Fifth article: Review of getting subsets of a data frame, constructing data frames

Alan E. Berger December 9, 2020

version 1

available at https://github.com/AlanBerger/Practice-programming-exercises-for-R

Motivation

Data frames, which are analogous to Excel spreadsheets for which the entries in each column are of the same "type" and each column has the same number of rows, are a fundamental way of handling data in R.

I'll first review various ways of extracting subsets of a data frame, and then review several ways to construct a data frame from multiple vectors or from smaller data frames. This is a fairly long article - it is not intended to be read all at one time.

R often has several equivalent ways of doing something. Perhaps?? this is from R having been developed in a collaborative fashion by several people with their own favorite ways of doing various programming constructs, so they all got included. One can choose one's favorite ways of doing things, but one needs to be familiar with all the commonly used constructs in order to be able to understand code written by others (and, importantly, to understand code in instructions and examples for R packages one wants to use).

Review of how to get specified subsets of a data frame: A. getting a single column

This material is taken/modified from a pinned post of mine "Examples of extracting as a vector a single column from a data frame" in the Week 2 Discussion Forum for the R Programming Course in the Johns Hopkins Data Science Specialization on Coursera.

If you are taking this course, then note in the Week 1 pinned posts in the Discussion Forum, that Leonard Greski has written a very good more general article on getting row and column subsets from a data frame "Forms of the Extract Operator in R" (this article also contains some more advanced material covered later in the R course, so read what is relevant to where you are in the R Programming course and refer back later as you learn more R); and one is also well advised to read Al Warren's pinned post in the Week 2 Discussion Forum "Subsetting with bracket notation".

Getting (often referred to as *extracting*) a single column from a data frame is a common step in an R function, and one usually will want to get the column in the form of a vector, not as a data frame with that one column.

Let's see how to get, as a vector, for example the sulfate column of a simple example data frame;

```
df <- data.frame(sulfate = c(4.79, 1.46, 4.28, NA), nitrate = c(0.299, NA, 4.280, 3.560))
df
  sulfate nitrate
1    4.79   0.299
2    1.46      NA
3    4.28   4.280
4      NA   3.560
```

To get the sulfate column as vector you can do either of the following 5 equivalent statements:

```
df[["sulfate"]] # double brackets
[1] 4.79 1.46 4.28   NA
# or
```

```
df$sulfate  # note sulfate does not need to be in quotes for the $ form of extraction
#            (but a text string with blanks in it would need to be)
[1] 4.79 1.46 4.28   NA
# or
df[, "sulfate"]  # single brackets but note the comma so we get all the rows in the sulfate column
[1] 4.79 1.46 4.28   NA
# or one could use the column number
df[[1]]
[1] 4.79 1.46 4.28   NA
df[, 1]
[1] 4.79 1.46 4.28   NA

# Note that df["sulfate"] # single brackets, no comma,
# is a 1 column data frame containing the sulfate column; if you are getting
# 1 column from a data frame you will usually want it as a vector

df["sulfate"] # single brackets gives a data frame
  sulfate
1    4.79
2    1.46
3    4.28
4      NA

class(df["sulfate"])
[1] "data.frame"
# Note for example the mean function "expects" a vector and
# will return NA and give a not very informative message if you
# "feed it" a data frame

mean(df["sulfate"])
[1] NA
Warning message:
In mean.default(df["sulfate"]) :
  argument is not numeric or logical: returning NA

# If pollutant is an R variable containing the text string "sulfate"
# then these will work to extract the column as a vector
pollutant <- "sulfate"
df[[pollutant]]
[1] 4.79 1.46 4.28   NA
# or
df[, pollutant]
[1] 4.79 1.46 4.28   NA

# BUT NOT
df$pollutant
NULL
```

`df$pollutant` does NOT work since pollutant is NOT an actual column name; it is a variable *containing* the text string sulfate which is not acceptable for the $ form of getting/extracting a column from a data frame as a vector (those are the R "rules" and we have to live with them). And note R does NOT even warn you about this type of mistake - it just cheerfully gives back NULL which can lead to v e r y mysterious bugs. Similarly, mistyping the name of a column in the following example commands results in NULL (with NO warning): `df$sulfffate` and also `df[["sulffffate"]]`. Programming requires very careful attention

to details - one might be tempted to think R should be able to "figure out" what you meant, but recall what type of mischief an "auto correct" in a word processor or message app can create - and in a programming language you wouldn't even get to view in real time what the "compiler" had done to your code. Better to know that if you program correctly exactly what you want, some "gremlin" won't be changing it!

Review of how to get specified subsets of a data frame: B. subsetting rows and/or columns

If v is a vector of row indices (that are in the range of the number of rows of the data frame df) one can get the rows of df corresponding to v

```
df <- data.frame(sulfate = c(4.79, 1.46, 4.28, NA), nitrate = c(0.299, NA, 4.280, 3.560))
df
  sulfate nitrate
1    4.79   0.299
2    1.46      NA
3    4.28   4.280
4      NA   3.560

v <- c(1, 3, 2, 2, 2)
df[v, ]
    sulfate nitrate
1      4.79   0.299
3      4.28   4.280
2      1.46      NA
2.1    1.46      NA
2.2    1.46      NA

# note reordering and repeats are allowed
# note the indication of repeats in the row numbers R generates (R "does not like"
# duplicate row names and so does modifications to make them unique)
```

Note R does not issue warnings or errors for indices that are "out of range", it just fills in NA's

```
v <- c(1, 3, 2, 6, 2)  # 6 is out of the range of the number of rows of df
df[v, ]
    sulfate nitrate
1      4.79   0.299
3      4.28   4.280
2      1.46      NA
NA       NA      NA
2.1    1.46      NA
```

One can use negative row indices to **exclude** those rows:

```
df
  sulfate nitrate
1    4.79   0.299
2    1.46      NA
3    4.28   4.280
4      NA   3.560
```

```
v <- c(-1, -3)  # exclude rows 1, 3 and keep the rest
df[v, ]
  sulfate nitrate
2    1.46      NA
4      NA    3.56
```

One can also specify desired columns (and repeats of columns)

```
v <- c(1, 3, 2, 2, 2)
# if we just want the first column (sulfate) with these rows
# we can do
df[v, 1]
[1] 4.79 4.28 1.46 1.46 1.46
# or
df[v, "sulfate"]
[1] 4.79 4.28 1.46 1.46 1.46
# or
df[v, ]$sulfate
[1] 4.79 4.28 1.46 1.46 1.46

# we can also specify several columns
w <- c(1, 2, 1, 2)
df[v, w]
    sulfate nitrate sulfate.1 nitrate.1
1      4.79   0.299      4.79     0.299
3      4.28   4.280      4.28     4.280
2      1.46      NA      1.46        NA
2.1    1.46      NA      1.46        NA
2.2    1.46      NA      1.46        NA
```

Note R "does not like" duplicate column names and so does modifications to make them unique.

Also one can use a logical vector V having the same number of rows as df; rows where the corresponding entry of V is TRUE are kept, rows where the corresponding entry of V is FALSE are not kept.

```
df
  sulfate nitrate
1    4.79   0.299
2    1.46      NA
3    4.28   4.280
4      NA   3.560

logicalVector <- c(T, F, F, T)
df[logicalVector, ]
  sulfate nitrate
1    4.79   0.299
4      NA   3.560
```

The which function

If one has a vector V, one can ask for which rows of V is some logical condition TRUE; R's **which** function does this: the conceptual description is

which(some logical condition on each entry of V)

returns the vector of the **indices** of V for which the the condition is TRUE (Any entries of V that are NA will be considered to yield FALSE, so those indices of V will **not** be included in the result.) If there are no indices for which the condition is TRUE, the which function returns an empty integer vector (integer(0))

For example

```
df
  sulfate nitrate
1    4.79   0.299
2    1.46      NA
3    4.28   4.280
4      NA   3.560

result <- which(df[["sulfate"]] > 2)
result
[1] 1 3
# any entries of V that are NA are considered to yield FALSE

# one can then use result to get the rows of df for which the condition was TRUE
df[result, ]
  sulfate nitrate
1    4.79   0.299
3    4.28   4.280

# Note the result of having an NA involved in the following:

df  # repeating what df is
  sulfate nitrate
1    4.79   0.299
2    1.46      NA
3    4.28   4.280
4      NA   3.560

V <- df[["sulfate"]] > 2  # a logical vector with a value for each row of df
V
[1]  TRUE FALSE  TRUE    NA

df[V, ]  # keeps rows of the data frame where V is TRUE, but note the effect of the NA
   sulfate nitrate
1     4.79   0.299
3     4.28   4.280
NA      NA      NA
```

I like the conceptual viewpoint of the which function. Apparently it is rather universal in that, for example, **IDL** has the corresponding function called **where** and **MATLAB** has a corresponding function called **find**

The %in% function

The **%in%** function addresses the question of whether or not each entry of some vector v occurs in another vector w. It returns a logical vector z with z[k] being TRUE if v[k] is equal to some entry in w, and z[k] FALSE if v[k] is not equal some entry in w. This can be used to obtain a logical vector for use in selecting rows of a data frame

An example of how %in% behaves:

```
?"%in%"  # look at the help on %in% Note because of the special
#            character % one needs to "protect" %in% by enclosing it in
#            either quotes or apostrophes when "asking for help" on it

v <- c(1, 2, 3, 4, 5, NA)
w <- c(12, 3, 8, 22, 4)
v %in% w
[1] FALSE FALSE  TRUE  TRUE FALSE FALSE

v <- c(1, 2, 3, 4, 5, NA)
w <- c(12, 3, 8, 22, 4, NA)
v %in% w
[1] FALSE FALSE  TRUE  TRUE FALSE  TRUE

# note %in% will declare a match for an NA in v if there is an NA in w
```

Creating data frames

Reading in a data file as a data frame

As noted above, data frames are a fundamental way that R handles data. Many data files that are text files (as opposed to binary files) are naturally suitable for reading in as a data frame using for example **read.csv** (where the column separator (delimiter) is a comma), or more generally **read.table** The options for read.table also apply for read.csv but note some of the important default choices are different, in particular the default for "telling R" whether there is a header line containing column names is **header = TRUE** for read.csv and **header = FALSE** for read.table, and the default column separator for read.csv is `sep = ","` while for read.table one should usually specify it since the default is "white space"; a common column separator other than comma is a tab, which is specified by `sep = "\t"` (the backslash "tells" R to interpret the t in a special way).

To start with, when learning R, there are 2 other options one should be aware of: **stringsAsFactors = FALSE** "tells" R to read in character columns as character data, not as factors which is the default (unless a column is to be used as a factor in a statistical analysis it is likely better to have it read in as character data).

The second option one should be aware of when starting to learn read.csv and read.table is **na.strings** This option lets one specify the character string (or several character strings) that should be interpreted as NA (the default is `"NA"`). For example `na.strings = c("NA", "data is missing", "not available", "the experimenter dropped the sample", "the experimenter was texting when the data should have been measured")` If there is character data other than NA signifying missing data in a column that should be read in as numeric data, and R is not "informed" about this, then R will read in that column as factor or character data, which can lead to "issues" that are best avoided by properly reading in the data.

Another option to be aware of if one is dealing with a file that has "non-standard for R" column names is **check.names** which if set equal TRUE (the default), then R will modify read in column names to conform with what R considers standard. That means blank spaces and many characters that are not letters will get replaced by a period. I find this rather annoying since I like to use long descriptive column headers in files I create, and as long as I am not having R use the column names other than to write them back out after I have done some analysis on the data, it is OK to instruct R to leave the column names alone (by setting check.names = FALSE).

Creating a data frame from smaller data frames and/or vectors and matrices: A. the data.frame function

Looking over the R help on **data.frame**, one sees that it can combine objects that are or can be converted to be data frames into one combined data frame. As with read.csv and read.table, one may well want to use the option **stringsAsFactors = FALSE** (unless one needs to have one or more factor columns). (The data frame function will do *recycling* on rows but I would recommend having the number of rows in objects being combined into a data frame all be the same.) Note the R **rep** (replicate) function can be used to replicate "patterns", for example

```
rep(c(1,2), times = 4)  # repeat the pattern 4 times
[1] 1 2 1 2 1 2 1 2

# rep can also be used this way:
rep(c(1,2), each = 4)
[1] 1 1 1 1 2 2 2 2
```

Some examples with the data.frame function:

```
df1 <- data.frame(sulfate = c(4.79, 1.46, 4.28, NA), nitrate = c(0.299, NA, 4.280, 3.560))
df1  # our continuing example data frame
  sulfate nitrate
1    4.79   0.299
2    1.46      NA
3    4.28   4.280
4      NA   3.560

v1 <- c(1,2,3,4)
v2 <- c(TRUE, FALSE, TRUE, TRUE)
v3 <- c("a", "b", "c", "d")

m1 <- matrix(1:12, nrow = 4, ncol = 3)
m1
     [,1] [,2] [,3]
[1,]    1    5    9
[2,]    2    6   10
[3,]    3    7   11
[4,]    4    8   12

df <- data.frame(df1, v1, v2, m1, v3, stringsAsFactors = FALSE)
df
  sulfate nitrate v1    v2 X1 X2 X3 v3
1    4.79   0.299  1  TRUE  1  5  9  a
2    1.46      NA  2 FALSE  2  6 10  b
3    4.28   4.280  3  TRUE  3  7 11  c
4      NA   3.560  4  TRUE  4  8 12  d

# The row names of df are the row names of the first argument of data.frame, i.e.,
# the first of the objects being combined into one data frame

# If one wants to change some column names one could do, for example,

colnames(df)[c(5, 6, 7)] <- c("m1", "m2", "m3")
```

```
df
  sulfate nitrate v1    v2 m1 m2 m3 v3
1    4.79   0.299  1  TRUE  1  5  9  a
2    1.46      NA  2 FALSE  2  6 10  b
3    4.28   4.280  3  TRUE  3  7 11  c
4      NA   3.560  4  TRUE  4  8 12  d

# and similarly with row names
rownames(df) <- c("r1", "r2", "r3", "r4")
df
   sulfate nitrate v1    v2 m1 m2 m3 m4
r1    4.79   0.299  1  TRUE  1  5  9  a
r2    1.46      NA  2 FALSE  2  6 10  b
r3    4.28   4.280  3  TRUE  3  7 11  c
r4      NA   3.560  4  TRUE  4  8 12  d
```

Creating a data frame from smaller data frames and/or vectors and matrices: B. rbind

If one has several data frames that have the same number of columns AND the same column names, then one can "stack them vertically" using the **rbind** (row bind) function. For example with 2 data frames df1 and df2

```
df1 <- data.frame(sulfate = c(4.79, 1.46, 4.28, NA), nitrate = c(0.299, NA, 4.280, 3.560))
df1
  sulfate nitrate
1    4.79   0.299
2    1.46      NA
3    4.28   4.280
4      NA   3.560

df2 <- data.frame(sulfate = c(24.79, 21.46, 24.28, NA), nitrate = c(2.299, NA, 2.280, 2.560))
df2
  sulfate nitrate
1   24.79   2.299
2   21.46      NA
3   24.28   2.280
4      NA   2.560

# then one can do
df <- rbind(df1, df2)
df
  sulfate nitrate
1    4.79   0.299
2    1.46      NA
3    4.28   4.280
4      NA   3.560
5   24.79   2.299
6   21.46      NA
7   24.28   2.280
8      NA   2.560

# The column names for all the items being "rbinded" must be the same
```

```
# (except for an important exception described below).

# will get an error message if the column names don't match: for example below I have
# the column names in df2 not matching those in df1

colnames(df2)[2] <- "another.name"
df2
  sulfate another.name
1   24.79        2.299
2   21.46           NA
3   24.28        2.280
4      NA        2.560

df <- rbind(df1, df2)  # gets error message
Error in match.names(clabs, names(xi)) :
  names do not match previous names
```

Creating a data frame from smaller data frames and/or vectors and matrices: C. Using rbind in a loop

In some circumstances one might be reading in or constructing a succession of data frames, each with the same number of columns and the same column names, and want to combine them vertically. One can do this in a loop if one initializes an empty data frame via `df <- data.frame()`

One can rbind any data.frame to this empty data frame; this is the exception to the rule on same number of columns and column names so then this "conceptual" for loop will work:

```
df <- data.frame()  # initialize an empty data frame
for (i in some.set) {
   read in or derive a data frame dfi corresponding to i (each dfi must have the same
   number of columns and the same column names)
   df <- rbind(df, dfi)
}

# after this loop the data frame df will consist of all the data frames dfi stacked vertically
```

Creating a data frame from smaller data frames and/or vectors and matrices: D. the cbind function

The **cbind** (column bind) function can combine data frames or combine objects that are or can be converted to be data frames. cbind is the same as data.frame except that the default for cbind is check.names = FALSE

Hope this review was informative. The next set of exercises will get into practicing using and creating data frames.

= =

Some of the material above (Review of how to get specified subsets of a data frame: getting a single column) was taken/modified from a post of mine in the Discussion Forum for the R Programming Course

in the Johns Hopkins Data Science Specialization on Coursera, as noted above. As such Coursera and Coursera authorized Partners retain additional rights to that material as described in their "Terms of Use" https://www.coursera.org/about/terms

Note the reader should not infer any endorsement or recommendation or approval for the material in this article from any of the sources or persons cited above or any other entities mentioned in this article.