Melting & Casting

Contents

Note: I wrote this tutorial before tidyverse was a thing. I almost exclusively use those packages now for shaping my data, but the explanations here might still be useful.

Why Reshape Your Data

Reshape2 is a package that allows us to easily transform our data into whatever structure we may need. Many of us are used to seeing our data structured so that corresponds to a single participant and each column corresponds to a variable. This type of data structure is known as **wide format**. However, many of the packages in R require that we stretch our data so that a single participant may occupy multiple rows. This type of data structure is known as **long format**. For example, `ggplot2` and some data analysis functions require **long format**. Any of you who have tried to restructure their data using Excel or SPSS will immediately recognize the immense power of this package. Therefore, without further ado, let's get to it.

As a quick note, this tutorial will be heavily based around learning from examples. Therefore, I strongly encourage you to follow along with the example data and code provided below. Before we can begin our demonstration, it will be helpful to clear our environment, set our working directory, load the necessary packages, and take a quick glance at the description of melting and casting as described by the wondrous RStudio helper window.

So please, join me on this magical journey by running the code below.

```
#First, remove all your stuff
rm(list=ls())

#Make sure to set your working directory
#setwd("PATH")

#Next, you need to download and install the reshape package
#install.packages("reshape2")
library(reshape2)
```

R Help

Running the following code will give you the description of the reshape2 package

```
?melt
#or
?cast
```

Data Frame

Great job! Now, we need a data frame to play with. The code below will create one for you with two within-subjects factors and one between subjects factor.

```
#Creating a toy data set to melt
## Run everything below TOGETHER
ID<-c(1:6)                                  #Creating an ID variable for 6 participants
set.seed(66)                                #Setting a seed so we can all have the same values
WTN1<-runif(6, min = 1, max = 7)            #Creating a within-Subjects variable
set.seed(16)                                #Setting a seed so we can all have the same values
WTN2<-runif(6, min = 1, max = 7)            #Creating a within-Subjects variable
BTW<- replicate(3,1:2, simplify = T)        #Creating a between subjects variable
BTW<-as.vector(BTW)                         #Turning between-Subjects variable into a vector
mind<-cbind.data.frame(ID, WTN1, WTN2, BTW) #Combining variables into a dataframe
#mind$BTW<-as.factor(mind$BTW)              #Turning between-subjects variable into a factor
#mind$ID<-as.factor(mind$I)                 #Turning ID into a factor

mind                                        #Let's view this data set
```

```
##   ID     WTN1     WTN2 BTW
## 1  1 6.939619 5.098660   1
## 2  2 5.978156 2.464705   2
## 3  3 4.515321 3.700668   1
## 4  4 3.502277 2.376611   2
## 5  5 4.972581 6.181047   1
## 6  6 3.275813 2.867202   2
```

```
str(mind)
```

```
## 'data.frame':    6 obs. of  4 variables:
##  $ ID  : int  1 2 3 4 5 6
##  $ WTN1: num  6.94 5.98 4.52 3.5 4.97 ...
##  $ WTN2: num  5.1 2.46 3.7 2.38 6.18 ...
##  $ BTW : int  1 2 1 2 1 2
```

Melting

Below is the generic code for `melt`ing.

```
#This is commented out so that the generic code does not run
#melt(data, ..., na.rm = FALSE, value.name = "value")
```

Ok, now that we have a dataframe, let's `melt` the sucker!

```
#Melt your mind!
melt.mind<-melt(mind)

head(melt.mind)                                #Look at this!
```

```
##   variable value
## 1       ID     1
## 2       ID     2
## 3       ID     3
## 4       ID     4
## 5       ID     5
## 6       ID     6
```

```
tail(melt.mind)                                #Look at that!
```

```
##    variable value
## 19      BTW     1
## 20      BTW     2
## 21      BTW     1
## 22      BTW     2
## 23      BTW     1
## 24      BTW     2
```

Explanation

So what just happened?

The melt function took our data frame that had a column for each variable, and created a data frame with only **TWO** columns: One column named `variable` and one column named `value`.

As psychologists, we are often used to seeing things in **wide format** because SPSS defaults to **wide format**.

Wide format has a column for each variable, and every row is an instance of the variable **participant**, that is, each row represents one **participant**.

However, it is often useful, even occasionally necessary, to stretch out the data frame so that each row is an instance of a different variable. But what does that mean?

When we `melt`ed the data frame `mind`, we told R that we wanted each row to be a single instance of a value. In order to do that, we needed to collapse across all other variables into the new variables: `variable` and `value`. We stretched out our dataset so that it was **longer**, or into **long format**.

In **long format**, each row no longer represents a single participant. In our example, each participant was stretched into four rows, one row for each variable: one **between-subjects** variable, one `ID` variable, and two **within-subjects** variables.

But what if we wanted to maintain some other columns? For example, what if we wanted `ID` and our **between-subjects** variable `BTW` to be, well, **between-subjects**?

```
melt.mind2<-melt(mind, id=c("ID","BTW"))

melt.mind2                                     #Look at this!
```

```
##    ID BTW variable    value
## 1   1   1     WTN1 6.939619
## 2   2   2     WTN1 5.978156
## 3   3   1     WTN1 4.515321
```

```
## 4   4   2     WTN1 3.502277
## 5   5   1     WTN1 4.972581
## 6   6   2     WTN1 3.275813
## 7   1   1     WTN2 5.098660
## 8   2   2     WTN2 2.464705
## 9   3   1     WTN2 3.700668
## 10  4   2     WTN2 2.376611
## 11  5   1     WTN2 6.181047
## 12  6   2     WTN2 2.867202
```

In the `melt` function, you can specify your `ID`, or **between-subjects** variables, as in the previous line of code. R is smart and will assume all other variables are to be collapsed into each other.

By specifying that `ID` and `BTW` were **between-subjects** variables, we told R that we wanted our dataset structured so that each row is an instance of one of the **within-subjects** variables. This means that we now have two rows per subject because each subject experienced both of the levels of the **within-subjects** variable. Neat, right?

Now, what if we wanted to name our **within-subjects** variable to something other than `variable`? Try this...

```
melt.mind3<-melt(mind,id=c("ID","BTW"),
                 variable.name = "WTN")

melt.mind3                                    #Look at that!
```

```
##    ID BTW  WTN    value
## 1   1   1 WTN1 6.939619
## 2   2   2 WTN1 5.978156
## 3   3   1 WTN1 4.515321
## 4   4   2 WTN1 3.502277
## 5   5   1 WTN1 4.972581
## 6   6   2 WTN1 3.275813
## 7   1   1 WTN2 5.098660
## 8   2   2 WTN2 2.464705
## 9   3   1 WTN2 3.700668
## 10  4   2 WTN2 2.376611
## 11  5   1 WTN2 6.181047
## 12  6   2 WTN2 2.867202
```

As you can see, all we needed to do was specify that we would name our variable with the command `variable.name`. Easy-peasy!

And if we also wanted to name our `values`?

```
melted.mind<-melt(mind,id=c("ID","BTW"),
                  variable.name = "WTN",
                  value.name = "Results")

head(melted.mind)                             #Look at this!
```

```
##   ID BTW  WTN  Results
## 1  1   1 WTN1 6.939619
## 2  2   2 WTN1 5.978156
## 3  3   1 WTN1 4.515321
## 4  4   2 WTN1 3.502277
## 5  5   1 WTN1 4.972581
## 6  6   2 WTN1 3.275813
```

```
tail(melted.mind)                          #Look at that!
```

```
##    ID BTW  WTN  Results
## 7   1   1 WTN2 5.098660
## 8   2   2 WTN2 2.464705
## 9   3   1 WTN2 3.700668
## 10  4   2 WTN2 2.376611
## 11  5   1 WTN2 6.181047
## 12  6   2 WTN2 2.867202
```

Cool cool cool. We've successfully `melt`ed our `mind` in a way that we'd like. Our data is structured so that each row represents an instance of the **within-subjects** variable, `WTN`, and we've maintained the variables `ID` and `BTW`. Now, let's continue this magical journey onto the wondrous land of `cast`ing.

Casting

`Cast`ing will transform **long format** back into **wide format**. This will, essentially, make your data look as it did in the beginning (or in any other way you'd prefer).

There are multiple `cast` functions depending on the structures of your data. If you want to `cast` your data into a data frame, use `dcast`, and if you want to `cast` your data into vector/matrix/array, then use `acast`.

Because we will be working with a data frame, we will use `dcast`. The generic code for both types is below.

```
#These have been commented out so that they do not run.

#dcast(data, formula, fun.aggregate = NULL, ..., margins = NULL,
#  subset = NULL, fill = NULL, drop = TRUE,
#  value.var = guess_value(data))

#acast(data, formula, fun.aggregate = NULL, ..., margins = NULL,
#  subset = NULL, fill = NULL, drop = TRUE,
#  value.var = guess_value(data))
```

Before we begin, it's important to note that `cast`ing is much more challenging than `melt`ing. This may often take some trial and error, and you should not feel bad about that. Just remember: **You're awesome**. Feeling good about yourself? Good. *Good.*

Now, let's `cast` our data

```
cast.mind<-dcast(melted.mind, ID+BTW~WTN)

cast.mind                                  #Look at this!
```

```
##   ID BTW     WTN1     WTN2
## 1  1   1 6.939619 5.098660
## 2  2   2 5.978156 2.464705
## 3  3   1 4.515321 3.700668
## 4  4   2 3.502277 2.376611
## 5  5   1 4.972581 6.181047
## 6  6   2 3.275813 2.867202
```

Explanation

Al-righty then. First, we needed to specify the data frame we would be using. Here, we used the `melted.mind` data frame.

Next, we put in our `cast`ing formula. Now, R is pretty smart, and it assumes that the order in which you put the variables is meaningful. The description will tell you that whichever variable you put in first will be the *"slowest varying"* variable. In our case, the slowest varying is actually the **between-subjects** variable `BTW` because there are only 2 levels. In other words, our `BTW` variable will vary only once. However, if we put that in first, then the `ID` numbers would be out of order. Like this...

```
cast.mind2<-dcast(melted.mind, BTW+ID~WTN)

cast.mind2                                    #Look at that!
```

```
##   BTW ID     WTN1     WTN2
## 1   1  1 6.939619 5.098660
## 2   1  3 4.515321 3.700668
## 3   1  5 4.972581 6.181047
## 4   2  2 5.978156 2.464705
## 5   2  4 3.502277 2.376611
## 6   2  6 3.275813 2.867202
```

We can also choose to specify only one side of the `cast`ing formula, like this...

```
cast.mind3<-dcast(melted.mind, ID+BTW~...)

cast.mind3                                    #Look at this!
```

```
##   ID BTW     WTN1     WTN2
## 1  1   1 6.939619 5.098660
## 2  2   2 5.978156 2.464705
## 3  3   1 4.515321 3.700668
## 4  4   2 3.502277 2.376611
## 5  5   1 4.972581 6.181047
## 6  6   2 3.275813 2.867202
```

...or this...

```
cast.mind4<-dcast(melted.mind, ...~WTN)

cast.mind4                                    #Look at that!
```

```
##   ID BTW     WTN1     WTN2
## 1  1   1 6.939619 5.098660
## 2  2   2 5.978156 2.464705
## 3  3   1 4.515321 3.700668
## 4  4   2 3.502277 2.376611
## 5  5   1 4.972581 6.181047
## 6  6   2 3.275813 2.867202
```

Now, we could have started from the original dataset `melt.mind`, but we will need to do something extra to recover something close to the original **wide** data. How about you run the code below, and I'll walk you through what you see? Sound good? Ok, go for it. I'll wait.

```
cast.mind5<-dcast(melt.mind, ID+BTW+WTN1+WTN2~"Row")

cast.mind5                                    #Look at this!
```

```
##   ID BTW     WTN1     WTN2 Row
```

```
## 1  1   1 6.939619 5.098660   1
## 2  2   2 5.978156 2.464705   2
## 3  3   1 4.515321 3.700668   3
## 4  4   2 3.502277 2.376611   4
## 5  5   1 4.972581 6.181047   5
## 6  6   2 3.275813 2.867202   6
```

As you can see, we have a new column named "`Row`". Why did I do that? In our original dataset `mind`, I did not specify that `ID` and `BTW` were factors. If you'll scroll back up to the section where I called the structure of the `mind` data, the variable types for `ID` and `BTW` are `int`. This is why the two variables get folded into each other in the `melt.mind` data. If you'll go back *even farther* to the code where we were creating our data frame, I commented out two lines that would have converted `ID` and `BTW` into factors. If you run that code prior to creating the `melt.mind` data, then it will look exactly like the `melt.mind2` data. Don't believe me? You can try it, if you'd like. If you do try it, however, you should probably rename the datasets you create something else so you can keep everything straight.

Now, what happens if you forget to include a variable in your casting formula? Well, that all depends on which variable you forget. If you forget to include your between-subjects, `BTW` variable, then it may disappear from your casted data. Like this...

```
error.mind<-dcast(melted.mind, ID~WTN)

error.mind                                    #Look at that!
```

```
##   ID     WTN1     WTN2
## 1  1 6.939619 5.098660
## 2  2 5.978156 2.464705
## 3  3 4.515321 3.700668
## 4  4 3.502277 2.376611
## 5  5 4.972581 6.181047
## 6  6 3.275813 2.867202
```

Notice how the `BTW` variable just disappeared? That is something you should be aware of and keep an eye out for. Personally, I always double-check my work after `melt`ing and `cast`ing.

What happens if you forget the subject `ID` variable? Well, this will result in a very different result. Check it out...

```
error.mind2<-dcast(melted.mind, BTW~WTN)

error.mind2                                    #Look at this!
```

```
##   BTW WTN1 WTN2
## 1   1    3    3
## 2   2    3    3
```

Notice the error message `Aggregation function missing: defaulting to length`? Notice that there are now only 2 rows, one for each level of your `BTW` variable, and the values in under each `WTN` level is `3`? That's because `defaulting to length` appears to have meant that `R` will collapse all values that you used to have into just the number of columns in your dataset. Now, maybe you want to collapse across all participants. Hey, it's possible. But what if instead of a worthless number, like the number of columns in you data, you wanted the average for each within-subjects variable at each between-subjects variable? Now that sounds like some info that could be useful!! To do this, we will need to use the `fun.aggregate` function in the `cast`ing formula. Like this...

```
mind.summary<-dcast(melted.mind, BTW~WTN, fun.aggregate = mean)

mind.summary                                    #Look at that!
```

```
##   BTW     WTN1     WTN2
## 1   1 5.475841 4.993459
## 2   2 4.252082 2.569506
```

You could have actually included any function after the `fun.aggregate` function, including a local function. I chose to demonstrate the `mean` function simply because it was easy and may be useful to you in the future.

Well, that's it. Read on for more useful links and information.

Final Words

This is it for my introduction to the `reshape2` package, but there are a *ton* of things you can do with `cast` that I did not get around to describing. I recommend that you play around with this package and figure out what works best for you. And, if you need any other help, or if my explanations were too ridiculous for your serious `mind`, then you can go to the sources below for additional help...

Sources

http://seananderson.ca/2013/10/19/reshape.html
This tutorial is amazing

https://cran.r-project.org/web/packages/reshape2/reshape2.pdf
This one is pretty good too

http://had.co.nz/reshape/
This is the `reshape2` website