Functions and automation

Dr. Maria Tackett

10.01.19

Click for PDF of slides

Announcements

- Writing Exercise #2 final revision - **due TODAY at 11:59p**
- Lab 05 - **due Wednesday at 11:59p**
- Exam 01 - **due Sunday at 11:59p**
  - Friday's lab sessions for exam office hours
- Thursday's Class: Text Analysis with Becky Tang and Graham Tierney
  - EC for Exam 01: Attendance + complete activity
  - Email me documentation **before Thursday's class** if you have an excused absence but would like to participate

Application Exercise: Popular TV Shows

RStudio Cloud \(\rightarrow\) 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:

    - Genre
    - Runtime
    - How many episodes so far
    - First five plot keywords

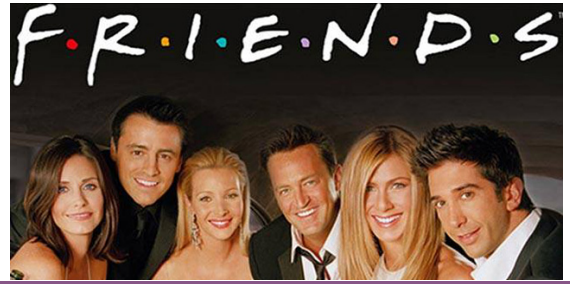
Add this information to the data frame you created in step 1.

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(".bp_sub_heading") %>%
  html_text() %>%
  str_replace(" episodes", "") %>%
  as.numeric()

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

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

Inputs

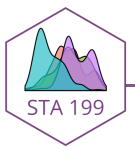
How many inputs does the following code have?

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

```
scrape_episode(st)
```

```
## [1] 26
```

Check your function

Peaky Blinders

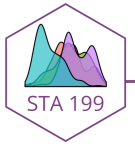
```
scrape_episode(pb)
```

```
## [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
    - e.g. don't name a function **`summarise()`**

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_trim()

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

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

```
scrape_show_info(pb)
```

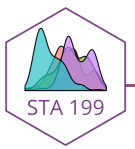
```
## # A tibble: 1 x 3
##   title          runtime genres
##   <chr>          <chr>   <chr>
## 1 Peaky Blinders 1h      Crime,  Drama
```

```
scrape_show_info(st)
```

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

```
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_trim()

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

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

```r
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_trim()

  genres <- y %>%
    html_nodes(".see-more.canwrap~ .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)
```

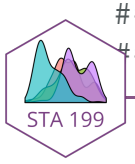
```
## # A tibble: 1 x 3
##   title          runtime genres
##   <chr>          <chr>   <chr>
## 1 Peaky Blinders 1h      Crime, Drama
```

```
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/tt2442560/?pf_rd_m=A2FGELUUNOQJNL&pf_rd_p=332cb927
##   [4] "http://www.imdb.com/title/tt5687612/?pf_rd_m=A2FGELUUNOQJNL&pf_rd_p=332cb927
##   [5] "http://www.imdb.com/title/tt9348692/?pf_rd_m=A2FGELUUNOQJNL&pf_rd_p=332cb927
##   [6] "http://www.imdb.com/title/tt1830379/?pf_rd_m=A2FGELUUNOQJNL&pf_rd_p=332cb927
##   [7] "http://www.imdb.com/title/tt0944947/?pf_rd_m=A2FGELUUNOQJNL&pf_rd_p=332cb927
##   [8] "http://www.imdb.com/title/tt5555260/?pf_rd_m=A2FGELUUNOQJNL&pf_rd_p=332cb927
##   [9] "http://www.imdb.com/title/tt1190634/?pf_rd_m=A2FGELUUNOQJNL&pf_rd_p=332cb927
##  [10] "http://www.imdb.com/title/tt0903747/?pf_rd_m=A2FGELUUNOQJNL&pf_rd_p=332cb927
##  [11] "http://www.imdb.com/title/tt9067020/?pf_rd_m=A2FGELUUNOQJNL&pf_rd_p=332cb927
##  [12] "http://www.imdb.com/title/tt0489974/?pf_rd_m=A2FGELUUNOQJNL&pf_rd_p=332cb927
##  [13] "http://www.imdb.com/title/tt5363918/?pf_rd_m=A2FGELUUNOQJNL&pf_rd_p=332cb927
##  [14] "http://www.imdb.com/title/tt5290382/?pf_rd_m=A2FGELUUNOQJNL&pf_rd_p=332cb927
##  [15] "http://www.imdb.com/title/tt1632701/?pf_rd_m=A2FGELUUNOQJNL&pf_rd_p=332cb927
##  [16] "http://www.imdb.com/title/tt7971476/?pf_rd_m=A2FGELUUNOQJNL&pf_rd_p=332cb927
##  [17] "http://www.imdb.com/title/tt1520211/?pf_rd_m=A2FGELUUNOQJNL&pf_rd_p=332cb927
##  [18] "http://www.imdb.com/title/tt1606375/?pf_rd_m=A2FGELUUNOQJNL&pf_rd_p=332cb927
##  [19] "http://www.imdb.com/title/tt0108778/?pf_rd_m=A2FGELUUNOQJNL&pf_rd_p=332cb927
##  [20] "http://www.imdb.com/title/tt7660850/?pf_rd_m=A2FGELUUNOQJNL&pf_rd_p=332cb927
##  [21] "http://www.imdb.com/title/tt10327354/?pf_rd_m=A2FGELUUNOQJNL&pf_rd_p=332cb92
##  [22] "http://www.imdb.com/title/tt4574334/?pf_rd_m=A2FGELUUNOQJNL&pf_rd_p=332cb927
##  [23] "http://www.imdb.com/title/tt7366338/?pf_rd_m=A2FGELUUNOQJNL&pf_rd_p=332cb927
##  [24] "http://www.imdb.com/title/tt0413573/?pf_rd_m=A2FGELUUNOQJNL&pf_rd_p=332cb927
##  [25] "http://www.imdb.com/title/tt10312066/?pf_rd_m=A2FGELUUNOQJNL&pf_rd_p=332cb92
```

Go to each page, scrape show info

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

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

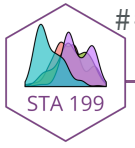
```
## # A tibble: 1 x 3
##   title        runtime genres
##   <chr>        <chr>   <chr>
## 1 Unbelievable 58min   Crime, Drama
```

```
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 Peaky Blinders 1h      Crime, Drama
```



Oh no!

We're repeating our code again!

Automation

- We need a way to programmatically 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

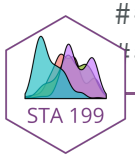
Goal: Scrape info from individual pages of TV shows using iteration with for loops. We'll use only 5 shows for now to keep things simple.

`for` loop

1) Set up a tibble to store the results

```
n <- 10
top_n_shows <- tibble( title = rep(NA, n),
                       runtime = rep(NA, n),
                       genres = rep(NA, n)
  )
top_n_shows
```

```
## # A tibble: 10 x 3
##    title runtime genres
##    <lgl> <lgl>   <lgl>
##  1 NA    NA      NA
##  2 NA    NA      NA
##  3 NA    NA      NA
##  4 NA    NA      NA
##  5 NA    NA      NA
##  6 NA    NA      NA
##  7 NA    NA      NA
##  8 NA    NA      NA
##  9 NA    NA      NA
## 10 NA    NA      NA
```

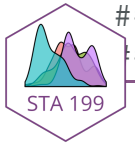


`for` loop

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
```

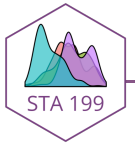
```
## # A tibble: 10 x 3
##    title                 runtime genres
##    <chr>                 <chr>   <chr>
##  1 Unbelievable          58min   Crime, Drama
##  2 American Horror Story 1h      Drama, Horror, Thriller
##  3 Peaky Blinders        1h      Crime, Drama
##  4 Fleabag               27min   Comedy, Drama
##  5 Criminal: UK          <NA>    ""
##  6 Top Boy               1h      ""
##  7 Game of Thrones       57min   Action, Adventure, Drama, Fantasy, Romance
##  8 This Is Us            45min   Comedy, Drama, Romance
##  9 The Boys              1h      Action, Comedy, Crime, Sci-Fi
## 10 Breaking Bad          49min   Crime, Drama, Thriller
```



`map`ping

- **`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: 10 x 3
##    title                 runtime genres
##    <chr>                 <chr>   <chr>
##  1 Unbelievable          58min   Crime, Drama
##  2 American Horror Story 1h      Drama, Horror, Thriller
##  3 Peaky Blinders        1h      Crime, Drama
##  4 Fleabag               27min   Comedy, Drama
##  5 Criminal: UK          <NA>    ""
##  6 Top Boy               1h      ""
##  7 Game of Thrones       57min   Action, Adventure, Drama, Fantasy, Romance
##  8 This Is Us            45min   Comedy, Drama, Romance
##  9 The Boys              1h      Action, Comedy, Crime, Sci-Fi
## 10 Breaking Bad          49min   Crime, Drama, Thriller
```

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))

  ...

}
```

Exam 01

Exam 01 - due Sunday at 11:59p

- Read the exam rules carefully! They can be found in the REAMDE file of the exam-01 repo
- In addition to the correctness of your responses, you will be graded on using tidyverse syntax, style, naming code chunks, commits, and organization
- Make sure to knit, commit, and push your .Rmd and .md documents frequently
    - We will grade the most recent .md file in your repo submitted by the deadline

Exam 01 - due Sunday at 11:59p

- This is an individual assignment - you may not talk about the exam with anyone other than me or the TAs
    - You may only ask me and the TAs clarifying questions.
    - Members of the teaching team will not help you write or debug code
- Friday's lab is exam office hours. You can use this time to ask the TAs clarifying questions
- Use lecture notes and *R for Data Science* as your primary resources.