convert matrix to datafram

Paul Bradshaw

27 March 2017

To convert to matrix use `as.matrix` . However, if the first column contains words, then it will make *all* data text, so you need to omit the first column:

```
regionsmatrix <- as.matrix(regions[,-1])
```

This means you have no row names:

```
rownames(regionsmatrix)
```

```
## NULL
```

You can get the names from the columns and put them in as row names:

```
rownames(regionsmatrix)  <- colnames(regionsmatrix)
rownames(regionsmatrix)
```

```
##  [1] "North.East"               "North.West"
##  [3] "Yorkshire.and.The.Humber" "East.Midlands"
##  [5] "West.Midlands"            "East"
##  [7] "London"                   "South.East"
##  [9] "South.West"               "Wales"
## [11] "Scotland"                 "Northern.Ireland"
```

```
#added by Steve
regionnames <- colnames(regionsmatrix)
```

Now we have the rows and columns both using the region names, and the data is numerical. *And* it's all in a matrix.

But how do we convert this to a format which can be used for network graphs?

Data formats for network graphs

Network graphs normally need two sets of data:

1. A list of the entities to be graphed
2. A table of the relationships between the entities

Google Fusion Tables's network charts functionality only needs the second. But Kumu and D3 network charts, for example, need both.

The table of relationships is the key part: the list of entities can always be generated from that (a pivot table, for

example, will do this).

Normally that table has at least two columns:

1. Column 1: the entity that the relationship comes *from*
2. Column 2: the entity that the relationship goes *to*
3. An optional column 3 might specify the *strength* of the relationship (for example, the amount of money, or the numbers of people, going from entity 1 to entity 2)
4. There might also be a column with some sort of *classification* of the relationship (for example, directorship, or family relationship)

We have data showing what regions people come from and go to, but it's not in that format.

How do we get it in that format? It's good to mock up an example first so you know what it needs to look like. Here is what we need:

144 rows (12 x 12 relationships = 144) of 3 columns (from, to, amount).

The first row will be the 12 regions, in sequence, 12 times (North East, North West, Yorkshire…). The second row will be each region, 12 times, in sequence (North East, North East, North East…). The third row will be the numbers

This last row can actually be generated by using `array` like so, which turns a table into a vector:

```
regionnums <- array(regionsmatrix)
```

What about the other columns? One way to get it in that format is to use loops.

A basic loop

A basic `for` loop looks like this: `for (item in list of items) { do something each time}`

Here, for example, are two lines:

1. Create an empty vector variable called `rs`
2. Run a `for` loop which goes through the range of numbers 1 to 12, and for each adds it to that previously empty vector variable. (Note: apparently it's not ideal to add to vectors this way, but I'm doing it this way for now (https://stackoverflow.com/questions/22235809/append-value-to-empty-vector-in-r)).

When the loop has run 12 times (once for each item in that range of numbers), the vector contains all 12 numbers.

```
rs <- c()
for (r in c(1:12)) { rs <-c(rs, r) }
```

Of course we don't want a list of numbers. Instead we can use those numbers to serve as *indexes* to grab items from another vector, like so:

```
#First, make sure the rs variable is empty again, or we'll add to what we created bef
ore
rs <- c()
for (r in c(1:12)) { rs <-c(rs, regionnames[r]) }
```

The key part here is `regionnames[r]`: this uses `[r]` as an index to specify the item position in `regionnames` to grab. The first time the loop runs, when `r` is `1`, this means `regionnames[1]` (the first item in the vector `regionnames`)

Now, we need to convert our list of 12 regions into a list of 144 cells: each 12 regions must appear 12 times. To do that we need to put another loop *inside* of the loop above, like so:

```
#First, make sure the rs variable is empty
rscol1 <- c()
for (r in c(1:12))
  { for (r in c(1:12))
    { rscol1 <-c(rscol1, regionnames[r]) }
  }
```

This is harder to get your head around, so break it down:

- The first loop starts at `1`
- While it is `1`, it then begins the second loop, and goes from 1 to 12. This grabs all 12 names from the vector and adds them once.
- The first loop then moves on to `2`
- The second loop *again* grabs the 12 names from the vector, and appends them, so we now have the same 12 names twice, in sequence.
- This process repeats so we end up with the list of 12 names appended 12 times to that original empty list.

The loop for the second column

Here's the loop to create the second column. What's changed? Well, apart from the name of the vector being created: `rscol2`

```
rscol2 <- c()
for (r1 in c(1:12))
  { for (r2 in c(1:12))
    { rscol2 <-c(rscol2, regionnames[r1]) }
  }
```

The difference is that in the previous code `r` was used for the number being grabbed in both loops. This time we're differentiating: the first loop calls each item `r1` as it cycles through. The second loop: `r2`.

What this means is that while the second loop is running 12 times, the actual *number* (1, 2, 3, whatever) isn't *used* for anything. Instead `r1` is used as the *index*. So it runs 12 times using the index 1, then 12 times using the index 2, and so on. This generates a list that looks different.

Combining the results into a data frame

We now have three vectors containing our three columns. To combine them into a data frame we use… `data.frame`:

```
networktable <- data.frame(rscol1, rscol2, regionnums)
#Rename the columns
column_names <- c('destination', 'origin', 'numbers')
colnames(networktable) <- column_names
write.csv(networktable, 'networktable.csv')
```

We can make this adapt to another similar matrix by using the length of the column as the end of our range.

Saving the code as a reusable R script

If we expect to run this in other projects and not just this one, we can save it as an R script and then run it in different projects.

To do this, select **File > New File > R Script**. Paste your code in there, and then save it (it will save in the same directory as the current R project).

To run the script use the `source` command followed by the name that you used for the script, in inverted commas. If you called the script 'squaretabletolong', for example, the code to run that script would be:

```
source('squaretabletolong.R')
```

Creating a function instead of a script

It might be easier to save the code in a function instead of a script.

Here's the code to create a function that takes 2 pieces of information (the number of columns, and a list of headings) and returns a column for you that contains those headings multiplied by the number of columns:

```
squaretotable <- function(colnum,headers){
  #create empty vector, which will be filled and returned
  col1 <- c()
  #first loop
  for (i in c(1:colnum)){
    #second loop
    for (a in c(1:colnum)) {
      col1 <- c(col1, headers[a])
    }
  }
  return(col1)
}
```

Running the function looks like this:

```
col1 <- squaretotable(12,regionnames)
```

Or indeed:

```
#Note that we can call the results anything we want - col1 is just used within the fu
nction
col2 <- squaretotable(length(regionnames),regionnames)
```

In fact, if we know that the number is always going to be the length of the list of column headings, we only need one variable in the function, like so:

```
squaretotable2 <- function(headers){
  #create empty vector, which will be filled and returned
  col1 <- c()
  colnum <- length(headers)
  #first loop
  for (i in c(1:colnum)){
    #second loop
    for (a in c(1:colnum)) {
      col1 <- c(col1, headers[a])
    }
  }
  return(col1)
}
```

And test:

```
col3 <- squaretotable2(regionnames)
```

Expanding the function to convert the whole table

That function only generates one column of what we end up with. Can we expand it to do the whole thing?

First we need to list the tasks to complete:

1. We have a square table (in this case 12x12).
2. Column 1 needs to contain the headings, each of which is repeated once, 12 times
3. Column 2 needs to contain the headings, each of which is repeated 12 times, once
4. Column 3 needs to contain the data as a single array

```
squaretotableall <- function(headers, yourtable){
  #create empty vector, which will be filled and returned
  col1 <- c()
  colnum <- length(headers)
  #first loop
  for (i in c(1:colnum)){
    #second loop
    for (a in c(1:colnum)) {
      col1 <- c(col1, headers[a])
    }
  }
  col2 <- c()
    #first loop
  for (i in c(1:colnum)){
    #second loop
    for (a in c(1:colnum)) {
      col2 <- c(col2, headers[i])
    }
  }
  #convert to matrix
  yourmatrix <- as.matrix(yourtable[,-1])
  #create an array
  col3 <- array(yourmatrix)
  newtable <- data.frame(col1, col2, col3)
  return(newtable)
}
```

Now to test

```
newtable <- squaretotableall(regionnames,regions)
```

We can now copy the function code to the self-contained R script so when called it adds the function to any R project, which can then be called.

Simplifying further

We could further improve this function by extracting the headings from the table (we need to exclude the first one). In fact, all it needs is one line: `headers <- colnames(yourtable[-1])`

This grabs all the column names apart from the first one.

```
stot_all <- function(yourtable){
  #create empty vector, which will be filled and returned
  col1 <- c()
  headers <- colnames(yourtable[-1])
  colnum <- length(headers)
  #first loop
  for (i in c(1:colnum)){
    #second loop
    for (a in c(1:colnum)) {
      col1 <- c(col1, headers[a])
    }
  }
  col2 <- c()
    #first loop
  for (i in c(1:colnum)){
    #second loop
    for (a in c(1:colnum)) {
      col2 <- c(col2, headers[i])
    }
  }
  #convert to matrix
  yourmatrix <- as.matrix(yourtable[,-1])
  #create an array
  col3 <- array(yourmatrix)
  newtable <- data.frame(col1, col2, col3)
  return(newtable)
}
```

Now to test

```
stotalltest <- stot_all(regions)
```

Rewrite using better practice outlined here (https://stackoverflow.com/questions/22235809/append-value-to-empty-vector-in-r)