Piping Data

James Balamuta

Departments of Informatics and Statistics
University of Illinois at Urbana-Champaign

September 20, 2017

Annoucements

- **hw01** due on Friday, September 22nd, 2017 at **1:59 PM**
  - Graded for *accuracy*.
- Form a group by 11:59 PM tonight, Friday, September 22nd, 2017.
  - Between 3 to 4 students.
  - Only one group member needs to email the *TA*:
    - NetIDs and Names of Group Members
    - A team name.
  - Anyone not in a group will be assigned one.
- **hw02** to be released on Friday, September 22nd before *midnight*
  - Due September 29th, 2017 at **1:59 PM**
- **hw00** grades to be released this weekend.
  - This was graded on a *completion* basis.

On the Agenda

Present day. . .

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

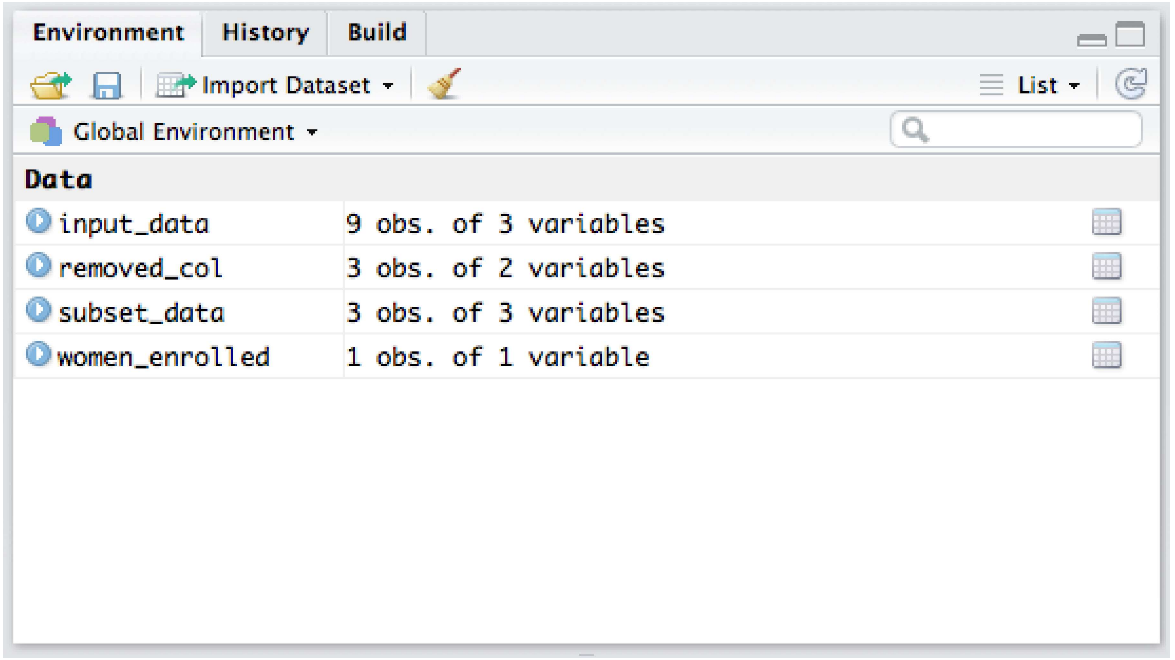
```r
library("dplyr")
input_data  = read.csv('enrolled_fa17_tidy.csv')
subset_data = filter(input_data, Gender == "Women")
removed_col = select(subset_data, -Gender)
women_enrolled = summarise(removed_col,
                           Total_Enrolled = sum(Enrolled))
```

LITTER!

Source: The Odyssey Online

No, not that kind of litter. . .

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



```
Data
input_data        9 obs. of 3 variables
removed_col       3 obs. of 2 variables
subset_data       3 obs. of 3 variables
women_enrolled    1 obs. of 1 variable
```

What variable do we care about?

Long and painful. . .

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

```
women_enrolled = summarise(
                       select(
                         filter(
                             read.csv(
                               'enrolled_fa17_tidy.csv'
                             ),
                          Gender == "Women"),
                       -Gender),
                  Total_Enrolled = sum(Enrolled))
```

Why is this approach not ideal?

Enter the Pipe Operator

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

```r
# install.packages("magrittr")
library("magrittr")

# One parameter function
y = square(x)       # Before
y = x %>% square() # After

# Two parameter functions
y = add(a, b)       # Before
y = a %>% add(b)    # After
```

Piping is Sequential Logic

Take for example ordering a Starbucks drink via Mobile Order:

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

Embedded Functions:

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

Piped Functions:

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

Switching to the Pipe

Old:

```
women_enrolled = summarise(
                       select(
                         filter(
                             read.csv(
                               'enrolled_fa17_tidy.csv'
                             ),
                          Gender == "Women"),
                       -Gender),
                  Total_Enrolled = sum(Enrolled))
```

Switching to the Pipe

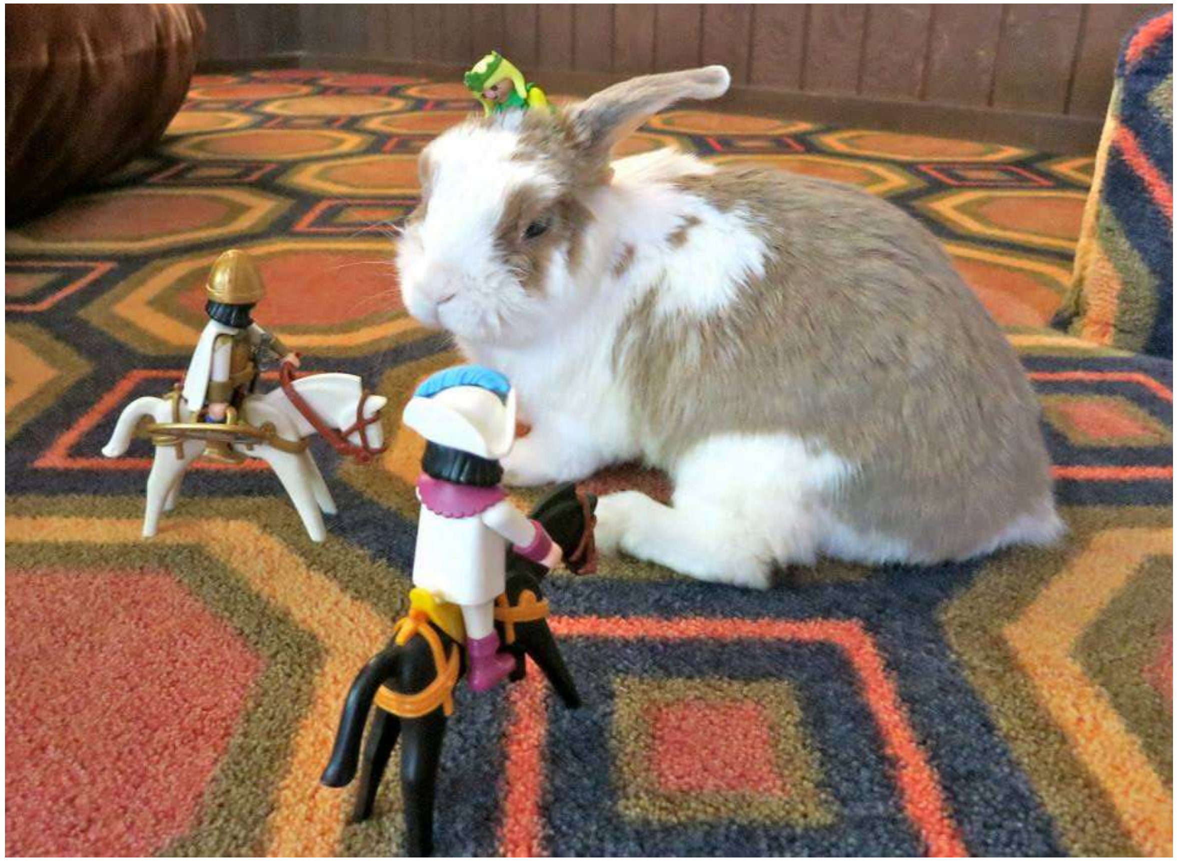
New:

```
women_enrolled = read.csv('enrolled_fa17_tidy.csv') %>%
  filter(Gender == "Women") %>%
  select(-Gender) %>%
  summarise(Total_Enrolled = sum(Enrolled))
```

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 at UseR 2016 Keynote

Clip starts at 33m 48s and goes till 36m 30s...

Problems associated with Piping: Argument Order

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

```
myfunc(other_param, x)
```

To get around this issue, use the . character to redirect pipe input.

```
x %>% myfunc(other_param, . )
```

Returning to Sequential Statements

```r
# Install demagrittr if needed
# devtools::install_github("TobCap/demagrittr")

# Load the demagrittr package
library("demagrittr")
# Pipe for function composition (f o g)(x) = f(g(x))
demagrittr(x %>% g() %>% f(), mode = "eager")
```

```
## {
##     `#0` <- x
##     `#1` <- g(`#0`)
##     f(`#1`)
## }
```

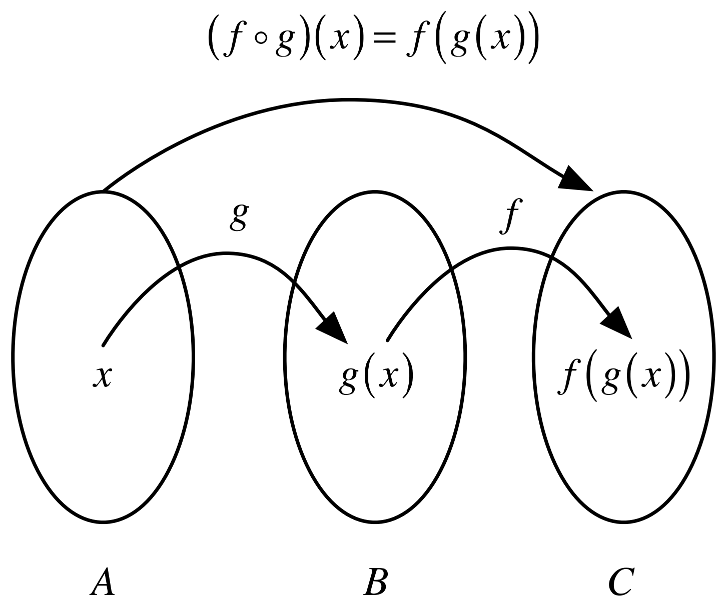
GitHub Package: `demagrittr`

Mathematical Origin of Piping: Composition of Functions

Consider two functions $f : B \to C$ and $g : A \to B$.

These functions can be chained together by taking the output of one function and inserting it into the next, *e.g.* $f(g(x))$. Thus, the input g is x and the result $g(x)$ serves as the input for f.

The notation for this is $f \circ g$ (read as "f follows g"). The previous pipe can be represented visually as:

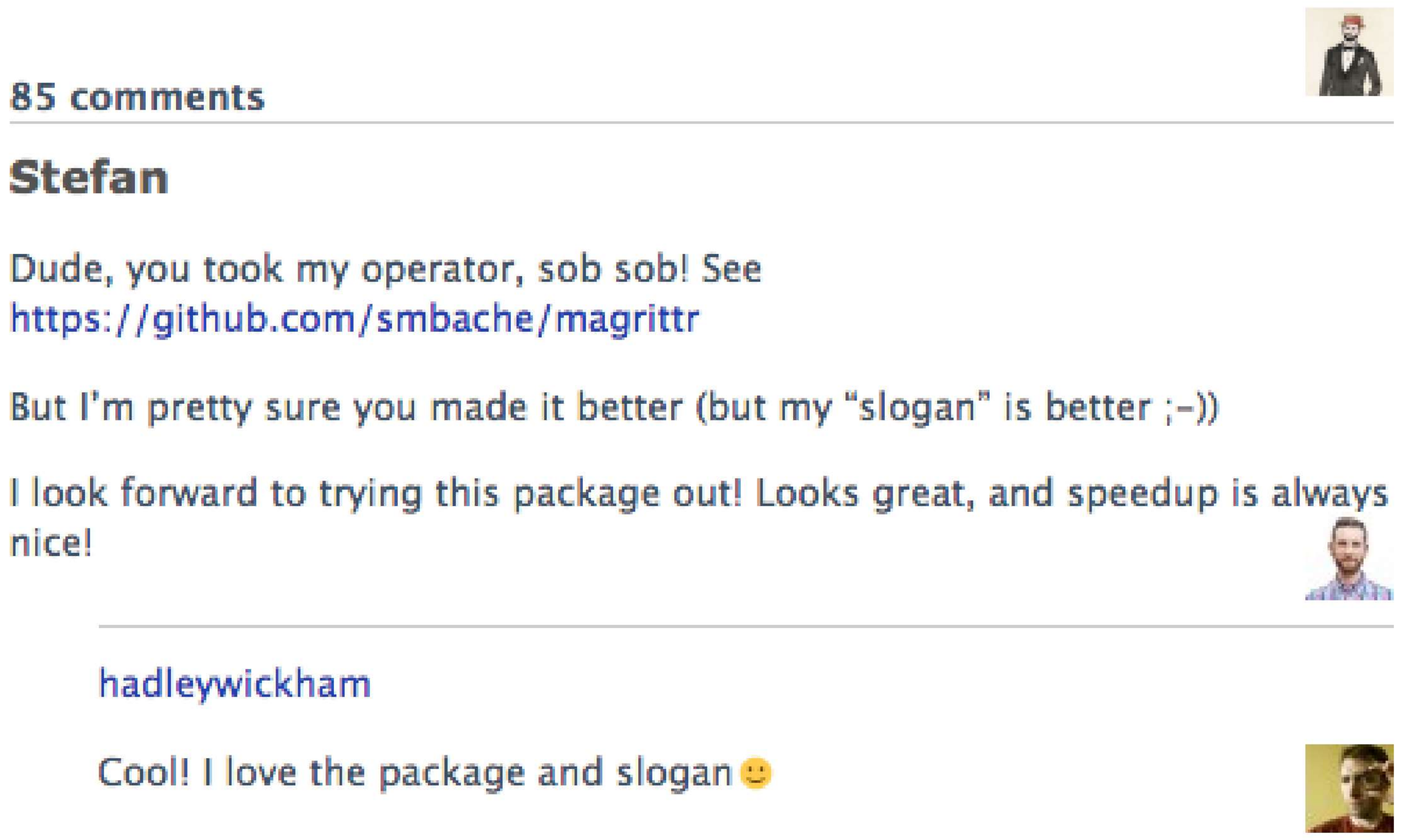


History of the Piping Operator

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

- Shell/Terminal: Pass command from one to the next with pipeline character `|`.
- `F#`: Forward pipe operator `|>` and served as the motivation for *R*'s.
- `Haskell`: Contains many piping operations derived from shell/terminal.
- `Python`: **Lacks** a similar implementation to *R*'s. The closest after 4 years appears to be in the toolz module.
- `R`: Stefan Milton Bache created `%>%` in the magrittr package.
    - Unbeknowist to Hadley, he introduced this functionality via `%.%` in his rewrite of `plyr` called `dplyr` to which Stefan famously replied...

Origins of the Pipe Operator in R



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

Ceci n'est pas une pipe.

In English: **This is not a pipe** | **René Magritte's The Treachery of Images**

Exercises

1. Make the following “pipeable”

```
library("dplyr")
tail(filter(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..."
```

On the Agenda

Acknowledgements and References

- Pipes chapter in R for Data Science
- `magrittr` Vignette: Introducing magrittr
- `demagrittr` GitHub
- The styling given in RStudio cheatsheets were influential in the design of this slide deck.