Introduction

As surprising as it may sound, many new R programmers are unaware of Base R syntax and the powerful functions that come with it- many of which do what functions from packages like `dplyr` and the rest of the tidyverse do. In this short blog post I am going to talk about the `within()` function and its synonmity of the `dplyr` package's `mutate()` function.

The lore that I heard around the `within()` function and `mutate()` stems from healthy competition between RStudio and the R-Core to create a function that does exactly what they do. If someone has a verifiable source for this I would love to see it myself!

Disclaimer: I am a big fan of the work Hadley Wickham and RStudio have done and have personally adopted a lot of tidy practices myself.This blog is to highlight a Base R function that I personally have overlooked and recently learned about. I want to thank my professor Georges Monette for showing me this. I would have never known about it without him!

For the example I will be using the all too classic `iris` data set for demonstration purposes.

The Question

Suppose we're interested in creating new variables that will give us the sepal length/width ratio for each flower measured. How would we do this?

Using `mutate()`

`mutate()` is generally written by piping the data set to the function and then defining the variables.

For brevity, I'll just show the first six rows.

```
library(dplyr)

iris %>% mutate(Sepal.LWRatio=Sepal.Length/Sepal.Width) %>% head()
```

Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species	Sepal.LWRatio
5.1	3.5	1.4	0.2	setosa	1.457143
4.9	3.0	1.4	0.2	setosa	1.633333
4.7	3.2	1.3	0.2	setosa	1.468750
4.6	3.1	1.5	0.2	setosa	1.483871
5.0	3.6	1.4	0.2	setosa	1.388889
5.4	3.9	1.7	0.4	setosa	1.384615

Using `within()`

Interestingly enough, we could do the same thing with `within()` as well! The syntax looks pretty similar to `mutate()`'s if I were to not utilize pipes.

```
head(
within(iris,{
  Sepal.LWRatio = Sepal.Length/Sepal.Width
})
)
```

Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species	Sepal.LWRatio
5.1	3.5	1.4	0.2	setosa	1.457143
4.9	3.0	1.4	0.2	setosa	1.633333
4.7	3.2	1.3	0.2	setosa	1.468750
4.6	3.1	1.5	0.2	setosa	1.483871
5.0	3.6	1.4	0.2	setosa	1.388889
5.4	3.9	1.7	0.4	setosa	1.384615

Interchangeability

Its important to note the way I wrote the above codes is only how I would personally use them. There are no hard rules for them. I can easily write the above codes with or without 'tidy' syntax.

```
# Looks like mutate()

iris %>% within({
  Sepal.LWRatio = Sepal.Length/Sepal.Width
}) %>% head()

# Looks like within()
head(
mutate(iris,
       Sepal.LWRatio = Sepal.Length/Sepal.Width)
)
```

Which one is faster

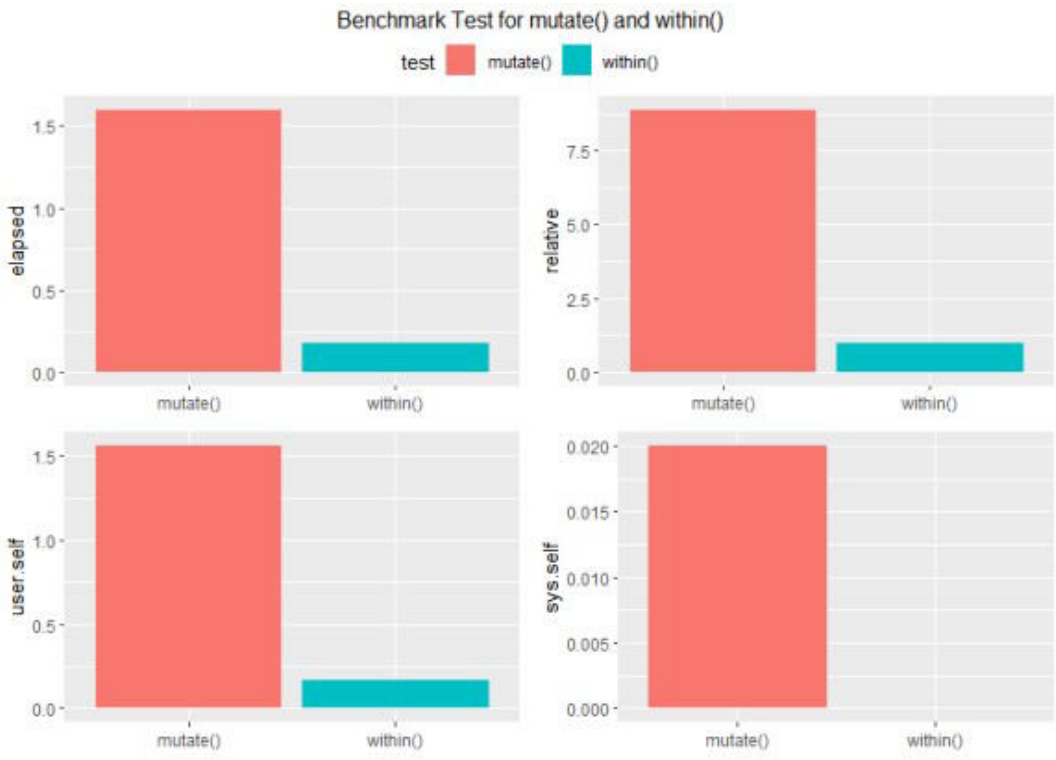
Now for what really matters. Which one is faster? For this we will use the `rbenchmark` package. To make the game as even as possible I will not incorporate any pipes which may slow the code down.

```
library(rbenchmark)

benchmark(
  'mutate()'=mutate(iris, Sepal.LWRatio = Sepal.Length/Sepal.Width),
  'within()'=within(iris,{Sepal.LWRatio<-Sepal.Length/Sepal.Width}),
  replications = 1000
  )
```

test	replications	elapsed	relative	user.self	sys.self	user.child	sys.child
mutate()	1000	1.60	8.889	1.56	0.02	NA	NA
within()	1000	0.18	1.000	0.17	0.00	NA	NA

Looking at this visually, we see that `within()` is much faster than `mutate()`. Very interesting indeed.



Which one uses more Memory.

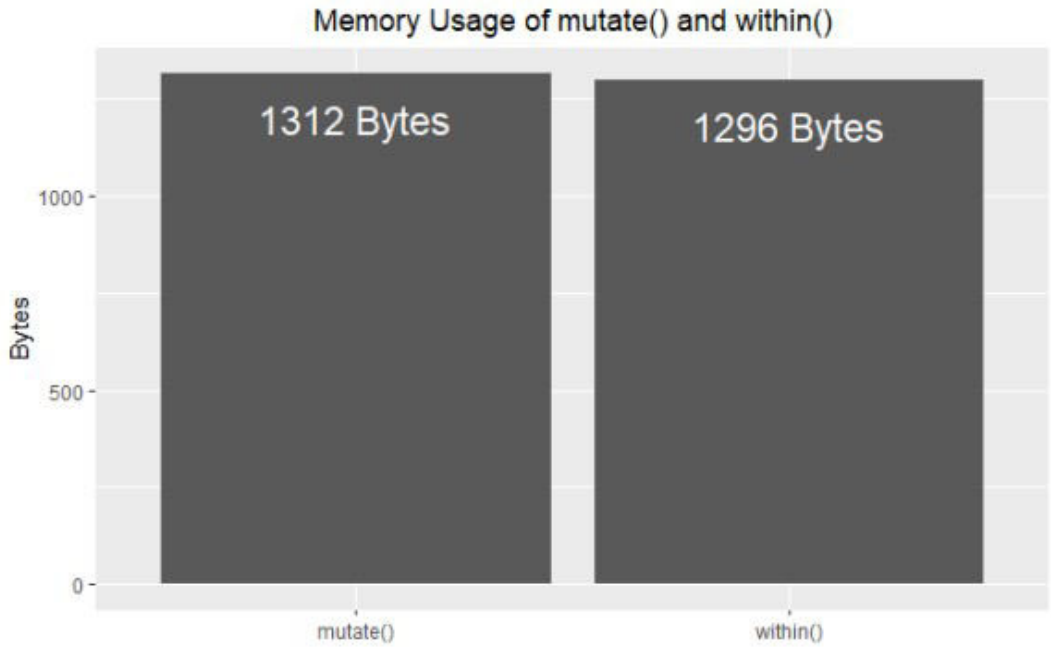
Now lets talk about production. In order to ensure that applications are lightweight, memory use is essential. In order to check how much memory is used, we will use Hadley Wickham's `pryr` package.

```
library(pryr)

df<- tibble::tibble(
  `Function`=c('mutate','within'),
   Bytes=c(object_size(mutate),object_size(within)))

df
```

Function	Bytes
mutate()	1312
within()	1296



`mutate()` is slightly larger than within in terms of memory, but the size difference is minute. In terms of application size, I wouldn't sweat it with that much of a size difference (unless you will be using `mutate()` multiple times in your code).

Conclusion

As far as I can see, Base R's `within()` is a faster alternative to `dplyr`'s `mutate()`. I'm sure there is something even faster out there with `data.table` but I have to still spend some time with it. (If you know some `data.table` code for this I would love to see it!)