% html_nodes("div h1") %>%
        html_text() # used to get beer name

      # Then parse the results...

      # Parse Beer Style & Description
      split_4 <- unlist(strsplit(scrape_2[5], "\\:")) # clean the string
      split_5 <- unlist(strsplit(split_4[2], "\\(")) # clean the string
      beer_style <- str_trim(split_5[1]) # get the style of beer
      split_6 <- unlist(strsplit(split_4[3], "\\Y"))
      beer_type <- str_trim(split_6[1]) # gets the type of beer
      description <- scrape_1[2] # parses the description associated with the beer
      beer_name <- scrape_4[3] # parses the beer's name

      # Parse Rating
      split_7 <- unlist(strsplit(scrape_3[40], "\\n")) # clean the string
      split_8 <- unlist(strsplit(split_7[2], " ")) # clean the string
      beer_rating <- str_trim(split_8[1]) # parses the beers rating (out of 5 stars)
      rating_votes <- split_8[5] # parses the beers ratings

      # Create data frame
    }
  }
}
```

```

    beer_data_new <- data.frame(beer_name, beer_style,
                                beer_type, beer_rating, rating_votes, description,
                                stringsAsFactors = FALSE) # create data frame
    beer_data_all <- rbind(beer_data_all, beer_data_new) # bind with previous loops

    # Keep the loop going
    i <- i + 1
  }
}
return(beer_data_all)
}

```

Now, let's test the `getBeer()` function of a sample of 5 beer recipe IDs. Remember, I have included one ID, '111111', which doesn't exist. Therefore, the final output should be a data frame containing 4 observations.

```

# Get Recipe IDs to Scrape
id <- c("542", "278", "339", "111111", "316") # vector of beer recipe ids, purposefully
# put in '111111' which doesn't exist to show robustness -
# loop can handle 404 errors

```

```

## Apply getBeer function to scrape the data
beer_data_results <- getBeer(id)

```

```

# To test the output, we should have 4 observations
nrow(beer_data_results) == 4 # should be TRUE

```

```

## [1] TRUE

```

```

# To test the output, we know beer rating should have a max
# of 5 stars
max(beer_data_results$beer_rating) # should be no higher than 5

```

```

## [1] "5.0"

```

```

# To test the output, we can compare it to the information we
# know about the Rainy Day Porter We can also look at the
# output directly to see
print(beer_data_results)

```

```

##           beer_name           beer_style beer_type
## 1 Green Chili Beerito Spice, Herb, or Vegetable Beer extract
## 2           Steam Beer       California Common Beer all-grain
## 3       Rainy Day Porter           Robust Porter extract
## 4 Colorado Crankcase Stout           Sweet Stout extract
## beer_rating rating_votes
## 1          5.0           1
## 2          5.0           1
## 3          5.0           5
## 4          5.0           2
##

```

```

## 1           One of the advantages of running the Oregon State Fair competition has been
## 2
## 3 I used two types of yeast pitched simultaneously for this brew. One was 5 grams (rehydrated) Doric
## 4

```

*# It appears the data matches what is shown of the webpage of  
# the Rainy Day Porter receipe*