Blueprint. A tool to auto-merge datasets by rules specified in a meta-file

Marc Schwenzer

October 20, 2016

Contents

1 Overview

Modern data users face the need to combine very different data sources which can easily become confusing. The R-package blueprint offers the opportunity to select, combine and transform data in an easy and intuitive way by using a meta-data file (a "blueprint") that can be easily edited with a chosen Spreadsheet application or text editor capable of e.g. saving the file formats OpenOffice Calc, Excel, CSV, HTML. Merging variables that

are split over dozens of different files is done by entering the absolute necessary information into a text-based meta-file and let the package do the rest. Blueprints define which variables from files are imported and (left) joins variables from other files. Blueprint is also best suited for the needs of joining longitudinal data and unifying variable names that changed across different waves into one new variable. By relying on the versatile rio-package it offers the opportunity to handle very different kind of data formats (for the imported data as well as the blueprint-files.). Every variable can be transformed by functions or a chain of functions (operations seqentially applied to the original variable from the corresponding file *before* it is merged into the final data frame). Repetitive recoding of similar variables becomes very easy. By relying on meta-files data-processing becomes more versatile, clearer and more easy expandable. The workhorse-functions that merge and transform data in the background rely mainly on functions from the brilliant and comparable fast packages dplyr and data.table (CITE).

2 Merge same cases from different files

In blueprint the merge process is specified by a so-called blueprint file into which we enter data in a standard Spreadsheet-Application like Excel or Open Office Spreadsheet. To enter data, we first have to create a new blueprint template. This can be done easily by entering:

`open.blue('my.blueprint.name.csv')`
Alternatively we could change the suffix of the filename to xlsx to open an Excel-File:
`open.blue('my.blueprint.name.xlsx)`

Every row represents an unique variable of a final data.frame. The name of this variable can be chosen according to the needs of the analysis that is specified in column one ('newvar'). Assume we have 2 files 'individual.csv' and 'household.csv' and want to import the variables '*idno*, '*gender*' and '*birthY*' from file 'individual.csv'. This original variable names have to be written into the column var1 like in figure X.

newvar	var1	file1	link1	fun1
individual.id	idno	individual.csv		
gender	gend	individual.csv		
birth.year	birthY	individual.csv		
household.id	houseID	individual.csv		
###				
household.income	income	household.csv	household.id=idno	
household.property	property	household.csv	household.id=idno	

Adding the additional variable '*household income*' and '*property*' from the seperate file 'household.csv' typically includes left joining them, which equals to adding columns for additional variables and keeping only the values of household.csv of those units that are also in individual.csv. For this we have to define conditions which rows in the files individual.csv and household.csv relate to the same unit. This is achieved by specifiying the column link for the files to add from. Relying on the syntax of $dplyr::left_{join}$ (while leaving out apostrophes) the condition expresses which variables must be equal: You give the new variable name (specified in newvar) and the name of the matching old variable name in the original data file that is to be added: *name.in.newvar=old.variable.name.in.data.file.to.merge*

Assuming e.g. that in file 'household.csv' every household has an id number called '*idno*' we would specify the link condition *household.id=idno*. Several link conditions can be combined by comma. The first part relates to new variable name specified as '*newvar*', the second to the original name in the data file to merge. If only single variables are given, it is assumed that the variable names are the same (name of newvar and var in a file to add are identical).
If the variables used inside of the link condition are not in the *var1...* column they will be used for merging but dropped afterwards.

When a blueprint has been modified and saved in a external application, the merged data can be imported into R by simply stating:

```
my.df <- blue('my.blueprint.name.csv')
```

my.df is a new data.frame with the selected variables from individual.csv plus the household variables that match the link condition.

3 Transforming variables by functions

By now the column fun1 has been empty. This column offers a convenient way to transform variables (e.g. for recoding categories). The specified functions are executed with the original variable which will be automatically replaced by the result of the function. Let's say we want to recode gender from *0* to '*Male*' and from *1* to '*Female*'. You could do this by using recode from the dplyr package like shown in figure XXX (leaving out the first argument x (variable name) which the original variable will be inserted automatically). The result of this blueprint will be a data.frame with gender containing the values '*Male*' and '*Female*'. Note that a fun entry can be used to execute a whole chain of transformations (seperated by '%>%'). Have a look at the documentation on pipes provided by *magrittr* how to use pipes *?magrittr::%>%*. Also note that e.g. recoding different variables with the same coding scheme can be done by copy and paste of fields of fun1 to other rows.

4 Merging different waves

Repeated surveys typically are delivered as so-called waves. Blueprint makes it easy to merge different waves into a long format. Independent of whether the same units are measured repeatedly or the same variables are measured repeatedly for different units, the purpose will be to combine data *rowwise*. Data providers (like e.g. OECD PISA) sometimes emit wave-files with changed variable names for the same items. Harmonizing this variable is very easy with using blueprint. The *newvar* column contains the variable names. Additional variable columns starting with var (e.g. *var1*, *var2*,...) contain the

variable names from the files specific to the wave. We can join two waves by entering additional columns containing the identifier var,file,link,fun. Or we can initialize a new blueprint with the waves argument using:
`open.blue('my.blueprint.name2.xlsx,waves=1:2)`

Figure XXX shows a more advandced blueprint file reflecting this structure.

newvar	var1	file1	link1	fun1	var2	file2	link2	fun2
i.id	idno	i.w1.dta			ID	i.w2.Rdata		
gender	gend	i.w1.dta			gend	i.w2.Rdata		
birth.year	birthY	i.w1.dta			birth	i.w2.Rdata		
hh.id	houseID	i.w1.dta			houseID	iw.2.Rdata		
###								
hh.income	income	h.w2.dta	hh.id=idno			h.w2.csv		hh.id=idno
hh.property	property	h.w2.dta	hh.id=idno			h.w2.csv		hh.id=idno

Entering the appropriate data will rename, transform and join the data automatically. In short: Waves are specified *columnwise* (with blocks of 4 columns for each wave containing the original variable name, filepath, link conditions, and transformation functions). Columns that relate to units in the same wave are specified *rowwise* by giving different names and setting the link condition.

5 Source code

Blueprint by default creates a corresponding .R code file containing all operations necessary to merge the data, by default written to $filepath_{withoutextension}.blueprint.code.R$. If you choose to do so you can use, edit and share this standalone file without the need to have the blueprint package installed.

6 Logging and descriptives of the merging / transfomation process

Blueprint is constructed to not be very verbose when called. Nonetheless it has a logging feature that can be activated by setting *blue(. . . ,logfile=TRUE)*. In this case a extended logfile is created that contains also information on the transformation process (recode table, descriptives, distribution and information about automatic type conversions, statistics on dimension of the data). The parameter *logfile* is set to *FALSE* by default, only standard information will be written to this file. If you set logfile to a character string (by e.g. *blue(. . . ,logfile='a.file.txt')*), an extended logfile will be written to this path. If you don't specify a path, the name will be resembled by the name of the blueprint file The computation of the statistics take some time and therefore there is a tradeoff

between time and the comfort of additional information.

```
----Transformation. Variable ‘ST03Q01‘  (wave 1): recode(‘2‘=0L,‘1‘=1L,.default=NA_integer_)  ----------------------------

 ==============================
 old    1      2     7   8   9
 ..     |      |     |   |   |
 ...    v      v     v   v   v
 new    1      0
 X.n. 115030 112128 1055 15 556
 ------------------------------
!!! Type conversion from numeric to integer. Was this intended?


>>> Distribution after recoding -----
variable
 n missing  unique    Info     Sum    Mean
 227158    1626       2    0.75  115030  0.5064
```

7 Assigning fixed values

It might be convenient to create new variables that are constant for every unit of the same wave. This can be done by entering names for var1 , var2,... that are not in the original data file. To assign fixed value to a new variable you either use the integer specification or encapsulating characters into apostrophe (') . Note that since Excel has a special treatment of captioning characters using two beginning Apostrophes and one ending apostrophe probably will have to be used (”character value') .

newvar	var1	file1	link1
i.id	idno	i.w1.dta	
survey.year	2000L	i.w1.dta	
wavec	'PISA2000'	i.w1.dta	

Note the difference between e.g. *i.id* stemming from a column in the file i.w1.dta and *survey.year* which will be the same ("PISA2000") for all units in file i.w1.dta.

8 Selection of multiple variables

Assume you have 80 weight variables specified by rep.weight1 to rep.weight80. You can specify these in var1 as *rep.weight[1:80]*. The rows containing brackets will be expanded to 80 additional rows with the specific name, resulting in the import (and if specified also individual transformation) of all of this variables.

9 File formats for importing and exporting.

When importing files the package *blueprint* relies on the function *import* from the package *rio*. This package recognises the most frequently used file formats based on their

corresponding suffix as listed in Figure XXX. This applies to loading blueprint files as well as data file specified in this data files.

Suffix	Assumed file format
R binary file	.Rdata
Stata files	.rda
SPSS files	.sav
Comma seperated Text files	.csv
HTML files	.html
Excel files	.xlsx
Open Spreadsheet files	.ods

Note that additional arguments of blue are postponed to the function import resulting in additional specifications for import process. If you e.g. want to import the specific spreadsheet "MyData" in a Excel file (default would be the first Spreadsheet) it can be selected by specifying a *which* argument.
blue('/path/to/blueprint.file.xlx',which='MyData')

10 Exporting files

By giving the argument *export.file*, e.g. blue(..., export.file='*path/to/file.csv') the merged data.frame will be written directly to the file specified. In this case the data.frame will be returned /invisible* - it can be used in other functions or pipes, but will not be printed and automatically deleted frome memory if not assigned to a new specifier.

11 Aggregating data

Will follow.