Data frame operations

Robert McDonald

There are typically several different ways to accomplish a specific task in R. The purpose of this document is to illustrate different ways to undertake operations on data frames. Even if you prefer one method and stick to it, the odds are that you will encounter examples and co-workers who have different preferences.

Here are the libraries we will use:

```
library(knitr)
library(tidyverse)
opts_chunk$set(collapse=TRUE,
               message=FALSE,
               comment=NA
               )
```

We begin by defining a data frame, named `df`:[1]

```
df <- data.frame(what=c('crust', 'cookie', 'cake', 'pasta'),
                 flour=c(3, 3, 1, 3),
                 butter=c(2, 2, 1, 0),
                 liquid=c(1, 0, 0, 0),
                 sugar=c(0, 1, 1, 0),
                 egg=c(0, 0, 1, 2),
                 stringsAsFactors=FALSE
                 )

dft <- tibble(what=c('crust', 'cookie', 'cake', 'pasta'),
              flour=c(3, 3, 1, 3),
              butter=c(2, 2, 1, 0),
              liquid=c(1, 0, 0, 0),
              sugar=c(0, 1, 1, 0),
              egg=c(0, 0, 1, 2)
              )
all.equal(df, dft, check.attributes=FALSE)
[1] TRUE
```

The `data.frame` function will automatically create strings as factors unless we tell it not to. Many would prefer character vectors not being treated as factors, so this can be an advantage of the `tibble` function.

The `all.equal` function will complain that `df` and `dft` have different classes unless we specify `check.attributes=FALSE`.

The data frame looks like this:

```
kable(df)
```

what	flour	butter	liquid	sugar	egg
crust	3	2	1	0	0
cookie	3	2	0	1	0
cake	1	1	0	1	1
pasta	3	0	0	0	2

[1]This examples was inspired by Ruhlman (2009). Ratios are by weight. Any errors are mine.

Data frame operations include:

- adding one or more new column
- deleting one or more columns
- adding one or more new rows
- deleting one or more rows
- renaming columns

We will illustrate ways to do these in base R and in `dplyr`.

Extract a column

Suppose we want to extract the flour column and have it be a vector.

```
c1 <- df[, 'flour']
c2 <- df$flour
c3 <- df[['flour']]
c4 <- df['flour'][[1]]
class(c1)  ## this is a vector
[1] "numeric"
all.equal(c1, c2)
[1] TRUE
all.equal(c1, c3)
[1] TRUE
all.equal(c1, c4)
[1] TRUE
```

Suppose we want to extract the `flour` column and keep it as a dataframe (a list).

```
c5 <- df['flour']
c6 <- df %>% select(flour)
class(c5)  ## this is a dataframe
[1] "data.frame"
all.equal(c5, c6)
[1] TRUE
```

It is *very important* to understand that in base R, `df[, 'flour']` and `df['flour']` are different operations. They appear similar, but the first extracts and the second subsets:

```
c1 <- df[, 'flour']
c5 <- df['flour']
all.equal(c1, c5)  ## a vector and dataframe are not the same!
[1] "Modes: numeric, list"
[2] "Lengths: 4, 1"
[3] "names for current but not for target"
[4] "Attributes: < target is NULL, current is list >"
[5] "target is numeric, current is data.frame"
```

The object `c1` is a vector. The object `c5` is a subset, a new dataframe consisting of the `flour` column from the original dataframe. When you perform operations on a `tibble`, you get a new `tibble`. If you want a vector, you have to ask for a vector by using the extraction function `[[1]]` (which I recommend) or by using the dplyr `pull` function‘.

```
test1 = dft[, 'flour']
test2 = dft['flour']
test3 = dft %>% select(flour)
all.equal(test1, test2)
```

```
[1] TRUE
all.equal(test1, test3)
[1] TRUE
all.equal(c1, test1[[1]])
[1] TRUE
all.equal(c1, pull(test1))
[1] TRUE
```

Add a column

We will add "nuts" and "yeast" columns:

```
df1 <- df2 <- df3 <- df  ## create copies of the original data frame
df1$nuts <- c(0, 0.5, 0, 0)
df1$yeast <- 0  ## the recycling rule in action
df2[, "nuts"] <-  c(0, 0.5, 0, 0)
df2[, "yeast"] <- 0
df3 <- df %>% mutate(nuts=c(0, 0.5, 0, 0), yeast=0)
all.equal(df1, df2)
[1] TRUE
all.equal(df2, df3)
[1] TRUE
kable(df1)
```

what	flour	butter	liquid	sugar	egg	nuts	yeast
crust	3	2	1	0	0	0.0	0
cookie	3	2	0	1	0	0.5	0
cake	1	1	0	1	1	0.0	0
pasta	3	0	0	0	2	0.0	0

Very important: Notice that when using `dplyr` to construct `df3`, no quotes are needed. When using base R, quotes are needed for column names. The absence of quotation within the `dplyr` universe is great, but it can be very tricky (and frustrating) to mix `dplyr` and base R. My advice is to stick with one or the other as much as possible.

Delete columns

Now we decide we don't need the columns we just added:

```
df4 <- df1;
df4[c('nuts', 'yeast')] <- NULL  ## assigning to `NULL` deletes an object
df5 <- df1[-c(7, 8)]
df6 <- df1[-which(names(df1) %in% c('nuts', 'yeast'))]
df7 <- df1[, -which(names(df1) %in% c('nuts', 'yeast'))]
df8 <- df1 %>% select(-nuts, -yeast)
all.equal(df4, df5)
[1] TRUE
all.equal(df4, df6)
[1] TRUE
all.equal(df4, df7)
[1] TRUE
```

```
all.equal(df4, df8)
[1] TRUE
```

Add rows

We will use the dataframe including nut and yeast columns.

```
newrow1 <- list('bread', 1, 0, .67, 0, 0, 0, .02)
newrow2 <- list(what='bread', flour=1, butter=0,
                liquid=.67, sugar=0, egg=0, nuts=0, yeast=.02)
df.ar1 <- rbind(df1, newrow1)
df.ar2 <- rbind(df1, newrow2)
df.ar3 <- bind_rows(df1, newrow2) ## dplyr
all.equal(df.ar1, df.ar2)
[1] TRUE
all.equal(df.ar1, df.ar3)
[1] TRUE
```

Filter rows

We can choose to keep or delete rows that meet specific criteria. We will use `df.ar1` as the base dataframe.

Suppose we want only items that use butter

```
df.butter1 <- df.ar1[df.ar1$butter > 0, ]
df.butter2 <- df.ar1[df.ar1$"butter" > 0, ]
df.butter3 <- subset(df.ar1, df.ar1$butter > 0)
df.butter4 <- df.ar1 %>% filter(butter > 0)
all.equal(df.butter1, df.butter2)
[1] TRUE
all.equal(df.butter1, df.butter3)
[1] TRUE
all.equal(df.butter1, df.butter4)
[1] TRUE
```

Note that any expression evaluating to a logical will work. So compound conditions (e.g. `df.ar1$butter > 0 & df.ar1$liquid > 0`) will work fine.

References

Ruhlman, Michael. 2009. *Ratio: The Simple Codes Behind the Craft of Everyday Cooking*. Scribner.