Shaping of Data For Analysis Using Manipulation Functions DCAST() And MELT() In R

Ramaraju.S.V.S.V.Palla
[1, 2] Dept of CSE
[1, 2] DIET Engineering College, Visakhapatnam

Abstract- *In extracting the useful results from the huge date we need to perform the data analysis in which the organizing of data in need to be structured. We need to prepare the data to be used by the various data science projects without spending time on examining the data. So in this paper I reviewed the two manipulation functions DCAST() and MELT() for shaping the data into a form which is appropriate for analysis. The manipulation function DCAST() is capable of handling large data, guessing of result , filling the values with missing cells, provides informative messages through the console in more efficient manner with respective to the memory utilization. The function MELT() is capable of handling variable positions, or name, used to store measured variable names, NA values are converted form explicit missing to implicit missing. We also detect the resulting issues, and address them using analytical programming language R*

Keywords- data, manip, dcast, melt, statistical, visualization, shaping.

I. INTRODUCTION

Cleaning data and preparing data is an important characteristic for the time and effort which is normally spent in a data science project. It is better if we get into the modeling step directly without spending the time at the data set when we have a lot of data. But as we know that no data set is perfect; we may have incorrect data, we may have misinterpreted data, we may have missing data, among some of the data fields may be inconsistent and some data fields may be dirty. So, if we don't spend time on examining the data before to start the model, we may find our self redoing our work repeatedly as we determines bad data variables or fields which required to be transformed before modeling. In the worst case, we will build a model that generates incorrect predictions.

Therefore by handling such kind of data issues before, we can save our self some unnecessary redoing of same work, and we can save lot of time. In this paper, we'll clearly show the existence or truth of some of the things that can go wrong with data, and examine different approaches or methods to address those issues using the statistical language R before going on to analysis. For quick numerical libraries, we will use the Microsoft R Open distribution

Several issues crop up again and again when preparing data for analysis in R:

- Loading of data from databases, spreadsheets, or other formats into R
- Shaping data into a form which is appropriate for analysis
- Checking variable types
- Managing bad values:
- Dealing with missing values (NA)
- Anticipating future novel categorical values
- Re-encoding categorical values with too many levels

Among the several issues we focus on the shaping data into a form which is appropriate for analysis by using the data manipulation functions.

Analysts tend to follow 4 fundamental processes to turn data into understanding, knowledge & insight:

1. Data manipulation
2. Data visualization
3. Statistical analysis/modeling
4. Deployment of results

1.1 Data Manipulation : It is often said that 80% of data analysis is spent on the process of cleaning and preparing the data. Well structured data serves two purposes:

1. Makes data suitable for software processing whether that be mathematical functions, visualization, etc.
2. Reveals information and insights

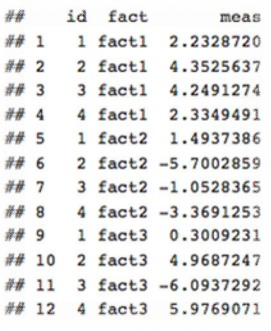
II. LOADING DATA

There are a various packages and functions in R to load data into it. Here, we represent a few of our preferred packages and functions for loading data from the most common data sources.

Data source	**Package**	**Function**	**Comments**
Fixed-field text file (no delimiters)	utils	read.fwf()	Assumes fields are in fixed position on each line.
Delimited text file (comma-separated, tab-separated, etc.)	utils	read.table() reads data file into a data frame	Assumes data is in a regular, tabular format (no ragged rows).
Database	RJDBC	dbConnect() to establish connection dbReadTable() to read entire table into data frame dbGetQuery() to access data via SQL	Requires Java VM and JDBC driver. Supplies a good, concrete interface to many databases.
Excel spreadsheets	gdata	read.xls()	Requires perl. There are other packages, but gdata is the most reliable in our experience.

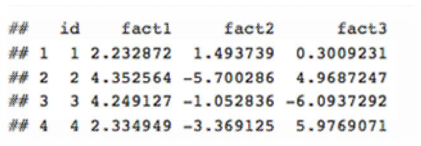
III. SHAPING DATA

The “shape” that is most efficient for storing or recording data information is not always the best shape for analyzing the data. For example, log data generally comes in a long and skinny format where information about a single entity (for example, a single customer or a single machine) is scattered across many rows.



```
##     id  fact       meas
## 1    1 fact1  2.2328720
## 2    2 fact1  4.3525637
## 3    3 fact1  4.2491274
## 4    4 fact1  2.3349491
## 5    1 fact2  1.4937386
## 6    2 fact2 -5.7002859
## 7    3 fact2 -1.0528365
## 8    4 fact2 -3.3691253
## 9    1 fact3  0.3009231
## 10   2 fact3  4.9687247
## 11   3 fact3 -6.0937292
## 12   4 fact3  5.9769071
```

For analysis, we generally prefer data in a wide format, where all facts about a single entity are stored in a single row. As shown below



```
##   id    fact1     fact2      fact3
## 1  1 2.232872  1.493739  0.3009231
## 2  2 4.352564 -5.700286  4.9687247
## 3  3 4.249127 -1.052836 -6.0937292
## 4  4 2.334949 -3.369125  5.9769071
```

This wide format is so central to R that R calls rows as observations and columns as variables.

R provides several functions for converting data from skinny to wide formats, and vice versa.

- To convert from skinny to wide data, use dcast().
- To convert in the opposite direction, use melt().
- More advanced users may wish to move on to the dplyr and tidyr ecosystems, though this involves learning additional notation.

3.1 Dcast()

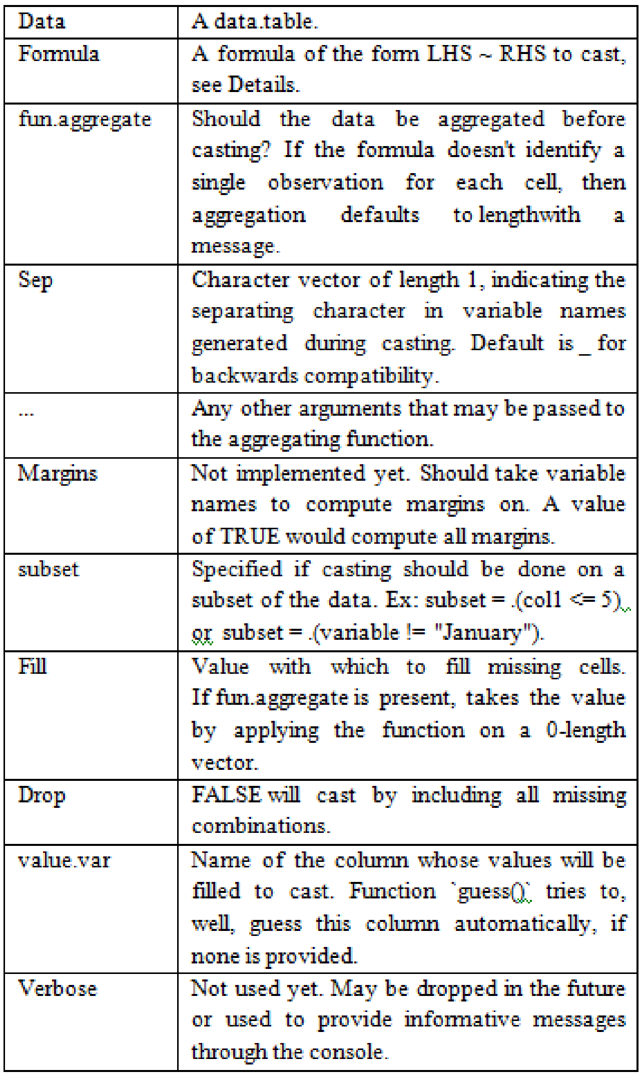
Fast Dcast For Data.Table, dcast.data.table is a much faster version of reshape2::dcast, but for data.tables. More importantly, it's capable of handling very large data quite efficiently in terms of memory usage in comparison to reshape2::dcast. From 1.9.6, dcast is implemented as an S3 generic in data.table. To melt or cast data.tables, it is not necessary to load reshape2 anymore. If you have load reshape2, do so before loading data.table to prevent unwanted masking.

NEW: dcast.data.table can now cast multiple value.var columns and also accepts multiple functions to fun.aggregate. See Examples for more.

Usage

# S3 method for data.table dcast (data, formula, fun.aggregate = NULL, sep = "_", ..., margins = NULL, subset = NULL, fill = NULL, drop = TRUE, value.var = guess(data), verbose = getOption("datatable.verbose"))

Arguments



Data	A data.table.
Formula	A formula of the form LHS ~ RHS to cast, see Details.
fun.aggregate	Should the data be aggregated before casting? If the formula doesn't identify a single observation for each cell, then aggregation defaults to lengthwith a message.
Sep	Character vector of length 1, indicating the separating character in variable names generated during casting. Default is _ for backwards compatibility.
...	Any other arguments that may be passed to the aggregating function.
Margins	Not implemented yet. Should take variable names to compute margins on. A value of TRUE would compute all margins.
subset	Specified if casting should be done on a subset of the data. Ex: subset = .(col1 <= 5), or subset = .(variable != "January").
Fill	Value with which to fill missing cells. If fun.aggregate is present, takes the value by applying the function on a 0-length vector.
Drop	FALSE will cast by including all missing combinations.
value.var	Name of the column whose values will be filled to cast. Function `guess()` tries to, well, guess this column automatically, if none is provided.
Verbose	Not used yet. May be dropped in the future or used to provide informative messages through the console.

Details

The cast formula takes the form LHS ~ RHS, ex: var1 + var2 ~ var3. The order of entries in the formula is essential. There are two special variables: . and .... .represents no variable; ... represents all variables not otherwise mentioned in formula; dcast also allows value.var columns of type list.

When variable combinations in formula doesn't identify a unique value in a cell, fun.aggregate will have to be specified, which defaults to length if unspecified. The aggregating function should take a vector as input and return a single value (or a list of length one) as output. In cases where value.var is a list, the function should be able to handle a list input and provide a single value or list of length one as output.

If the formula's LHS contains the same column more than once, ex: dcast(DT, x+x~ y), then the answer will have duplicate names. In those cases, the duplicate names are renamed using make.unique so that key can be set without issues.

Names for columns that are being cast are generated in same order (separated by underscore, _) from the (unique) values in each column mentioned in the formula RHS. From v1.9.4, dcast tries to preserve attributes whereever possible.

NEW: From v1.9.6, it is possible to cast multiple value.var columns and also cast by providing multiple fun.aggregate functions.

Multiple fun.aggregate functions should be provided as a list, for e.g., list(mean, sum, function(x) paste(x, collapse=""). value.var can be either a character vector or list of length=1, or a list of length equal to length(fun.aggregate). When value.var is a character vector or a list of length 1, each function mentioned under fun.aggregate is applied to every column specified under value.var column. When value.var is a list of length equal to length(fun.aggregate) each element of fun.aggregate is appled to each element of value.var column.
Value

A keyed data.table that has been cast. The key columns are equal to the variables in the formula LHS in the same order.

Examples

```
# NOT RUN {
require(data.table)
names(ChickWeight) <- tolower(names(ChickWeight))
DT <- melt(as.data.table(ChickWeight), id=2:4) # calls melt.data.table

# dcast is a S3 method in data.table from v1.9.6
dcast(DT, time ~ variable, fun=mean)
dcast(DT, diet ~ variable, fun=mean)
dcast(DT, diet+chick ~ time, drop=FALSE)
dcast(DT, diet+chick ~ time, drop=FALSE, fill=0)

# using subset
dcast(DT, chick ~ time, fun=mean, subset=.(time < 10 & chick < 20))

# drop argument, #1512
DT <- data.table(v1 = c(1.1, 1.1, 1.1, 2.2, 2.2, 2.2),
        v2 = factor(c(1L, 1L, 1L, 3L, 3L, 3L), levels=1:3),
        v3 = factor(c(2L, 3L, 5L, 1L, 2L, 6L), levels=1:6),
        v4 = c(3L, 2L, 2L, 5L, 4L, 3L))

# drop=TRUE
dcast(DT, v1 + v2 ~ v3)                  # default is
drop=TRUE
dcast(DT, v1 + v2 ~ v3, drop=FALSE) # all missing
combinations of both LHS and RHS
dcast(DT, v1 + v2 ~ v3, drop=c(FALSE, TRUE)) # all
missing combinations of only LHS
dcast(DT, v1 + v2 ~ v3, drop=c(TRUE, FALSE)) # all
missing combinations of only RHS

# using . and ...
DT <- data.table(v1 = rep(1:2, each = 6),
        v2 = rep(rep(1:3, 2), each = 2),
        v3 = rep(1:2, 6),
        v4 = rnorm(6))
dcast(DT, ... ~ v3, value.var = "v4") #same as v1 + v2 ~ v3,
value.var = "v4"
dcast(DT, v1 + v2 + v3 ~ ., value.var = "v4")

## for each combination of (v1, v2), add up all values of v4
dcast(DT, v1 + v2 ~ ., value.var = "v4", fun.aggregate =
sum)

# }


# NOT RUN {

# benchmark against reshape2's dcast, minimum of 3 runs
set.seed(45)
DT <- data.table(aa=sample(1e4, 1e6, TRUE),
   bb=sample(1e3, 1e6, TRUE),
   cc = sample(letters, 1e6, TRUE), dd=runif(1e6))
system.time(dcast(DT, aa ~ cc, fun=sum)) # 0.12 seconds
system.time(dcast(DT, bb ~ cc, fun=mean)) # 0.04 seconds
# reshape2::dcast takes 31 seconds
system.time(dcast(DT, aa + bb ~ cc, fun=sum)) # 1.2
seconds
# }

# NOT RUN {

# NEW FEATURE - multiple value.var and multiple
fun.aggregate
dt = data.table(x=sample(5,20,TRUE),
y=sample(2,20,TRUE),
        z=sample(letters[1:2], 20,TRUE), d1 = runif(20),
d2=1L)

# multiple value.var
dcast(dt, x + y ~ z, fun=sum, value.var=c("d1","d2"))

# multiple fun.aggregate
dcast(dt, x + y ~ z, fun=list(sum, mean), value.var="d1")

# multiple fun.agg and value.var (all combinations)
dcast(dt, x + y ~ z, fun=list(sum, mean), value.var=c("d1",
"d2"))

# multiple fun.agg and value.var (one-to-one)
dcast(dt, x + y ~ z, fun=list(sum, mean),
value.var=list("d1", "d2"))
# }
```

Melt()

Melt A Data Frame Into Form Suitable For Easy Casting. You need to tell melt which of your variables are id variables, and which are measured variables. If you only supply one of id.vars and measure.vars, melt will assume the remainder of the variables in the data set belong to the other. If you supply neither, melt will assume factor and character variables are id variables, and all others are measured.

Usage

```
"melt"(data, id.vars, measure.vars,
variable.name = "variable", ..., na.rm = FALSE,
value.name = "value", factorsAsStrings = TRUE)
```

Arguments

data	data frame to melt
id.vars	vector of id variables. Can be integer (variable position) or string (variable name). If blank, will use all non-measured variables.
measure.vars	vector of measured variables. Can be integer (variable position) or string (variable name)If blank, will use all non id.vars
variable.name	name of variable used to store measured variable names
...	further arguments passed to or from other methods.
na.rm	Should NA values be removed from the data set? This will convert explicit missings to implicit missings.
value.name	name of variable used to store values
factorsAsString	Control whether factors are converted to character when melted as measure variables. When FALSE, coercion is forced if levels are not identical across the measure.vars.

Script.R

1. names(airquality) <- tolower(names(airquality))
2. melt(airquality, id=c("month", "day"))
3. names(ChickWeight)<- tolower (names (ChickWeight))
4. melt(ChickWeight, id=2:4)

IV. CONCLUSION

In this, we have discussed at a common pain points that arise while preparing our data for the purpose analysis. We also shown how to detect or predict these issues, and how to address them using R. Some of these tasks can be instantly automated, but some will be more domain specific and those issues must be handled on a case-by-case basis

REFERENCES

Papers:

[1] More over ANOVA ralitza guerguieva
[2] Preparing data for analysis using R Nina Zumel, Win-Vector LLC March 2016

Books:

[3] Applied spatial data analysis withR RS Bivand, V Gomex-Rubio
[4] Efficient R Programming Colin Gillespie