Intuitive Programming Via Piping and the Magrittr Package

Owen McGrattan

10/28/2017

Where the magrittr package gets its name from

```
# load required packages
library(readr)
library(magrittr)
library(dplyr)
```

```
##
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
library(ggplot2)
```

```
## Warning: package 'ggplot2' was built under R version 3.3.2
```

Working through code in R and setting up data tables can be quite unintuitve with how we think. Often times before we start to create a variable or transform a data frame, we think of what we want to come out in the end. But the whole process is hindered by having to continually assign new names and carry them through each line. It ultimately is sort of a mess that can be difficult to follow and understand.

Insert Hadley Wickham. Wickham is responsible for creating magrittr and is arguably the biggest name in R community, only behind the creators of R Robert Ihaka and Robert. Wickham is man behind dplyr, ggplot2, tidyr, and so many more R packages that are essential in today's R usage. There's an endless amount of his work to hit on, but here I'll talk about magrittr.

For example I'll look at some data from the 2017 MLB season. I want to see all the players who hit 20 or more home runs, display their names, the number of Home runs they hit, and wOBA (weighted on base average). The data set should be sorted by wOBA and set to the variable name slug.

```
# load in mlb data
mlb <- read_csv("~/stat133/stat133-hws-fall17/post1/data/FanGraphs Leaderboard-51.csv")
```

```
## Parsed with column specification:
## cols(
##   .default = col_double(),
##   ` "Name"` = col_character(),
##   Team = col_character(),
##   G = col_integer(),
##   PA = col_integer(),
##   HR = col_integer(),
##   R = col_integer(),
##   RBI = col_integer(),
##   SB = col_integer(),
##   `BB%` = col_character(),
##   `K%` = col_character(),
##   `wRC+` = col_integer(),
##   playerid = col_integer()
## )
```

```
## See spec(...) for full column specifications.
```

```
# change first column name (troublesome as it is)
colnames(mlb)[1] <- "Name"

# change BB% and K% variables
mlb$`BB%` <- as.numeric(gsub("%","", mlb$`BB%`))
mlb$`K%` <- as.numeric(gsub("%","", mlb$`K%`))
```

```
# filter for 20 HR hitters
slug <- mlb[mlb$HR >= 20, c("Name", "HR", "wOBA")]
slug <- slug[order(slug$wOBA, decreasing = TRUE),]

head(slug, 10)
```

```
## # A tibble: 10 × 3
##                 Name    HR  wOBA
##                <chr> <int> <dbl>
## 1         Mike Trout    33 0.437
## 2        Aaron Judge    52 0.430
## 3      J.D. Martinez    45 0.430
## 4         Joey Votto    36 0.428
## 5       Bryce Harper    29 0.416
## 6   Charlie Blackmon    37 0.414
## 7         Matt Olson    24 0.411
## 8  Giancarlo Stanton    59 0.410
## 9    Freddie Freeman    28 0.407
## 10       Jose Altuve    24 0.405
```

I was able to generate the desired table in two lines of code but even in two lines of code you can start to see where things become cumbersome. Having to call on the same name twice is less than ideal and can make readability difficult for others. Even if I had chose a different name for the first line, I'm creating a variable and assigning a name for something I don't want in the end. Not to mention that the first line isn't ideally readable either. Filtering and selecting variables in this manner doesn't bode well for readability even for yourself if you were to go back and see what you had done.

Now let's do the same problem but with piping with the magrittr package.

```
# faster
slug <- mlb %>%
  filter(HR >= 20) %>%
  select(Name, HR, wOBA) %>%
  arrange(desc(wOBA))

head(slug, 10)
```

```
## # A tibble: 10 × 3
##                 Name    HR  wOBA
##                <chr> <int> <dbl>
## 1         Mike Trout    33 0.437
## 2        Aaron Judge    52 0.430
## 3      J.D. Martinez    45 0.430
## 4         Joey Votto    36 0.428
## 5       Bryce Harper    29 0.416
## 6   Charlie Blackmon    37 0.414
## 7         Matt Olson    24 0.411
## 8  Giancarlo Stanton    59 0.410
## 9    Freddie Freeman    28 0.407
## 10       Jose Altuve    24 0.405
```

Immediately you have the advantage of not having to create numerous variable names. Second you can easily make out code that is generally cluttered. The first line of code in the chunk above where I am filtering and selecting data is made clear thanks to the addition of piping and the simplicity of the dplyr functions. The addition of the indent adds immensely to the readability of the code as you can clearly follow the work that is being done for a particular object. If errors arise it is much easier to go through and identify where problems in your code may exist, saving you from possibly tweaking code that you shouldn't be.

As for what's going on, the result of a particular line of code is passed on as an argument in the next function/operation. The code for the first line would typically be filter(mlb, HR >= 20) but with the pipe there is no need to have to pass on the mlb object.

What's happening with the %>% notation

To give a visual sense of what the %>% notation is actually doing I'll go through the previous chunk showing a representation of what is going on.

```
# show what %>% is actually doing
mlb_pipe <- function(.) {
  . <- filter(. , HR >= 20)
  . <- select(. , Name, HR, wOBA)
  . <- arrange(., desc(wOBA))
}
mlb_pipe(mlb)
```

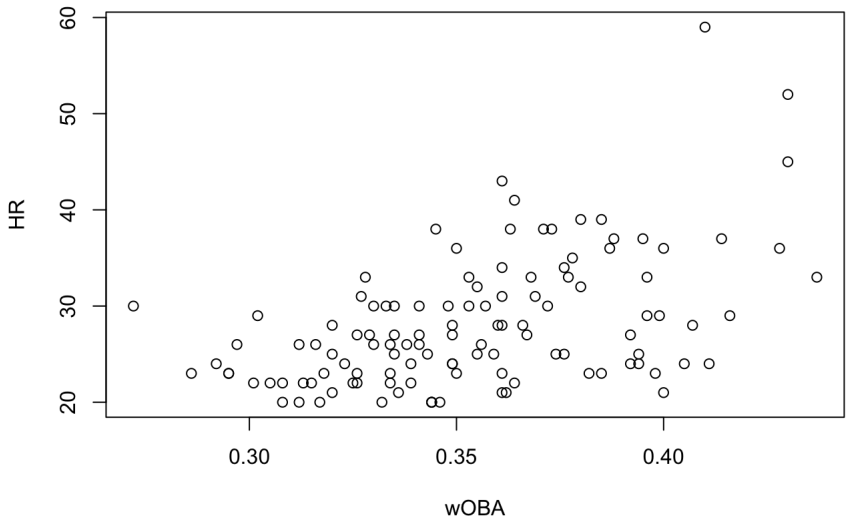
- The name overwriting still exists here but it's something going on behind the scenes. As a result the inherent flaw in piping is that you can only pass one variable through. You can not pass two variables through or pass one variable into a function that requires more inputs than given.

- Piping becomes something that is incredibly useful in simpler situations where you are working with functions that take in a single variable. It isn't something exclusive to dplyr functions as I've shown above. While the piping feature is prominent for dplyr, the piping feature in R is native to the magittr package.

Beyond just %>%

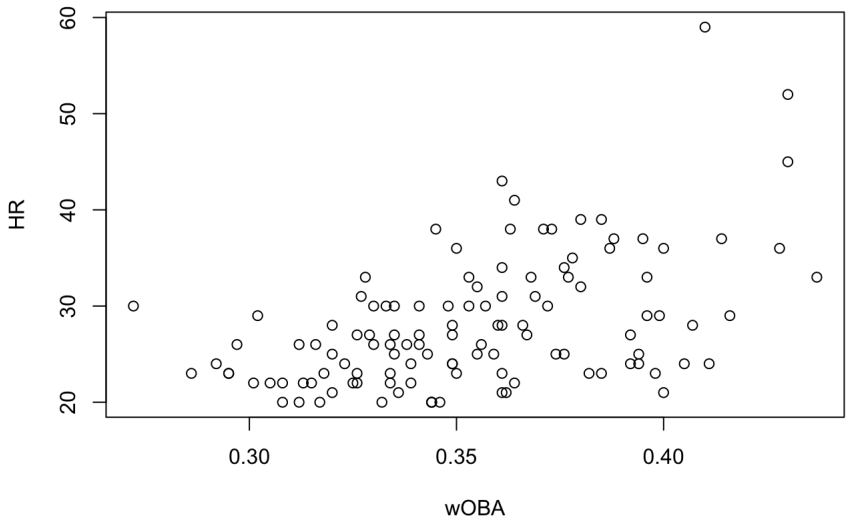
There are numerous advantages to piping but there is no inherent advantage when it comes to using %>% with plots

```
# for example, a plot of HR vs wOBA
plot(slug$wOBA, slug$HR, xlab = "wOBA", ylab = "HR")
```



Normally doing this with the %>% would be something messy but the magrittr package also has the helpful %$% function.

```
# and creating the same plot with the %$% notation
slug %$%
  plot(wOBA, HR)
```



The %$% notation works by taking in a data frame, list, or environment on the left hand side and allows you to select the names from that object on the right hand side. A continued benefit of working with piping is that I can easily filter out outliers and other data that is feel is giving me a poor plot. %$% works well for a function like plot() and is great for a function like lm() too.

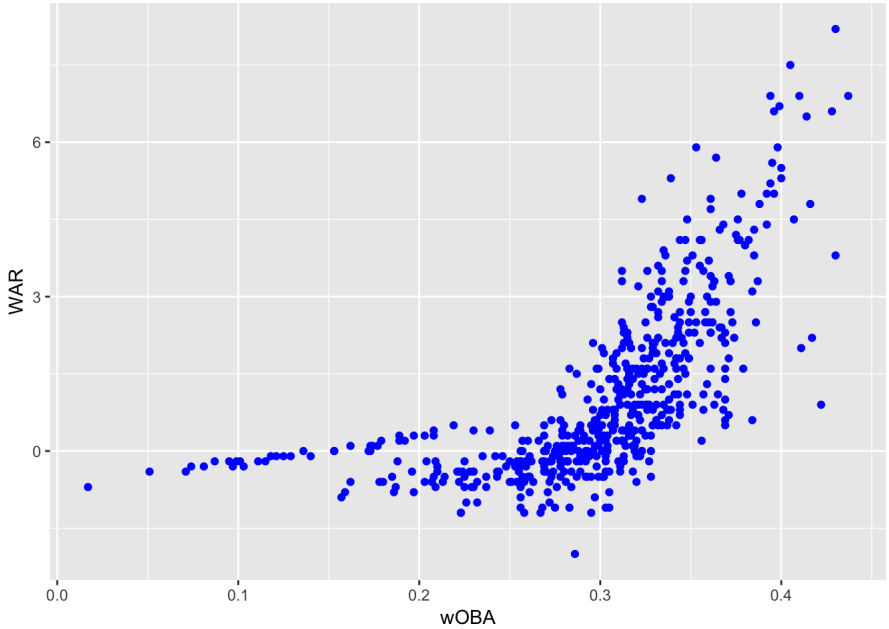
```
# more use of %$%, but with filtered data set first
mlb %>%
  filter(PA > 50) %$%
  summary(lm(WAR ~ PA + wOBA + Def))
```

```
##
## Call:
## lm(formula = WAR ~ PA + wOBA + Def)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -2.5269 -0.5134 -0.1542  0.4027  4.3051
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) -4.062619   0.211571  -19.20   <2e-16 ***
## PA           0.003564   0.000251   14.20   <2e-16 ***
## wOBA        13.198547   0.828704   15.93   <2e-16 ***
## Def          0.103513   0.006783   15.26   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.9403 on 540 degrees of freedom
## Multiple R-squared:  0.6924, Adjusted R-squared:  0.6907
## F-statistic: 405.1 on 3 and 540 DF,  p-value: < 2.2e-16
```

After first filtering for players who had more than 50 plate appearances, I was able to use the %$% function to pass through the multiple variable names into my regression equation. Again this is something that is great for me to quickly play around without having to create a mess of variable names. It helps keep things clean and if I wanted to quickly filter data.

I can use this notation to set up a ggplot as well.

```
# filter data first then set up ggplot
mlb %>%
  filter(PA > 50) %$%
  ggplot(.,aes(wOBA, WAR)) + geom_point(color = "blue") #scatterplot between WAR and wOBA variables
```



Here we see some of where the function falls short. In order to pass through the dataset I still had to specify the '.' as the data argument in this case. This is also a case where where I decided to use the %>% and %$% notation because I was starting with data that wasn't already filtered. It's great for short use and setting up a plot in a short amount of time but isn't ideal.

Some other useful notation comes in the form of %<>%. Generally when you transform part of a data frame it looks a little unintuitve to call on a variable name twice.

```
# manipulate
# mlb$HR <- log(mlb$HR)

# log manipulation using %<>%
mlb$HR %<>% log
```

%<>% is something that's simple enough but there is repetitiveness in having to write the same name multiple times. The %<>% function works by applying the right side function on the object on the left side. It's intuitive but can be a little confusing or misleading to those who aren't all too familiar with R. Regardless it's useful to eliminate repition in these kinds of cases and is faster to write out.

Conclusion

The functions in magrittr have their flaws but are incredibly useful for improving workflow. When it comes to manipulating data with dplyr, the %>% function does wonders for readability and is faster to implement. Even if you are working with more complex functions that require multiple arguments, playing with your data, filtering your data, and creating quick plots on the fly could not be easier.

The additional %$% and %<>% functions are wonderful when it comes to manipulating variables and selecting variables to pass into functions that require multiple arguments. They help keep the flow along with the %>% function and are intuitive to use.

The functions introduced in the magrittr package help make programing in R simpler and faster. Whatever they may lack in complex applications they make up in ease of use, readabilty, and cutting down time.

Thank you Hadley Wickham for all the work you've done with R, packages like magrittr make data science simpler and more accessible for everyone.

Sources

http://www.fangraphs.com/leaders.aspx?pos=all&stats=bat&lg=all&qual=0&type=8&season=2017&month=0&season1=2017&ind=0&team=0&rost=0&age=0&filter=&players=0

https://www.r-bloggers.com/more-readable-code-with-pipes-in-r/

http://r4ds.had.co.nz/pipes.html

https://www.r-bloggers.com/simpler-r-coding-with-pipes-the-present-and-future-of-the-magrittr-package/

https://cran.r-project.org/web/packages/magrittr/README.html

https://github.com/tidyverse/magrittr/blob/master/R/pipe.R

http://blog.revolutionanalytics.com/2014/07/magrittr-simplifying-r-code-with-pipes.html

http://hadley.nz