Functions and automation

Dr. Maria Tackett

09.26.19

Click for PDF of slides

Announcements

- Writing Exercise #2 initial draft - **due TODAY at 11:59p**
  - Peer Review available tomorrow and due Sunday 9/29 at 11:59p
- HW 02 - **due TODAY at 11:59p**
- Team Feedback #1 **due TODAY at 11:59p**
  - Please provide honest and constructive feedback. This team feedback will be graded for completion.

Check in: Regrade Requests

- All regrade requests should be submitted through Gradescope. See updated course policy
- Only submit a regrade request if you still have concerns about your grade after you have attended office hours and asked a member of the teaching team to explain the feedback you received. This will ultimately help with your understanding of the course material and help the teaching team get an idea about points to clarify.
- **When you submit a regrade request, please indicate who you've talked with prior to submitting the request.**
- Professor Tackett is the only person who can update grades, so do not ask your TAs to regrade your assignment.

Check in: Lab 04

- Will get Lab 04 assignment from RStudio Cloud project.
- Fill out form with the name of the RStudio Cloud project for grading.

Web scraping

Clean up / enhance

May or may not be a lot of work depending on how messy the data are

- See if you like what you got:

```
glimpse(imdb_top_250)
```

```
## Observations: 250
## Variables: 3
## $ title <chr> "The Shawshank Redemption", "The Godfather", "The Godfathe…
## $ year  <dbl> 1994, 1972, 1974, 2008, 1957, 1993, 2003, 1994, 1966, 1999…
## $ score <dbl> 9.2, 9.1, 9.0, 9.0, 8.9, 8.9, 8.9, 8.9, 8.8, 8.8, 8.8, 8.8…
```

- Add a variable for rank

```
imdb_top_250 <- imdb_top_250 %>%
  mutate(
    rank = 1:nrow(imdb_top_250)
  )
```

title	year	score	rank
The Shawshank Redemption	1994	9.2	1
The Godfather	1972	9.1	2
The Godfather: Part II	1974	9	3
The Dark Knight	2008	9	4
12 Angry Men	1957	8.9	5
Schindler's List	1993	8.9	6
The Lord of the Rings: The Return of the King	2003	8.9	7
Pulp Fiction	1994	8.9	8
The Good, the Bad and the Ugly	1966	8.8	9
Fight Club	1999	8.8	10
...	...	...	...

Analyze

How would you go about answering this question: Which 1995 movies made the list?

```
imdb_top_250 %>%
  filter(year == 1995)
```

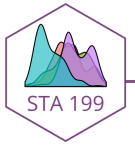
```
## # A tibble: 8 x 4
##   title                year score  rank
##   <chr>               <dbl> <dbl> <int>
## 1 Se7en                1995   8.6    20
## 2 The Usual Suspects   1995   8.5    32
## 3 Braveheart           1995   8.3    75
## 4 Toy Story            1995   8.3    81
## 5 Heat                 1995   8.2   121
## 6 Casino               1995   8.2   138
## 7 Before Sunrise       1995   8.1   194
## 8 La Haine             1995   8     228
```

Analyze

How would you go about answering this question: Which years have the most movies on the list?

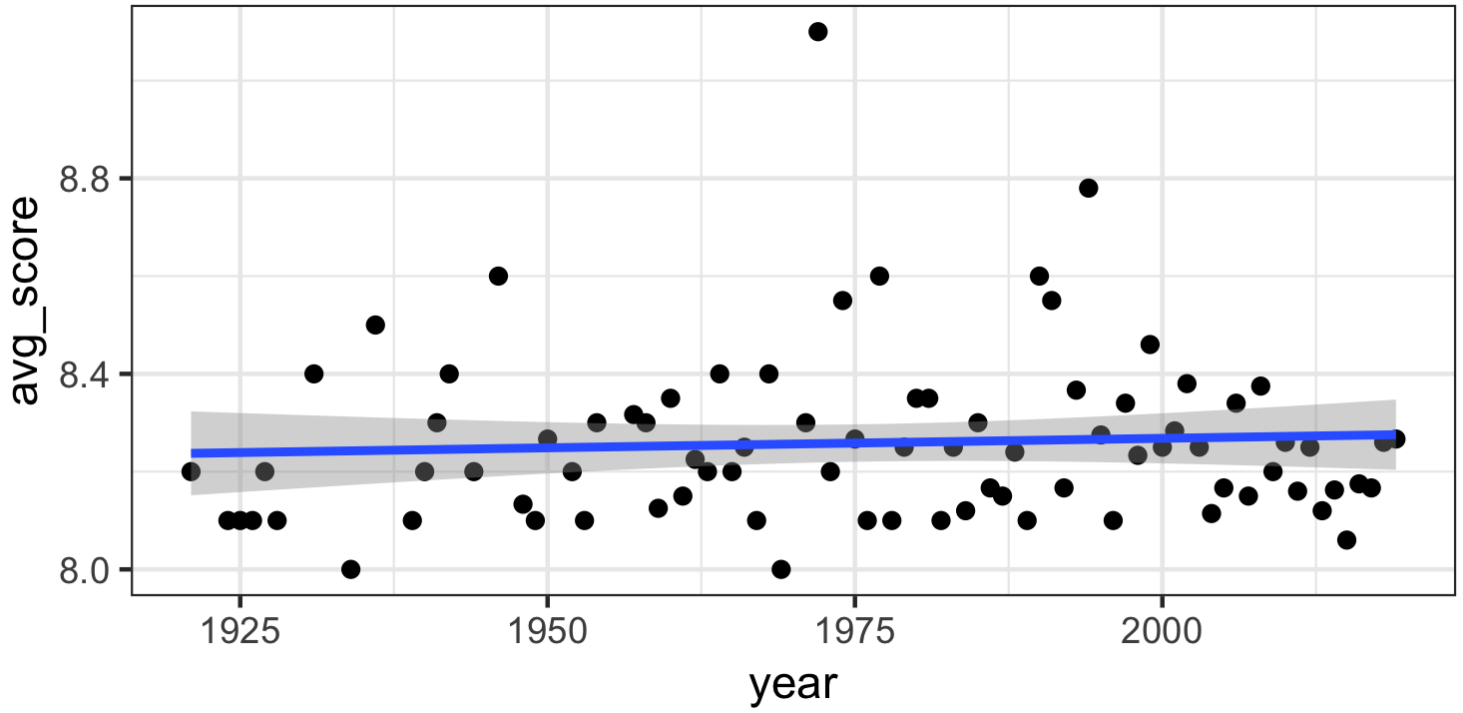
```
imdb_top_250 %>%
  group_by(year) %>%
  summarise(total = n()) %>%
  arrange(desc(total)) %>%
  head(5)
```

```
## # A tibble: 5 x 2
##    year total
##   <dbl> <int>
## 1  1995     8
## 2  2014     8
## 3  2004     7
## 4  1957     6
## 5  1998     6
```



Visualize

How would you go about creating this visualization: Visualize the average yearly score for movies that made it on the top 250 list over time.



Top Rated

- Which year has the highest average score for movies that made the Top 250?

- What is one reason we should write code to answer this question rather than look through the data?
- What is one reason we only want to print the year with the highest average rather than entire table?

Potential challenges

- Unreliable formatting at the source
- Data broken into many pages
- ...

Compare the display of information at raleigh.craigslist.org/search/apa to the list on the IMDB top 250 list.

What challenges can you foresee in scraping a list of the available apartments?

Application Exercise

Popular TV Shows

RStudio Cloud → Web scraping

1. Scrape the list of most popular TV shows on IMDB: http://www.imdb.com/chart/tvmeter

2. Examine each of the first three (or however many you can get through) tv show subpage to also obtain genre and runtime.

3. Time permitting, also try to get the following:

    - How many episodes so far
    - Certificate
    - First five plot keywords
    - Country
    - Language

Add this information to the data frame you created in step 1.

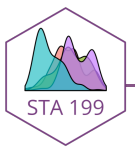
Writing More Efficient Code

Writing Exercise #1: Original code

```
# create data set for each education level
educ_1 <- NHANES %>%
  filter(Education=="8th Grade") %>%
  select(HomeOwn)

educ_2 <- NHANES %>%
  filter(Education=="9 - 11th Grade") %>%
  select(HomeOwn)

educ_3 <- NHANES %>%
  filter(Education=="High School") %>%
  select(HomeOwn)

educ_4 <- NHANES %>%
  filter(Education=="Some College") %>%
  select(HomeOwn)

educ_5 <- NHANES %>%
  filter(Education=="College Grad") %>%
  select(HomeOwn)
```

Writing Exercise #1: Original Code

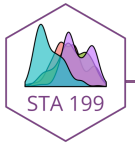
```
# calculate proportion of HomeOwn
educ_1 %>%
  count(HomeOwn) %>%
  mutate(proportion = n/sum(n))

educ_2 %>%
  count(HomeOwn) %>%
  mutate(proportion = n/sum(n))

educ_3 %>%
  count(HomeOwn) %>%
  mutate(proportion = n/sum(n))

educ_4 %>%
  count(HomeOwn) %>%
  mutate(proportion = n/sum(n))

educ_5 %>%
  count(HomeOwn) %>%
  mutate(proportion = n/sum(n))
```



Writing Exercise #1: Efficient code

```
NHANES %>%
  group_by(Education) %>%
  count(HomeOwn) %>%
  mutate(proportion=n/sum(n))
```

- What are two reasons we prefer to write code in this efficient way instead of in the original format?



Functions

Setup

```
library(tidyverse)
library(rvest)

pb <- read_html("https://www.imdb.com/title/tt2442560/")
st <- read_html("https://www.imdb.com/title/tt4574334/")
fr <- read_html("https://www.imdb.com/title/tt0108778/")
```

Why functions?

- Automate common tasks in a power powerful and general way than copy-and-pasting:
  - You can give a function an evocative name that makes your code easier to understand.
  - As requirements change, you only need to update code in one place, instead of many.
  - You eliminate the chance of making incidental mistakes when you copy and paste (i.e. updating a variable name in one place, but not in another).
- Down the line: Improve your reach as a data scientist by writing functions (and packages!) that others use

When should you write a function?

Whenever you've copied and pasted a block of code more than twice.

Do you see any problems in the code below?

```
pb_episode <- st %>%
  html_nodes(".np_right_arrow .bp_sub_heading") %>%
  html_text() %>%
  str_replace(" episodes", "") %>%
  as.numeric()

st_episode <- got %>%
  html_nodes(".np_right_arrow .bp_sub_heading") %>%
  html_text() %>%
  str_replace(" episodes", "") %>%
  as.numeric()

fr_episode <- twd %>%
  html_nodes(".np_right_arrow .bp_sub_heading") %>%
  html_text() %>%
  str_replace(" episodes", "") %>%
  as.numeric()
```

Inputs

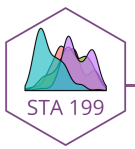
How many inputs does the following code have?

```
st_episode <- st %>%
  html_nodes(".np_right_arrow .bp_sub_heading") %>%
  html_text() %>%
  str_replace(" episodes", "") %>%
  as.numeric()
```

Turn your code into a function

1. Pick a short but informative **name**, preferably a verb.

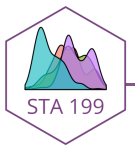
```
scrape_episode <-
```



Turn your code into a function

1. Pick a short but informative **name**, preferably a verb.
2. List inputs, or arguments, to the function inside **`function`**. If we had more the call would look like **`function(x, y, z)`**.

```
scrape_episode <- function(x){




}
```



Turn your code into a function

1. Pick a short but informative **name**, preferably a verb.
2. List inputs, or **arguments**, to the function inside **`function`**. If we had more the call would look like **`function(x, y, z)`**.
3. Place the code you have developed in body of the function, a **`{`** block that immediately follows **`function(...)`**.

```
scrape_episode <- function(x){
  x %>%
    html_nodes(".np_right_arrow .bp_sub_heading") %>%
    html_text() %>%
    str_replace(" episodes", "") %>%
    as.numeric()
}
```

```
scrape_episode(st)
```

```
## [1] 25
```

Check your function

Peaky Blinders

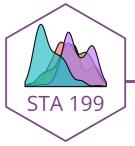
```
scrape_episode(pb)
```

```
## [1] 37
```

Friends

```
scrape_episode(fr)
```

```
## [1] 236
```



Naming functions

> "There are only two hard things in Computer Science: cache invalidation and naming things." - Phil Karlton

- Names should be short but clearly evoke what the function does
- Names should be verbs, not nouns
- Multi-word names should be separated by underscores (**`snake_case`** as opposed to **`camelCase`**)
- A family of functions should be named similarly (**`scrape_title`**, **`scrape_episode`**, **`scrape_genre`**, etc.)
- Avoid overwriting existing (especially widely used) functions

Scraping show info

```
scrape_show_info <- function(x){

  title <- x %>%
    html_node("h1") %>%
    html_text(trim = TRUE)

  runtime <- x %>%
    html_node("time") %>%
    html_text() %>% # could use trim = TRUE instead of str_ functions
    str_replace("\\n", "") %>%
    str_trim()

  genres <- x %>%
    html_nodes(".txt-block~ .canwrap a") %>%
    html_text() %>%
    str_c(collapse = ", ")

  tibble(title = title, runtime = runtime, genres = genres)
}
```

```
scrape_show_info(pb)
```

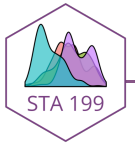
```
## # A tibble: 1 x 3
##   title          runtime genres
##   <chr>          <chr>   <chr>
## 1 Peaky Blinders 1h      " Crime,  Drama"
```

```
scrape_show_info(st)
```

```
## # A tibble: 1 x 3
##   title          runtime genres
##   <chr>          <chr>   <chr>
## 1 Stranger Thin… 51min   " Drama,  Fantasy,  Horror,  Mystery,  Sci-Fi,  T…
```

```
scrape_show_info(fr)
```

```
## # A tibble: 1 x 3
##   title   runtime genres
##   <chr>   <chr>   <chr>
## 1 Friends 22min   " Comedy,  Romance"
```



How would you update the following function to use the URL of the page as an argument?

```
scrape_show_info <- function(x){

  title <- x %>%
    html_node("h1") %>%
    html_text() %>%
    str_trim()

  runtime <- x %>%
    html_node("time") %>%
    html_text() %>%
    str_replace("\\n", "") %>%
    str_trim()

  genres <- x %>%
    html_nodes(".txt-block~ .canwrap a") %>%
    html_text() %>%
    str_trim() %>%
    paste(collapse = ", ")

  tibble(title = title, runtime = runtime, genres = genres)
}
```

```
scrape_show_info <- function(x){

   y <- read_html(x)

  title <- y %>%
    html_node("h1") %>%
    html_text() %>%
    str_trim()

  runtime <- y %>%
    html_node("time") %>%
    html_text() %>%
    str_replace("\\n", "") %>%
    str_trim()

  genres <- y %>%
    html_nodes(".txt-block~ .canwrap a") %>%
    html_text() %>%
    str_trim() %>%
    paste(collapse = ", ")

  tibble(title = title, runtime = runtime, genres = genres)
}
```

Let's check

```
pb_url <- "https://www.imdb.com/title/tt2442560/"
st_url <- "https://www.imdb.com/title/tt4574334/"
fr_url <- "https://www.imdb.com/title/tt0108778/"
```

```
scrape_show_info(pb_url)
```

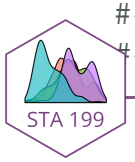
```
## # A tibble: 1 x 3
##   title          runtime genres
##   <chr>          <chr>   <chr>
## 1 Peaky Blinders 1h      Crime, Drama
```

```
scrape_show_info(st_url)
```

```
## # A tibble: 1 x 3
##   title           runtime genres
##   <chr>           <chr>   <chr>
## 1 Stranger Things 51min   Drama, Fantasy, Horror, Mystery, Sci-Fi, Thriller
```

```
scrape_show_info(fr_url)
```

```
## # A tibble: 1 x 3
##   title   runtime genres
##   <chr>   <chr>   <chr>
## 1 Friends 22min   Comedy, Romance
```



Automation

You now have a function that will scrape the relevant info on shows given its URL. Where can we get a list of URLs of top 100 most popular TV shows on IMDB? Write the code for doing this in your teams.

```
urls <- read_html("http://www.imdb.com/chart/tvmeter") %>%
  html_nodes(".titleColumn a") %>%
  html_attr("href") %>%
  paste("http://www.imdb.com", ., sep = "")
```

```
##   [1] "http://www.imdb.com/title/tt7909970/?pf_rd_m=A2FGELUUNOQJNL&pf_rd_p=332cb927
##   [2] "http://www.imdb.com/title/tt1844624/?pf_rd_m=A2FGELUUNOQJNL&pf_rd_p=332cb927
##   [3] "http://www.imdb.com/title/tt9067020/?pf_rd_m=A2FGELUUNOQJNL&pf_rd_p=332cb927
##   [4] "http://www.imdb.com/title/tt2442560/?pf_rd_m=A2FGELUUNOQJNL&pf_rd_p=332cb927
##   [5] "http://www.imdb.com/title/tt1830379/?pf_rd_m=A2FGELUUNOQJNL&pf_rd_p=332cb927
##   [6] "http://www.imdb.com/title/tt0489974/?pf_rd_m=A2FGELUUNOQJNL&pf_rd_p=332cb927
##   [7] "http://www.imdb.com/title/tt1190634/?pf_rd_m=A2FGELUUNOQJNL&pf_rd_p=332cb927
##   [8] "http://www.imdb.com/title/tt5290382/?pf_rd_m=A2FGELUUNOQJNL&pf_rd_p=332cb927
##   [9] "http://www.imdb.com/title/tt6905542/?pf_rd_m=A2FGELUUNOQJNL&pf_rd_p=332cb927
##  [10] "http://www.imdb.com/title/tt8101850/?pf_rd_m=A2FGELUUNOQJNL&pf_rd_p=332cb927
##  [11] "http://www.imdb.com/title/tt10875696/?pf_rd_m=A2FGELUUNOQJNL&pf_rd_p=332cb92
##  [12] "http://www.imdb.com/title/tt4998212/?pf_rd_m=A2FGELUUNOQJNL&pf_rd_p=332cb927
##  [13] "http://www.imdb.com/title/tt0460681/?pf_rd_m=A2FGELUUNOQJNL&pf_rd_p=332cb927
##  [14] "http://www.imdb.com/title/tt8634332/?pf_rd_m=A2FGELUUNOQJNL&pf_rd_p=332cb927
##  [15] "http://www.imdb.com/title/tt0944947/?pf_rd_m=A2FGELUUNOQJNL&pf_rd_p=332cb927
##  [16] "http://www.imdb.com/title/tt1586680/?pf_rd_m=A2FGELUUNOQJNL&pf_rd_p=332cb927
##  [17] "http://www.imdb.com/title/tt5952634/?pf_rd_m=A2FGELUUNOQJNL&pf_rd_p=332cb927
##  [18] "http://www.imdb.com/title/tt4574334/?pf_rd_m=A2FGELUUNOQJNL&pf_rd_p=332cb927
##  [19] "http://www.imdb.com/title/tt1520211/?pf_rd_m=A2FGELUUNOQJNL&pf_rd_p=332cb927
##  [20] "http://www.imdb.com/title/tt1606375/?pf_rd_m=A2FGELUUNOQJNL&pf_rd_p=332cb927
##  [21] "http://www.imdb.com/title/tt7660850/?pf_rd_m=A2FGELUUNOQJNL&pf_rd_p=332cb927
##  [22] "http://www.imdb.com/title/tt0903747/?pf_rd_m=A2FGELUUNOQJNL&pf_rd_p=332cb927
##  [23] "http://www.imdb.com/title/tt0108778/?pf_rd_m=A2FGELUUNOQJNL&pf_rd_p=332cb927
##  [24] "http://www.imdb.com/title/tt1837492/?pf_rd_m=A2FGELUUNOQJNL&pf_rd_p=332cb927
##  [25] "http://www.imdb.com/title/tt6468322/?pf_rd_m=A2FGELUUNOQJNL&pf_rd_p=332cb927
```

Go to each page, scrape show info

Now we need a way to programatically direct R to each page on the **`urls`** list and run the **`scrape_show_info`** function on that page.

```
scrape_show_info(urls[1])
```

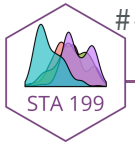
```
## # A tibble: 1 x 3
##   title        runtime genres
##   <chr>        <chr>   <chr>
## 1 Unbelievable 58min   Crime, Drama
```

```
scrape_show_info(urls[2])
```

```
## # A tibble: 1 x 3
##   title                 runtime genres
##   <chr>                 <chr>   <chr>
## 1 American Horror Story 1h      Drama, Horror, Thriller
```

```
scrape_show_info(urls[3])
```

```
## # A tibble: 1 x 3
##   title      runtime genres
##   <chr>      <chr>   <chr>
## 1 The I-Land 40min   Adventure, Drama, Mystery, Sci-Fi
```



Oh no!

1. We're not scraping genre for every TV show!

2. We're repeating our code again!

Update genre code

```
scrape_show_info <- function(x){

  y <- read_html(x)

  title <- y %>%
    html_node("h1") %>%
    html_text() %>%
    str_trim()

  runtime <- y %>%
    html_node("time") %>%
    html_text() %>%
    str_replace("\\n", "") %>%
    str_trim()

  genres <- y %>%
    html_nodes(".see-more.canwrap~ .canwrap a") %>%
    html_text() %>%
    str_trim() %>%
    paste(collapse = ", ")

  tibble(title = title, runtime = runtime, genres = genres)
}
```

Try again: go to each page, scrape show info

```
scrape_show_info(urls[1])
```

```
## # A tibble: 1 x 3
##   title        runtime genres
##   <chr>        <chr>   <chr>
## 1 Unbelievable 58min   Crime, Drama
```

```
scrape_show_info(urls[2])
```

```
## # A tibble: 1 x 3
##   title                 runtime genres
##   <chr>                 <chr>   <chr>
## 1 American Horror Story 1h      Drama, Horror, Thriller
```

```
scrape_show_info(urls[3])
```

```
## # A tibble: 1 x 3
##   title      runtime genres
##   <chr>      <chr>   <chr>
## 1 The I-Land 40min   Adventure, Drama, Mystery, Sci-Fi
```

.. but we're still repeating our code!

Automation

- We need a way to programatically repeat the code
- There are two ways to do this:
  - use a **`for`** loop
  - **`map`**ping with functional programming

`for` loops

- `for` loops are the simplest and most common type of loop in R
- Iterate through the elements of a vector and evaluate the code block for each

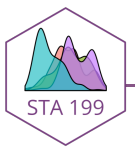
Goal: Scrape info from individual pages of TV shows using iteration with for loops. We'll use only 5 shows for now to keep things simple.

`for` loop

1) Set up a tibble to store the results

```
n <- 5
top_n_shows <- tibble( title = rep(NA, n),
                       runtime = rep(NA, n),
                       genres = rep(NA, n),
                       epsiodes = rep(NA, n),
                       keywords = rep(NA, n)
  )
top_n_shows
```

```
## # A tibble: 5 x 5
##   title runtime genres epsiodes keywords
##   <lgl> <lgl>   <lgl>  <lgl>    <lgl>
## 1 NA    NA      NA     NA       NA
## 2 NA    NA      NA     NA       NA
## 3 NA    NA      NA     NA       NA
## 4 NA    NA      NA     NA       NA
## 5 NA    NA      NA     NA       NA
```



`for` loop

2) Iterate through urls to scrape data and save results

```
for(i in 1:n){
  top_n_shows[i, ] = scrape_show_info(urls[i])
}
top_n_shows
```

```
## # A tibble: 5 x 5
##   title           runtime genres                  epsiodes         keywords
##   <chr>           <chr>   <chr>                   <chr>            <chr>
## 1 Unbelievable    58min   Crime, Drama            Unbelievable     58min
## 2 American Horro… 1h      Drama, Horror, Thriller American Horror… 1h
## 3 The I-Land      40min   Adventure, Drama, Myst… The I-Land       40min
## 4 Peaky Blinders  1h      Crime, Drama            Peaky Blinders   1h
## 5 Top Boy         1h      ""                      Top Boy          1h
```

mapping

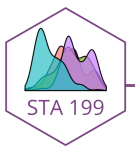
- `map` functions transform the input by applying a function to each element and returning an object the same length as the input
- There are various map functions (e.g. `map_lgl()`, `map_chr()`, `map_dbl()`, `map_df()`)
  - each of which return a different type of object (logical, character, double, and data frame, respectively)
- We will `map` the `scrape_show_info` function to each element of `urls`
  - This will go to each url at a time and get the info

Goal: Scrape info from individual pages of TV shows using functional programming with mapping. We'll use only 5 shows for now to keep things simple.

`map`: Go to each page, scrape show info

```
top_n_shows <- map_df(urls[1:n], scrape_show_info)
top_n_shows
```

```
## # A tibble: 5 x 3
##   title                 runtime genres
##   <chr>                 <chr>   <chr>
## 1 Unbelievable          58min   Crime, Drama
## 2 American Horror Story 1h      Drama, Horror, Thriller
## 3 The I-Land            40min   Adventure, Drama, Mystery, Sci-Fi
## 4 Peaky Blinders        1h      Crime, Drama
## 5 Top Boy               1h      ""
```



Slow down the function

- If you get **`HTTP Error 429 (Too man requests)`** you might want to slow down your hits.
- You can add a **`Sys.sleep()`** call to slow down your function:

```
scrape_show_info <- function(x){
#suspend execution between 0 to 1 seconds

    Sys.sleep(runif(1))

  ...

}
```