Data Reshaping and Merging

Arthur Allignol

arthur.allignol@uni-ulm.de

Table of Contents

Data Reshaping

An important operation in R

- Most R functions expect their input (usually data frames) to be arranged in particular ways
- It is the responsibility of the user to ensure that the data are in the appropriate form
- For instance, data for multiple groups are organised as columns, with a column for each group
- Most R functions expect values to be in **one** column with an additional column specifying the groups

Long versus Wide Format

Useful concept for, e.g., *longitudinal studies*, in which a patient may have several measurements over time

Wide If all the measurements for a single individual are in the same row, the data are said to be **wide**

```
id visit1 visit2
1      90     95
2      80     78
```

Long If each measurement is in a different row, the data are said to be in the **long** format

```
id visit measure
 1     1      90
 1     2      95
 2     1      80
 2     2      78
```

Most data sets are delivered in the wide format, modelling is done in the long format

The stack and unstack Functions

stack and unstack are simple functions (i.e., not very flexible) that permits to transform data in the wide format to long (stack) and vice versa (unstack)

```
data <- data.frame(group1 = rnorm(5, 10, 5),
                   group2 = rnorm(5, 10, 5),
                   group3 = rnorm(5, 10, 5))
data
```

```
##      group1    group2    group3
## 1  8.236471 16.158070 13.601809
## 2  7.347027 15.736366  9.069201
## 3  9.594164  7.017513  1.338025
## 4  5.673082  5.427388  3.025840
## 5 19.368087  9.980503  1.914245
```

The stack and unstack Functions

```
sdata <- stack(data)
head(sdata)
```

```
##      values    ind
## 1  8.236471 group1
## 2  7.347027 group1
## 3  9.594164 group1
## 4  5.673082 group1
## 5 19.368087 group1
## 6 16.158070 group2
```

The select option permits to select variables from the original data set

```
head(stack(data, select = -group1))
```

```
##      values    ind
## 1 16.158070 group2
## 2 15.736366 group2
## 3  7.017513 group2
## 4  5.427388 group2
## 5  9.980503 group2
## 6 13.601809 group3
```

The stack and unstack Functions

```
stack(data, select = c(group1, group3))
```

```
##        values    ind
## 1    8.236471 group1
## 2    7.347027 group1
## 3    9.594164 group1
## 4    5.673082 group1
## 5   19.368087 group1
## 6   13.601809 group3
## 7    9.069201 group3
## 8    1.338025 group3
## 9    3.025840 group3
## 10   1.914245 group3
```

The stack and unstack Functions

The unstack function does the inverse operation

```
unstack(sdata)
```

```
##       group1    group2    group3
## 1  8.236471 16.158070 13.601809
## 2  7.347027 15.736366  9.069201
## 3  9.594164  7.017513  1.338025
## 4  5.673082  5.427388  3.025840
## 5 19.368087  9.980503  1.914245
```

The form argument is a formula that specifies the vector to be unstacked (LHS) and the indicator of the groups to create (RHS)

```
unstack(sdata, form = values ~ ind)
```

```
##       group1    group2    group3
## 1  8.236471 16.158070 13.601809
## 2  7.347027 15.736366  9.069201
## 3  9.594164  7.017513  1.338025
## 4  5.673082  5.427388  3.025840
## 5 19.368087  9.980503  1.914245
```

The `reshape` Function

The `reshape` function performs the long $\rightarrow$ wide and wide $\rightarrow$ long transformations

- Motivated by longitudinal data (repeated measurements)
- Very flexible function (maybe too much)
- Google very useful for using this function

The reshape Function

Wide → Long Transformation

As an example, consider a data set on US personal expenditure

```
usp
```

```
##                       type  X1940  X1945 X1950 X1955 X1960
## 1    Food and Tobacco 22.200 44.500 59.60  73.2 86.80
## 2 Household Operation 10.500 15.500 29.00  36.5 46.20
## 3  Medical and Health  3.530  5.760  9.71  14.0 21.10
## 4       Personal Care  1.040  1.980  2.45   3.4  5.40
## 5   Private Education  0.341  0.974  1.80   2.6  3.64
```

The `reshape` Function

Wide → Long Transformation

Useful arguments for wide to long transformations

- `varying`: names of sets of variables in the wide format that correspond to single variables in long format. Can be a list of names (see later)
- `v.names`: The name we wish to give the variable containing these values in our long dataset
- `timevar`: The name we wish to give the variable describing the different times or metrics
- `times`: the values this variable will have
- `ids`: Values describing the different individuals
- `direction`: Character string indicating the direction of the transformation; either `"wide"` or `"long"`
- `times`, `split`, `sep`

The reshape Function

Wide → Long Transformation

Only specifying `varying` (as a list of variable names) and `direction` works

```
rr <- reshape(usp, varying = list(names(usp)[-1]), direction = "long")
head(rr)
```

```
##                     type time  X1940 id
## 1.1    Food and Tobacco    1 22.200  1
## 2.1 Household Operation    1 10.500  2
## 3.1  Medical and Health    1  3.530  3
## 4.1       Personal Care    1  1.040  4
## 5.1   Private Education    1  0.341  5
## 1.2    Food and Tobacco    2 44.500  1
```

But

```
reshape(usp, varying = names(usp)[-1], direction = "long")
```

```
## Error in guess(varying):  failed to guess time-varying variables
from their names
```

If `varying` is a vector of column names, `reshape` attempts to guess the `v.names` and `times` from these names

The reshape Function

Wide → Long Transformation

```
rr2 <- reshape(usp, varying = list(names(usp)[-1]), idvar = "type",
               times = seq(1940, 1960, 5), v.names = "expenditure",
               direction = "long")
head(rr2)
```

```
##                                          type time expenditure
## Food and Tobacco.1940       Food and Tobacco 1940      22.200
## Household Operation.1940 Household Operation 1940      10.500
## Medical and Health.1940   Medical and Health 1940       3.530
## Personal Care.1940             Personal Care 1940       1.040
## Private Education.1940     Private Education 1940       0.341
## Food and Tobacco.1945       Food and Tobacco 1945      44.500
```

The reshape Function

Wide → Long Transformation

```
rr3 <- reshape(usp, varying = names(usp)[-1], idvar = "type",
               times = seq(1940, 1960, 5), v.names = "expenditure",
               direction = "long")
head(rr3)
```

```
##                                          type time expenditure
## Food and Tobacco.1940       Food and Tobacco 1940      22.200
## Household Operation.1940 Household Operation 1940      10.500
## Medical and Health.1940   Medical and Health 1940       3.530
## Personal Care.1940             Personal Care 1940       1.040
## Private Education.1940     Private Education 1940       0.341
## Food and Tobacco.1945       Food and Tobacco 1945      44.500
```

The reshape Function

Wide → Long Transformation

```
rr3 <- reshape(usp, varying = names(usp)[-1], idvar = "type",
               times = seq(1940, 1960, 5), v.names = "expenditure",
               direction = "long")
head(rr3)
```

```
##                                          type time expenditure
## Food and Tobacco.1940       Food and Tobacco 1940      22.200
## Household Operation.1940 Household Operation 1940      10.500
## Medical and Health.1940   Medical and Health 1940       3.530
## Personal Care.1940             Personal Care 1940       1.040
## Private Education.1940     Private Education 1940       0.341
## Food and Tobacco.1945       Food and Tobacco 1945      44.500
```

Specifying a vector of names in `varying` now works because we also specify how the resulting variable should be named (`v.names`)

The reshape Function

Wide → Long Transformation

The split argument can be used to automatically determine the values for the times and names for the variables containing the values. It is a list with 3 components

- regexp: regular expression used to split the names used in varying
- include Logical that decides whether splitting occurs after the first character of the matched string
- Optionally fixed: Logical; Fixed-string matching

```
rr4 <- reshape(usp, varying = names(usp)[-1], idvar = "type",
               split = list(regexp = "X", include = TRUE),
               direction = "long")
head(rr4)
```

```
##                                          type time      X
## Food and Tobacco.1940       Food and Tobacco 1940 22.200
## Household Operation.1940 Household Operation 1940 10.500
## Medical and Health.1940   Medical and Health 1940  3.530
## Personal Care.1940             Personal Care 1940  1.040
## Private Education.1940     Private Education 1940  0.341
```

The reshape Function

Wide → Long Transformation

The sep argument is sometimes useful to help reshape automagically find the v.names

```
rr5 <- reshape(usp, varying = names(usp)[-1], idvar = "type",
               sep = "",
               direction = "long")
head(rr5)
```

```
##                                       type time      X
## Food and Tobacco.1940       Food and Tobacco 1940 22.200
## Household Operation.1940 Household Operation 1940 10.500
## Medical and Health.1940   Medical and Health 1940  3.530
## Personal Care.1940             Personal Care 1940  1.040
## Private Education.1940     Private Education 1940  0.341
## Food and Tobacco.1945       Food and Tobacco 1945 44.500
```

The reshape Function

Wide → Long Transformation

Reshape()'d data have additional attributes so that the inverse transformation is easy

```
reshape(rr2)
```

```
##                                          type  X1940  X1945 X1950 X1955
## Food and Tobacco.1940       Food and Tobacco 22.200 44.500 59.60  73.2
## Household Operation.1940 Household Operation 10.500 15.500 29.00  36.5
## Medical and Health.1940   Medical and Health  3.530  5.760  9.71  14.0
## Personal Care.1940             Personal Care  1.040  1.980  2.45   3.4
## Private Education.1940     Private Education  0.341  0.974  1.80   2.6
##                          X1960
## Food and Tobacco.1940    86.80
## Household Operation.1940 46.20
## Medical and Health.1940  21.10
## Personal Care.1940        5.40
## Private Education.1940    3.64
```

The reshape Function

Long → Wide Transformation

```
longdat <- data.frame(id = as.integer(mapply(rep, 1:3, 3)),
                      visit = rep(1:3, 3),
                      x = rnorm(9), y = rnorm(9))
longdat
```

```
##   id visit           x           y
## 1  1     1 -0.81388880  2.13614580
## 2  1     2  0.02367343  0.59968110
## 3  1     3 -0.75318770  0.10254288
## 4  2     1 -0.44997151  0.08131841
## 5  2     2  1.20391574  1.42280041
## 6  2     3 -0.85149491 -0.70167391
## 7  3     1 -0.60408186  0.21489069
## 8  3     2  0.18008182  0.97398995
## 9  3     3 -0.94984086 -0.99838284
```

The `reshape` Function

Long $\rightarrow$ Wide Transformation

Arguments needed (beside the data set to reshape)

- `idvar`: names of variable that define the experimental units
- `v.names`: Variables that are used to create the multiple variables in the wide format
- `timevar` identifies the "time" variable for the repeated measurements
- `direction`: `"long"` or `"wide"`

The reshape Function

Long → Wide Transformation

```
widedat <- reshape(longdat, idvar = "id", v.names = c("x", "y"),
                   timevar = "visit", direction = "wide")
widedat
```

```
##    id         x.1         y.1         x.2       y.2         x.3         y.3
## 1   1 -0.8138888 2.13614580 0.02367343 0.5996811 -0.7531877  0.1025429
## 4   2 -0.4499715 0.08131841 1.20391574 1.4228004 -0.8514949 -0.7016739
## 7   3 -0.6040819 0.21489069 0.18008182 0.9739900 -0.9498409 -0.9983828
```

The reshape Function

Long → Wide Transformation

Wide to long transformation again easy from the reshape()'d data

```
reshape(widedat)
```

```
##     id visit           x           y
## 1.1  1     1 -0.81388880  2.13614580
## 2.1  2     1 -0.44997151  0.08131841
## 3.1  3     1 -0.60408186  0.21489069
## 1.2  1     2  0.02367343  0.59968110
## 2.2  2     2  1.20391574  1.42280041
## 3.2  3     2  0.18008182  0.97398995
## 1.3  1     3 -0.75318770  0.10254288
## 2.3  2     3 -0.85149491 -0.70167391
## 3.3  3     3 -0.94984086 -0.99838284
```

Your Turn

Consider the `mammaca` data set

- Based on a real study on mammary carcinoma (the variable names are authentic)
- 65 women were followed for 4 visits
- We want to look at the evolution of all the variables over the visits

Reshape the data in the long format

Your Turn

Consider the `transplant` data set. It contains informations on the survival of patients on the waiting list for the Stanford heart transplant program. Important variables are

- `fustat` Follow up status (1 dead; 0 alive at the end of the study)
- `futime` Follow up time
- `wait.time` Waiting time until transplant (NA if no transplant
- `transplant` Transplant status (0 no transplant, 1 transplanted at `wait.time`)

Your Turn

To be analysed correctly, these data need to be transformed in some kind of long format. Consider 2 individuals as illustration

```
id   accept.dt      age surgery fustat futime wait.time transplant
1   1967-11-15 30.84463       0      1     49        NA          0
4   1968-03-28 40.26283       0      1     38        35          1
```

The resulting data set should look like

```
id   accept.dt      age surgery status start stop transplant
1   1967-11-15 30.84463       0      1     0   49          0
2   1968-03-28 40.26283       0      0     0   35          0
2   1968-03-28 40.26283       0      1     35  38          1
```

Table of Contents

Combining Data Frames

At the most basic level, two or more data frames can be combined by rows using `rbind`, or by columns using `cbind`

rbind Data frames must have the same number of columns

cbind The data must have the same number of rows

```
d1 <- data.frame(x = rnorm(4), y = sample(letters, 4, replace = FALSE))
d2 <- data.frame(x = rnorm(4), y = sample(letters, 4, replace = FALSE))
d1
```

```
##             x y
## 1  0.51149711 w
## 2  0.02354815 r
## 3  0.40077915 f
## 4 -0.22095408 n
```

Combining Data Frames

cbind

```
cbind(d1, d2)
```

```
##             x y          x y
## 1  0.51149711 w -1.0056397 z
## 2  0.02354815 r -1.5335319 g
## 3  0.40077915 f -1.2513303 c
## 4 -0.22095408 n  0.6772757 r
```

Duplicate column names are not detected

Combining Data Frames

cbind

```
cbind(d1, d2)
```

```
##              x y           x y
## 1  0.51149711 w -1.0056397 z
## 2  0.02354815 r -1.5335319 g
## 3  0.40077915 f -1.2513303 c
## 4 -0.22095408 n  0.6772757 r
```

Duplicate column names are not detected

```
cbind(d1, z = c(1, 2))
```

```
##              x y z
## 1  0.51149711 w 1
## 2  0.02354815 r 2
## 3  0.40077915 f 1
## 4 -0.22095408 n 2
```

Smaller vectors/data are recycled

Combining Data Frames

rbind

For using `rbind`, names and classes of values to be joined must match

```
rbind(d1, d2)
```

```
##             x y
## 1  0.51149711 w
## 2  0.02354815 r
## 3  0.40077915 f
## 4 -0.22095408 n
## 5 -1.00563975 z
## 6 -1.53353187 g
## 7 -1.25133028 c
## 8  0.67727572 r
```

Combining Data Frames

rbind

```
d1$y <- factor(d1$y)
d2$y <- factor(d2$y)
rbind(d1, d2)
```

```
##             x y
## 1  0.51149711 w
## 2  0.02354815 r
## 3  0.40077915 f
## 4 -0.22095408 n
## 5 -1.00563975 z
## 6 -1.53353187 g
## 7 -1.25133028 c
## 8  0.67727572 r
```

It works!

Combining Data Frames

rbind

```
(d3 <- rbind(d1, data.frame(x = "X", y = 12)))
```

```
## Warning in `[<-.factor`(`*tmp*`, ri, value = structure(c(4L, 3L, 1L, 2L, :  invalid factor
level, NA generated
```

```
##                     x    y
## 1  0.511497108998716    w
## 2 0.0235481470618808    r
## 3  0.400779147687347    f
## 4 -0.220954084894623    n
## 5                   X <NA>
```

```
sapply(d3, class)
```

```
##           x           y
## "character"    "factor"
```

```
rbind(d1, data.frame(y = "X", d = 12))
```

```
## Error in match.names(clabs, names(xi)):  names do not match previous
names
```

Merge Data Frames

For more complicated tasks, the `merge` function can be used

- The default behaviour of merge is to join together rows of the data frames based on the values of all of the variables (columns) that the data frames have in common (*natural join*)
- When called without argument, `merge` only returns rows which have observations in both data frames

```
dd1 <- data.frame(a = c(1,2,4,5,6), x = c(9,12,14,21,8))
dd2 <- data.frame(a=c(1,3,4,6),y=c(8,14,19,2))
merge(dd1, dd2)
```

```
##   a  x  y
## 1 1  9  8
## 2 4 14 19
## 3 6  8  2
```

Merge Data Frames

To change the default behaviour the arguments

- `all = TRUE`: Includes all rows (*full outer join*)
- `all.x = TRUE`: Includes all rows of the first data frame (*left outer join*)
- `all.y = TRUE`: Includes all rows of the second data frame (*right outer join*)

Merge Data Frames

```
merge(dd1, dd2, all = TRUE)
```

```
##   a  x  y
## 1 1  9  8
## 2 2 12 NA
## 3 3 NA 14
## 4 4 14 19
## 5 5 21 NA
## 6 6  8  2
```

Merge Data Frames

```r
merge(dd1, dd2, all = TRUE)
```

```
##   a  x  y
## 1 1  9  8
## 2 2 12 NA
## 3 3 NA 14
## 4 4 14 19
## 5 5 21 NA
## 6 6  8  2
```

```r
merge(dd1, dd2, all.x = TRUE)
```

```
##   a  x  y
## 1 1  9  8
## 2 2 12 NA
## 3 4 14 19
## 4 5 21 NA
## 5 6  8  2
```

Merge Data Frames

```
merge(dd1, dd2, all = TRUE)
```

```
##   a  x  y
## 1 1  9  8
## 2 2 12 NA
## 3 3 NA 14
## 4 4 14 19
## 5 5 21 NA
## 6 6  8  2
```

```
merge(dd1, dd2, all.x = TRUE)
```

```
##   a  x  y
## 1 1  9  8
## 2 2 12 NA
## 3 4 14 19
## 4 5 21 NA
## 5 6  8  2
```

```
merge(dd1, dd2, all.y = TRUE)
```

```
##   a  x  y
## 1 1  9  8
## 2 3 NA 14
## 3 4 14 19
## 4 6  8  2
```

Merge Data Frames

- The by argument permits to specify the name of the variables that should be used for the merging.
- If the merging variables have different names in the data frames to merge, the by.x and by.y arguments can be used

```
dd1$PAT <- letters[1:5]
dd2$id <- letters[3:6]
merge(dd1, dd2, by.x = c("PAT"), by.y = c("id"))

##   PAT a.x  x a.y  y
## 1   c   4 14   1  8
## 2   d   5 21   3 14
## 3   e   6  8   4 19
```

Merge Data Frames

- The by argument permits to specify the name of the variables that should be used for the merging.
- If the merging variables have different names in the data frames to merge, the by.x and by.y arguments can be used

```
dd1$PAT <- letters[1:5]
dd2$id <- letters[3:6]
merge(dd1, dd2, by.x = c("PAT"), by.y = c("id"))

##   PAT a.x  x a.y  y
## 1   c   4 14   1  8
## 2   d   5 21   3 14
## 3   e   6  8   4 19
```

Note the new variables a.x and a.y

Your Turn