R Design Patterns, Base-R vs. Tidyverse
With a view toward the teaching of R beginners

Norman Matloff
Dept. of Computer Science, University of California, Davis

August 21, 2022

This document enables the reader to see at a glance the difference between base-R and the tidyverse in common R design settings. I believe the base-R versions are generally simpler, thus more appropriate for R learners. I discuss this in detail in `http://github.com/matloff/TidyverseSkeptic/README.md`, where I cite many other problems besides lack of simplicity, but I believe the examples here begin to illustrate why Tidy is a bad vehicle to use in teaching R beginners. By the way, I use the latter term to mean R learners with no prior programming background.

All examples use R's built-in datasets. After e.g., changing a data frame, it is restored for the next example, e.g. **data(mtcars)**. The examples are presented roughly in order of how often these operations tend to be performed by R users.

As this document is aimed at comparing base-R and the tidyverse in terms of teaching new R learners, advanced functions from either base-R or the tidyverse are excluded here.

Reading a specific cell in a data frame

```
mtcars$mpg[3]
```

```
mtcars %>%
select(mpg) %>%
  filter(row_number() == 3)
```

Comment: The Tidy version could be shortened a bit by using select(mtcars,mpg) instead of mtcars %>% select(mpg), but the latter seems to be the preferred form, e.g. on the official tidyverse page, *https://dplyr.tidyverse.org/*.

Adding a column to a data frame

```
mtcars$hwratio
  <- mtcars$hp / mtcars$wt
```

```
mtcars %>%
mutate(hwratio=hp/wt) -> mtcars
```

Comment: Of course, typically Tidy coders would use <− rather than −>. I feel that the former is more consistent with the "left to right flow" of pipes. But in any case, the point about code complexity is the same either way.

Extracting rows

```
mtc8 <-
  subset(mtcars, cyl==8)
```

```
mtcars %>%
filter(cyl == 8) -> mtc8
```

Mean by group

```
tapply(mtcars$mpg,
  mtcars$cyl,mean)
```

```
mtcars %>%
group_by(cyl) %>%
  summarize(meanMPG =
    mean(mpg,))
```

Comment: Strangely, many Tidy advocates dismiss the **tapply()** function, treating it as a niche function in base-R. On the contrary, it is a workhorse in classic base-R applications. For example, consider the **ggplot2** package, written by the (later) inventor of the tidyverse, Hadley Wickham. Hadley calls **tapply()** 7 times in the **ggplot2** code!

Row means

```
rowMeans(EuStockMarkets)
```

```
EuStockMarkets %>%
  as.data.frame %>%
  rowwise() %>%
  mutate(m =
    rowMeans(across(everything())))
 %>% select(m)
```

Comment: The row means operation is quite common in R usage. Here the Tidy user must go to much more trouble than in base-R.

Row operations, custom function

```
mM <- function(x)
  max(x) - min(x)
apply(EuStockMarkets,1,mM)
```

```
mM <- function(x)
    max(x) - min(x)
EuStockMarkets %>%
  as.data.frame %>% rowwise() %>%
  mutate(m =
     mM(across(everything()))) %>%
  select(m)
```

Comment: The **apply()** function is of central importance in traditional R. Here again, the Tidy user must go to much more trouble.

Means, grouped by more than one variable

```
tapply(mtcars$mpg,
  list(mtcars$cyl,
    mtcars$am),
  mean)
```

```
mtcars %>%
  group_by(cyl,am) %>%
  summarize(m = mean(mpg))
```

Comment: In this example, the two sets of code are not fully comparable, as there is quite a difference in type of output:

```
> tapply(mtcars$mpg, list(mtcars$cyl, mtcars$am), mean)
      0        1
4 22.900 28.07500
6 19.125 20.56667
8 15.050 15.40000
> mtcars %>% group_by(cyl,am) %>% summarize(m = mean(mpg))
# Groups:   cyl [3]
    cyl    am     m
  <dbl> <dbl> <dbl>
1     4     0  22.9
2     4     1  28.1
3     6     0  19.1
```

```
4      6      1   20.6
5      8      0   15.0
6      8      1   15.4
```

The base-R form returns a 3x2 table, which is often what one needs for reports, research papers and so. The Tidy version is less useful in such contexts, but would be of value in some other applications.

Binary recoding of a vector

```
NileHiLow <-
  ifelse(Nile >= 1000,
    'high','low')
```

```
Nile %>% as.data.frame %>%
  mutate(
    HighLow = case_when
    (x < 1000~'low',
     x >= 1000~'high')
  ) %>%
  select(HighLow) %>%
  as.vector -> HighLow
```

Comment: The step of conversion back to a vector at the end is needed for many R packages in which vector input is required.

Deleting columns from a data frame

```
mtcars[c('drat','carb')]
  <- NULL
```

```
mtcars %>%
select(-c(drat,carb))
  -> mtcars
```

Isn't It More Reasonable to Teach a Mix of Base-R and Tidy?

I believe most people react skeptically to extreme ideologies. I've advocated teaching a mix of base-R and Tidy, but have yet to find a prominent Tidy advocate who agrees.

Here is an example. (The Tidy version appeared in a presentation by a Tidy advocate.) In the **mtcars** dataset, say we wish to recode the number of gears into English.

```
gr <- mtcars$gear
mtcars$gear <-
  case_when(
    gr == 3 ~ 'three',
    gr == 4 ~ 'four',
    gr == 5 ~ 'five')
```

```
mtcars %>%
 mutate(
   gear =
   case_when(
     gear == 3 ~ "three",
     gear == 4 ~ "four",
     gear == 5 ~ "five"
   )
 ) -> mtcars
```

Comment: A purely-base-R version might use nested **ifelse()** calls. This would be shorter than the Tidy version, but not as clear. But by using a blend of base-R and Tidy—use of '$' to reference a data frame column and use of the Tidy function **case_when()**—we still end up with shorter and simpler code, compared to the pure-Tidy version.