

Lecture 10: Piping and Webscrapping

STAT 385 - James Balamuta

July 11, 2016

On the Agenda

- ▶ Piping Operator
 - ▶ Motivation
 - ▶ History
 - ▶ Details %>%
- ▶ Web Scrapping
 - ▶ Understanding Document Object Model
 - ▶ Using rvest

Present Day...

Up till now, if we wanted to have step-wise operations on reading data in we would need to do:

```
input_data = read.csv('/path/to/data.csv')  
subset_data = subset(input_data, treatment > 10)  
top_20_data = head(subset_data, 20)
```

Under this approach, we have successfully littered the global environment with tons of variables that have a one time only use.

Long and painful...

To get around that, we can embed the function calls.

```
top_20_data = head(  
    subset(  
        read.csv('/path/to/data.csv'),  
        treatment > 10),  
    20)
```

Though, that doesn't look very nice and the logic is *hard* to follow...

Enter the Pipe Operator

To simplify the process, we opt to use a *pipe* operator defined as `%>%` in the `magrittr` package.

Examples:

- ▶ `x %>% rfunction`
 - ▶ Same as `rfunction(x)`
- ▶ `x %>% rfunction(arg = value)`
 - ▶ Same as `rfunction(x, arg = value)`

Piping is Sequential Logic

Take for example ordering a Starbucks drink via Mobile Order:

find drink, select store, order, go to store, pick up coffee.

```
pickup(goto(store(drink("Java Chip Frap"),  
                    loc="Green St.")))
```

Or

```
"Java Chip Frap" %>% drink %>%  
  store(loc="Green St.") %>%  
  goto %>%  
  pickup
```

Switching to the Pipe

Old:

```
top_20_data = head(  
    subset(  
        read.csv('/path/to/data.csv'),  
        treatment > 10),  
    20)
```

New:

```
read.csv('/path/to/data.csv') %>%  
  subset(treatment > 10) %>%  
  head(20) -> top_20_data
```

Is the Piping Operator a save all?

- ▶ **No.**
- ▶ *However*, the pipe is probably the *most* significant operator to move into *R*'s ecosystem since 2014 since it makes *R* code more user friendly.
- ▶ The operator is **not** for internal package development as it makes for harder debugging.

Bunny Foo Foo and Piping



Hadley Wickham's Bunny Foo Foo Example during his keynote at UseR 2016!

- ▶ Clip starts at 33m 48s and goes till 36m 30s. . .
- ▶ Did you read about this example within the Pipes chapter in R for Data Science?

Example Piping Data

For the next few examples, we'll use the following simulated data:

```
# Set Seed for Reproducibility  
set.seed(1123)  
# Generate Data  
d = data.frame(x=rnorm(10), y = rnorm(10))
```

Problems associated with Piping: Argument Order

`x` may *not* be the first function parameter. e.g.

```
myfunc = function(other_param, x)
```

To get around this issue, use the `.` character to redirect pipe input to a different argument.

For example, when modeling with `lm` notice:

```
# Moved `d` to the data argument  
d %>% lm(y ~ x, data = .)
```

```
##
```

```
## Call:
```

```
## lm(formula = y ~ x, data = .)
```

```
##
```

```
## Coefficients:
```

```
## (Intercept)                x
```

```
##      -0.1744          0.1695
```

Problems associated with Piping: Extracting Information

Sometimes, you may wish to only be able to extract the n th element and pass that further along in the chain.

Use the `.` operator to represent the object on the right hand side (RHS) of the pipe within the left hand side (LHS).

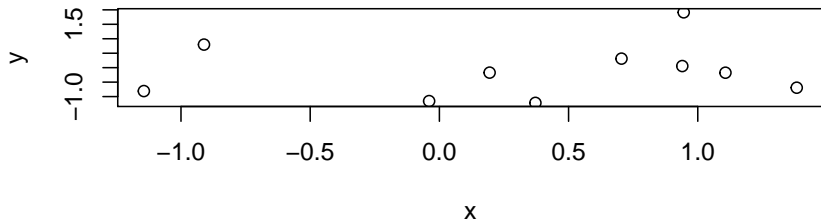
```
d %>% .[["y"]]
```

```
## [1] 0.31234925 -0.17384622 0.05596198 0.79823340
## [5] -0.69309540 -0.17145346 -1.15644219 1.91378654
## [9] -1.21734666 -0.80973361
```

Problems associated with Piping: The Tee Operator

Sometimes a function might not return a value and you want the chain to continue. In such cases, the “tee” operator given as `%T>%` should be used as it returns LHS value instead of the RHS operation result.

```
d %>%  
as.matrix %T>%  
plot %>% # plot will not return anything  
colSums # as.matrix goes into colSums.
```



```
##           x           y  
## 3.552043 -1.141586
```

History of the Piping Operator

The piping operator has existed in many forms over the years. . .

- ▶ Shell/Terminal: Pass command from one to the next with pipeline character `|`.
- ▶ Haskell: Contains many piping operations derived from shell/terminal.
- ▶ F#: Has a forward pipe operator `|>` and served as the motivation for *R*'s.
- ▶ Python: you do **not** have a similar implementation. The closest after 4 years appears to be in the `toolz` module.
- ▶ R: Stefan Milton Bache created `%>%` in the `magrittr` package.
- ▶ Before this, Hadley had introduced this functionality via `%.%` in his rewrite of `plyr` called `dplyr` to which Stefan famously said. . .

Origins of the Pipe Operator

85 comments



Stefan

Dude, you took my operator, sob sob! See
<https://github.com/smbache/magrittr>

But I'm pretty sure you made it better (but my "slogan" is better ;-))

I look forward to trying this package out! Looks great, and speedup is always nice!



hadleywickham

Cool! I love the package and slogan 😊



Stefan Milton Bache commenting on Hadley's Introducing dplyr post on the RStudio Blog.

Remember



- ▶ In English: **This is not a pipe.**
 - ▶ Follows from René Magritte's The Treachery of Images

Exercises

1. Make the following “pipeable”

```
tail(subset(iris, Petal.Width > mean(Petal.Width)))
```

2. Write a pipe that provides the sqrt of 2+2
3. Create another pipe that transforms two strings into one **upper** case string.

```
a = "stat 385 is evolving"  
b = "My pokemon is evolving faster..."
```

Summary

- ▶ Piping is a powerful tool
- ▶ Try to design functions so that they are “interconnected”
- ▶ Avoid using the piping operator within a package’s internals.

Moving along. . .

- ▶ Any questions on the **Piping Operator**?
- ▶ Up next, we're looking at **Web Scrapping**.

Web Scraping

Definition:

Web scraping (**web harvesting** or **web data extraction**) is a computer software technique of extracting information from websites.

From https://en.wikipedia.org/wiki/Web_scraping

Packages in R

There are many packages in *R* that provide web scraping functionality:

- ▶ `rvest` by Hadley Wickham
 - ▶ Downloads HTML and parses it. Support exists for user sessions
- ▶ `RSelenium` by John Harrison
 - ▶ Opens an installed web browser and controls the interface.
 - ▶ Use this when `rvest` fails completely due to websites requiring JavaScript.
- ▶ `xml2` by Hadley Wickham and Gang
 - ▶ Primarily an xml reader that can now also write xml (in turn also HTML).
- ▶ `XML` by Duncan Temple Lang
 - ▶ Original XML reader that has survived the test of time.

Focusing...

For simplicity, we will focus our attention on `rvest` by Hadley Wickham.

- ▶ Please download the Chrome Web browser
- ▶ Also, please visit from SelectorGadget draw their url to your bookmark bar.
 - ▶ Or download the Chrome extension
- ▶ For help, please see the SelectorGadget vignette with `vignette("selectorgadget")`
- ▶ Alternatively, we can use Chrome's Built in Developer Tools via:
- ▶ Windows: `Ctrl + Shift + C`
- ▶ macOS: `Command + Shift + C`

A primer on HTML

Before we begin, HTML stands for **H**yper **T**ext **M**arkup **L**anguage. This is remarkably different from **Markdown**, which wants the minimalist amount of comments.

- ▶ See Mozilla's HTML Reference Guide

A primer on HTML

The basic structure is:

```
<!DOCTYPE html>
<html>
<head>
<title>Title of Page</title>
</head>
<body>

<h1 align = "center">First order heading (large)</h1>
<p>Paragraph for text</p>

<!-- Comment -->

</body>
</html>
```


Core rvest functions

Within here are the key webscrapping functions you will likely use.

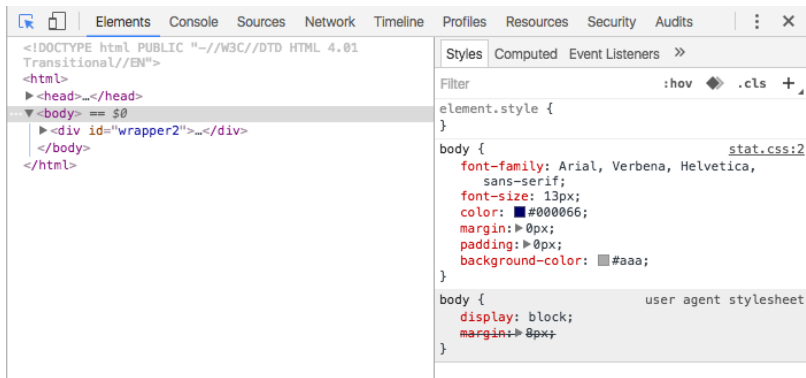
Function	Description
<code>read_html()</code>	Download HTML Output from a website and read into R
<code>html_nodes</code>	Extract HTML Nodes given by <code><tag></tag></code>
<code>html_table</code>	Convert an HTML table (<code><table></table></code>) into a <code>data.frame</code> object
<code>html_text</code>	Extract the text between an HTML tag <code><tag>content</tag></code>

Simple rvest example - Directory of PhD student information

Let's focus a bit on scrapping the stat.illinois.edu website
Specifically, we're going to visit the PhD Student Directory

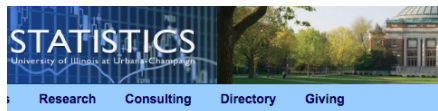
Simple rvest example - Finding the Selectors

- ▶ Go to PhD Student Directory
- ▶ Open Chrome's Dev Tool



Simple rvest example - Finding the Selectors

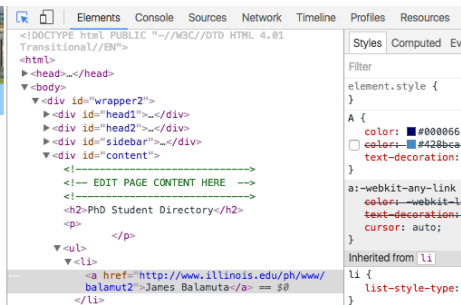
- ▶ Click on “James Balamuta” under the shortcut



PhD Student Directory

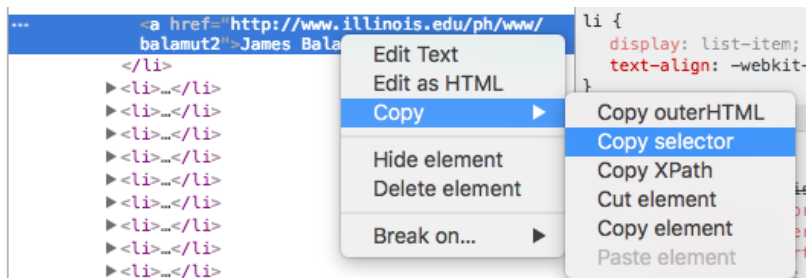
[111.7 × 17](#)

[James Balamuta](#)
Sourabh Banerjee
Xuan Bi
William Biscarri
Matthew Bradley
Yinghan Chen
Yinvin Chen



Simple rvest example - Finding the Selectors

- ▶ Right click on the element to bring up a Copy menu and select Copy Selector



This Gives:

```
#content > ul > li:nth-child(1) > a
```

Simple rvest example - Finding the Selectors

To generalize it, we'll aim to drop the `nth-child(1)` selector

```
#content > ul > li > a
```

Simple rvest example

Scrapping the Directory of PhD student information

```
# Load the Package
```

```
library("rvest")
```

```
# Grab a copy of the PhD Directory
```

```
phds = read_html(  
  "http://www.stat.illinois.edu/people/grad.shtml")
```

```
# Get a list of PhD Names
```

```
phds %>%
```

```
  # Uses selector given before
```

```
  html_nodes("#content > ul > li > a") %>%
```

```
  html_text() -> phd_names
```

```
phd_names %>% .[[1]]
```

```
## [1] "James Balamuta"
```

More complex operations

Sometimes you will need to extract information directly within the tag.

Here is a set of “ideal” functions for that.

Function	Description
<code>html_name</code>	Obtain the name of the tag e.g. <code><h1></h1></code> gives <code>h1</code>
<code>html_attrs</code>	Obtains all the attributes of the tag
<code>html_attr</code>	Obtain only the value associated with a specific attribute.

Obtaining NetIDs

```
# Get a list of PhD Names
phds %>%
  # Uses selector given before
  html_nodes("#content > ul > li > a") %>%
  # Get the linking information
  html_attr("href") %>%
  # Remove everything prior to the directory call
  gsub(".*www/", "", .) -> phd_netids

phd_netids %>% .[[1]] # Pop the first ID

## [1] "balamut2"
```

User Sessions

Often, you might need to create a *persistent* instance where you can make requests to a webserver and receive information. Here, you will find an overview of managing such a session.

Function	Description
<code>html_session</code>	Creates an HTML Session that has persistent cookies.
<code>jump_to</code>	Switches the session from being on one page to the next
<code>follow_link</code>	Enables the session to follow a specific link on a given page
<code>back</code>	Navigates the session back to the prior page
<code>forward</code>	Moves the browser forward to the next page

User Sessions - Example

```
# Similar to the `read_html`  
coatless = html_session("http://github.com/coatless")  
  
# Notice I'm resaving into `coatless`  
coatless %>%  
  follow_link("thecoatlessprofessor") -> coatless
```

```
## Navigating to http://thecoatlessprofessor.com
```

```
# Go back to GitHub (not saved)  
coatless %>% back()
```

```
## <session> https://github.com/coatless  
##   Status: 200  
##   Type:   text/html; charset=utf-8  
##   Size:   80834
```

User Session - continued

```
# Go to one of the repositories
```

```
coatless %>% back() %>% follow_link("@SMAC-Group")
```

```
## Navigating to https://github.com/SMAC-Group
```

```
## <session> https://github.com/SMAC-Group
```

```
##   Status: 200
```

```
##   Type:   text/html; charset=utf-8
```

```
##   Size:   42043
```

Exercises

1. Obtain the first news story title from
`https://news.google.com/`
2. Find the top listed stars of The Thomas Crown Affair
3. Obtain the Statistics faculty netid from
`http://www.stat.illinois.edu/people/faculty.shtml`

Summary

- ▶ `rvest` is small but powerful.
- ▶ Be mindful of the HTML tags.