# 5 Modifying and Combining Data Sets

## 5.1 Stacking Data Sets Using `rbind`

The `rbind` function concatenates or stacks two or more data sets with all of the same variables but different observations. You might, for example, have data from two different locations or data taken at two separate times, but you need the data together for analysis.

You specify the new data set in the left hand side, then list the names of the old data sets you want to combine :

```
new-data-set = rbind(data-set-1,...,data-set-n)
```

The number of observations in the new data set will equal the sum of number of observations in the old data sets. The order of observations is determined by the order of the list of old data sets. All old data sets must contain the same set of variables and in the same order.

**Example** The following contains data for two entrances of an amusement park. The files has entrance (S or N), pass number, size of parties and their ages. The file for the north entrace has an N and the parking lot.

```
#South.dat
S 43 3 27
S 44 3 24
S 45 3  2
#North.dat
N 21 5 41 1
N 87 4 33 3
N 65 2 67 1
N 66 2  7 1
```

The following program reads the data and prints them. The third part combines the two data using `rbind`, it also creates a new variable `AmountPaid`, which tells how much each customer paid based on their age.

```
(southentrance <- read.table("./dataRaw/South.dat",
 head = FALSE, col.names = c("Entrance", "PassNumber", "PartySize", "Age")))

##   Entrance PassNumber PartySize Age
## 1        S         43         3  27
## 2        S         44         3  24
## 3        S         45         3   2

(northentrance <- read.table("./dataRaw/North.dat",
 head = FALSE, col.names = c("Entrance", "PassNumber", "PartySize", "Age", "Lot")))

##   Entrance PassNumber PartySize Age Lot
## 1        N         21         5  41   1
## 2        N         87         4  33   3
## 3        N         65         2  67   1
## 4        N         66         2   7   1

southentrance$Lot <- NA
both <- rbind(southentrance, northentrance)
both$AmountPaid <- c(0, 35, 27)[cut(both$Age, breaks=c(-99, 3, 65, 999), label = FALSE)]
```

The following are the results. Notice that the final data has missing values for the variable `Lot` for all observations which came from the south entrance.

```
both

##   Entrance PassNumber PartySize Age Lot AmountPaid
## 1        S         43         3  27  NA         35
## 2        S         44         3  24  NA         35
```

```
## 3        S         45         3   2  NA         0
## 4        N         21         5  41   1        35
## 5        N         87         4  33   3        35
## 6        N         65         2  67   1        27
## 7        N         66         2   7   1        35
```

## 5.2 Interleaving Data Sets Using `merge`

The previous section explained how to stack data sets that have all the same variables at the same order, but different observations. However, you usually have data sets with similar set of variables, not necessarily in the same order. You could change the order and add in the missing variables before stacking the data sets. But a more general *join* operation would be more efficient, which performs all these operation for you. You need to use `merge` for such operations. Here is the general form:

```
new-data-set = merge(data-set-1, data-set-2, all=TRUE)
```

You specify the new data set in the left hand side, then names of two old data sets you want to combine, the `all=TRUE` denotes that all observations will be returned, whether they are from the first or the second data set. The number of observations in the new data set will equal the sum of number of observations in the old data sets. The order of observations is determined by the order of the list of old data sets. If one of the data sets has a variable not contained in the other data sets, values of that variable will be set to missing for observations from the other data sets.

**Example** We will use the amusement park data again.

```
#South.dat
S 43 3 27
S 44 3 24
S 45 3  2
#North.dat
N 21 5 41 1
N 87 4 33 3
N 65 2 67 1
N 66 2  7 1
```

Instead of stacking the two data sets, this program interleaves the data sets by pass number, entrance, party number and age. The data set `interleave` was created by combining the two data sets.

```
interleave <- merge(southentrance, northentrance[, c(2, 1, 3:5)], all = TRUE)
```

Here are the results. Notice that the results are the same even though the variables in the `northentrance` was permuted and missing values for `Lot` were automatically created.

```
interleave

##   Entrance PassNumber PartySize Age Lot
## 1        S         43         3  27  NA
## 2        S         44         3  24  NA
## 3        S         45         3   2  NA
## 4        N         21         5  41   1
## 5        N         65         2  67   1
## 6        N         66         2   7   1
## 7        N         87         4  33   3
```

## 5.3 Combining Data Sets Using a One-to-One Match Merge

When you want to match observations from one data set with observations from another, use the `merge` function. If you know the two data sets are in *EXACTLY* the same order, you don't have to have any common variables between the data sets. Typically, however, you will want to have, for matching purposes, a common variable or several variables which taken together uniquely identify each observation. This is important. Having a common variable to merge by ensures that the observations are properly matches. For example, to merge patient data with billing data, you would use the patient ID as a matching variable.

Merging data is a simple process. First name the new data set to hold the results, follow with a `merge` function, and list the data sets to be combined. Use the `by` argument to indicate the common variables:

```
new-data-set=merge(data-set-1,data-set-2,
  by=variable-list)
```

If you merge two data sets, and they have variables with the same names-besides the `by` variables, variables from both data sets will be kept, each renamed with suffix `.x` and `.y`.

**Example** A Belgian chocolatier keeps track of the chocolate sold each day. The code number for each chocolate and the number of pieces sold that day are kept in a file. In a separate file she keeps the names and descriptions of each chocolate as well as the code number. In order to print the day's sales along with the descriptions of the chocolates, the two files must be merged together using the code number. Here is a sample of the data.

```
## Sales dat
C865 15
K086  9
A536 21
S163 34
K014  1
A206 12
B713 29
## Description
A206 Mokka     Coffee buttercream in dark chocolate
A536 Walnoot   Walnut halves in bed of dark chocolate
B713 Frambozen Raspberry marzipan covered in milk chocolate
C865 Vanille   Vanilla-flavored rolled in ground hazelnuts
K014 Kroon     Milk chocolate with a mint cream center
K086 Koning    Hazelnut paste in dark chocolate
M315 Pyramide  White with dark chocolate trimming
S163 Orbais    Chocolate cream in dark chocolate
```

```r
descriptions <- read.fwf("./dataRaw/Chocolate.dat",
  widths = c(4, 10, 46),
  col.names = c("CodeNum", "Name", "Description"))
descriptions <- na.omit(descriptions)

sales <- read.table("./dataRaw/chocsales.dat", header = FALSE,
 col.names = c("CodeNum", "PiecesSold"))
```

The first parts of the program read the descriptions and sales data. The next part of the program creates a data set by merging the `sales` and the `descriptions` data set. The common variable `CodeNum` in the `by` statement is used for matching purposes.

```r
merge(sales, descriptions, by = "CodeNum", all = TRUE)

##   CodeNum PiecesSold       Name                                  Description
## 1    A206         12  Mokka             Coffee buttercream in dark chocolate
## 2    A536         21  Walnoot         Walnut halves in bed of dark chocolate
## 3    B713         29  Frambozen  Raspberry marzipan covered in milk chocolate
## 4    C865         15  Vanille     Vanilla-flavored rolled in ground hazelnuts
## 5    K014          1  Kroon           Milk chocolate with a mint cream center
## 6    K086          9  Koning                 Hazelnut paste in dark chocolate
## 7    S163         34  Orbais                Chocolate cream in dark chocolate
## 8    M315         NA  Pyramide             White with dark chocolate trimming
```

The above output shows the data set after merging. Notice that the final data set has a missing value for `PiecesSold` in the eighth observation. This is because there were no sales for the chocolate. All observations from both data sets were included in the final data set whether they had a match or not.

## 5.4 Combining Data Sets Using a One-to-Many Match Merge

Sometimes you need to combine two data sets by matching one observation from one data set with more than one observation in another. Suppose you had data for every state in the U.S. and wanted to combine it with data from every county. This would be a one-to-many match merge, because each state observation matches with many county observations.

The statements for a one-to-many match merge are similar to the statements for a one-to-one match merge.

```
new-data-set=merge(data-set-1, data-set-2,
  by = variable-list, all.x = T/F, all.y = T/F)
```

The order of the data sets in the `merge` function matters. `all.x=T` and `all.y=F` mean that extra observations from `data-set-1` will be added to the results, even if there is no matching observation in `data-set-2` using the `by` variable. This is termed *left outer join*. `all.x=F` and `all.y=T` means the same for `data-set-2`, is termed *right outer join*. When both are "T", it is the default *full outer join* between two data frames.

You usually need a `by` argument for one-to-many merge. R uses the variables listed in the `by` argument to decide which observations belong together. Without any `by` varialbe for match, R simply produces a *Cartesian product* of the observations from two data sets. This may generate large data sets, slow and probably is not what you want.

If you merge two data sets, and they have variables with the same names-besides the `by` variables, variables from both data sets will be kept, each renamed with suffix `.x` and `.y`.

**Example** A distributor of athletic shoes is putting all its shoes on sale at 20 to 30% off the regular price. The distributor has two data files, one with information about each type of shoe and one with the discount factors. The first file contains one record for each shoe with values for style, type of exercise and regular price. The second file contains one record for each type of exercise and its discount. Here are the two raw data files:

```
## Shoes data
Max Flight      running 142.99
Zip Fit Leather walking  83.99
Zoom Airborne   running 112.99
Light Step      walking  73.99
Max Step Woven  walking  75.99
Zip Sneak       c-train  92.99
## Discount data
c-train .25
running .30
walking .20
```

To find the sale price, the following program combines the two data files:

```
regular <- read.fwf("./dataRaw/Shoe.dat", widths = c(15, 9, 6),
                    col.names = c("Style", "ExerciseType", "RegularPrice"))
regular <- na.omit(regular)
discount <- read.table("./dataRaw/Disc.dat", header = FALSE,
                       col.names=c("ExerciseType", "Adjustment"))
levels(regular$ExerciseType) <- levels(discount$ExerciseType)

prices <- merge(regular, discount, by="ExerciseType", all.x = TRUE)
prices$Newprice <- with(prices, round(RegularPrice * (1 - Adjustment)))
```

The first part reads the regular prices, creating a data set named `regular`. The second creates a data named `discount`. The `levels` of the `by` variable must match in order to perform the merge. The `merge` statement creates a data set named `prices`, merging the first two data sets, keeping all observations from `regular`, thus a *left* outer join. The output look like this:

```
##   ExerciseType           Style RegularPrice Adjustment Newprice
## 1      c-train Zip Sneak                93       0.25       70
## 2      running Max Flight              143       0.30      100
## 3      running Zoom Airborne           113       0.30       79
## 4      walking Zip Fit Leather          84       0.20       67
## 5      walking Light Step               74       0.20       59
## 6      walking Max Step Woven           76       0.20       61
```

Notice that the values for `Adjustment` from the `Discount` data set are repeated for every observation in the `regular` data set with the same value of `ExerciseType`.

## 5.5 Merging Summary Statistics with the Original Data

Once in a while you need to combine summary statistics with your data, such as when you want to compare each observation to the group mean, or when you want to calculate a percentage using the group total. To do this, summarize your data using `aggregate`, and put the results in a new data set. Then merge the summarized data back with the original data using a one-to-many match merge.

**Example 1** A distributor of athletic shoes is considering doing a special promotion for the top selling styles. The vice-president of marketing has asked you to produce a report. The report should be divided by type of exercise, and the percentage of sales for each style within its type. For each shoe, the raw data file contains the style name, type of exercise and the total sales for the last quarter.

```
Max Flight      running 1930
Zip Fit Leather walking 2250
Zoom Airborne   running 4150
Light Step      walking 1130
Max Step Woven  walking 2230
Zip Sneak       c-train 1190
```

The program reads the raw data. Then it summarized the data with `aggregate` by the variable `ExerciseType`, creates a summary data named `summarydata`, containing a variable named `Total`, which equals the sum of the variable `Sales`.

```
shoes <- read.fwf("./dataRaw/Shoesales.dat",
                  widths = c(15, 9, 6), col.names = c("Style", "ExerciseType", "Sales"))
shoes <- na.omit(shoes)
summarydata <- aggregate(Sales ~ ExerciseType, data = shoes, FUN = sum)
names(summarydata)[2] <- "Total"
summarydata

##   ExerciseType Total
## 1      c-train  1190
## 2      running  6080
## 3      walking  5610
```

In the second part of the program, the original data set `shoes` is merged with `summarydata` to make a new data set `shoesummary`. It also computes a new variable called `Percent`.

```
shoesummary <- merge(shoes, summarydata, by = "ExerciseType", all.x = TRUE)
shoesummary$Percent <- with(shoesummary, Sales * 100 / Total)
shoesummary

##   ExerciseType           Style Sales Total Percent
## 1      c-train Zip Sneak        1190  1190     100
## 2      running Max Flight       1930  6080      32
## 3      running Zoom Airborne    4150  6080      68
## 4      walking Zip Fit Leather  2250  5610      40
## 5      walking Light Step       1130  5610      20
## 6      walking Max Step Woven   2230  5610      40
```

## 5.6 Adding Margins to a Data Set

The previous section discusses merging summary statistics with your data, that is summarizing variables across the observations. You may also need to summarize all variables corresponding to each observations, especially when all variables are recorded on the same scale. To do this, you will need to treat a data frame as a matrix and use `rowMeans` function to average over the columns instead of rows, put the results in a new data set or combine it with the original data sets.

**Example** A distributor of athletic shoes is evaluating the performance of her sales. The report should indicate the seasonal trend of the sales and the deviation of the sales from the overall mean. The raw data file contains the names of the sales and their sales in each season:

```
name  spring summer autumn winter
      Jane.Smith     14      18      11      12
    Robert.Jones     17      18      10      13
     Dick.Rogers     12      16       9      14
 William.Edwards     15      14      11      10
     Janet.Jones     11      17      11      16
```

The following program reads in the data and use `rowMeans` to calculate each people's average sales, its departure from the grand mean.

```
frame <- read.table("./dataRaw/sales.txt", header = TRUE, as.is = "name")
people <- rowMeans(frame[, 2:5])
(peopleeffect <- people - mean(people))

## [1]  0.30  1.05 -0.70 -0.95  0.30
```

The column means, which is the usual summary statistics for seasonal effects, are calculated in the similar way.

```
(season <- colMeans(frame[, 2:5]))

## spring summer autumn winter
##     14     17     10     13
```

Now we make a data frame that summarize the seasonal effects. We first create a data frame `newrow` to hold the results. The first column is the name, followed by the seasonal effects vectors.

```
newrow <- frame[1, ]
newrow[1,1] <- "Seasonal Effect"
newrow[1,2:5] <- season - mean(season)
```

The last task is to calculate the sales as deviation from the grand means and combine the seasonal effects with the original data, and add a column for sales performance.

```
newframe <- frame
newframe[,2:5] <- frame[,2:5] - mean(season)

data.frame(rbind(newframe, newrow),
           people=c(peopleeffect, mean(people)))

##                name spring summer autumn winter people
## 1      Jane.Smith   0.55   4.55   -2.5  -1.45   0.30
## 2    Robert.Jones   3.55   4.55   -3.5  -0.45   1.05
## 3     Dick.Rogers  -1.45   2.55   -4.5   0.55  -0.70
## 4 William.Edwards   1.55   0.55   -2.5  -3.45  -0.95
## 5     Janet.Jones  -2.45   3.55   -2.5   2.55   0.30
## 6 Seasonal Effect   0.35   3.15   -3.1  -0.45  13.45
```

Note that sales are highest in summer (+3.15) and lowest in autumn (−3.1). Robert Jones is the most effective sale person (1.05) and William Edwards is the least effective (−0.95).

## 5.7 Changing Observations to Variables Using `reshape2`

We have already seen ways to combine data sets, create new variables, and sort data. Now, using the `reshape2` package, we will flip data, that is to change observations into variables.

The `reshape2` transposes data sets, turning observations into variables or variables into observations. In most cases, to convert observations into variables, you first use the following statement.

```
long-data-set=melt(old-data-set, id-variables, measure-variables)
```

In the `melt` statement, `old-data-set` refers to the data set you want to transpose, and `long-data-set` refers to a long form of data suitable for transposition. `id-variables` are list of id variables. These variables are usually categorical, with combinations become the rows of the output data set, the "measure-variables" are variables that are stack onto each other in the long form, repeated for each unique combination of the id variables.

After `melt`, the `dcast` function can transpose the data:

```
new-data-set=dcast(long-data-set, by-variables ~ id-variables)
```

In the `dcast` statement, the `by-variables` are grouping variables separated by `+` that you want to keep as variables. These variables are included in the transposed data set, but they are not themselves transposed. The transposed data set will have one observation for each unique level of by-variables. The `id-variables` names the variable whose values become the variable names. The `id-variables` must occur only once in the data set within each by group. The function assumes that variables to transpose are in the field called `value`.

**Example** Suppose you have the following data about players for minor league baseball teams. You have the team name, player's number, the type of data (salary or batting average), and the entry:

```
Garlics 10 salary 43000
Peaches  8 salary 38000
Garlics 21 salary 51000
Peaches 10 salary 47500
Garlics 10 batavg .281
Peaches  8 batavg .252
Garlics 21 batavg .265
Peaches 10 batavg .301
```

You want to look at the relationship between batting average and salary. To do this, salary and batting average must be variables. The following program reads the raw data into a data frame. The data are first melted using all categorical variables, with the only one measurement variable `Entry`. The idea is to first make the data as long as possible, with each row denote one single variable corresponds to each unique combinations of id variables.

```r
library(reshape2)
baseball <- read.table("./dataRaw/Transpos.dat",
                       head = FALSE, col.names = c("Team", "Player", "Type", "Entry"))
baseball.m <- melt(baseball,
                   idvars=c("Team", "Player", "Type"), measure.vars = "Entry")
head(baseball.m)

##      Team Player   Type variable   value
## 1 Garlics     10 salary    Entry 4.3e+04
## 2 Peaches      8 salary    Entry 3.8e+04
## 3 Garlics     21 salary    Entry 5.1e+04
## 4 Peaches     10 salary    Entry 4.8e+04
## 5 Garlics     10 batavg    Entry 2.8e-01
## 6 Peaches      8 batavg    Entry 2.5e-01
```

The data are then transposed using the `dcast` function. In the function, the by variables are `Team` and `Player`. You want those variables to remain in the output, and they define the new observations (you want only one observation for each team and player combination). The ID variable is `Type`, whose values (`salary` and `batavg`) will be the new variable names. By default, the variable to be transposed is `value`.

```
dcast(baseball.m, Team + Player ~ Type)

##      Team Player batavg salary
## 1 Garlics     10   0.28  43000
## 2 Garlics     21   0.26  51000
## 3 Peaches      8   0.25  38000
## 4 Peaches     10   0.30  47500
```

The wide format data set can be transposed back to long format, simply changing the by and id variables.

```
dcast(baseball.m, Team + Player + Type ~ variable)

##      Team Player   Type   Entry
## 1 Garlics     10 batavg 2.8e-01
## 2 Garlics     10 salary 4.3e+04
## 3 Garlics     21 batavg 2.6e-01
## 4 Garlics     21 salary 5.1e+04
## 5 Peaches      8 batavg 2.5e-01
## 6 Peaches      8 salary 3.8e+04
## 7 Peaches     10 batavg 3.0e-01
## 8 Peaches     10 salary 4.8e+04
```

## 5.8 Exercises

The following hypotheticall data list the family income and spending for three years of three families.

```
#fam.txt
1 Jones
2 Smith
3 Jackson
#income.txt
1 96 40000 38000
1 97 40500 39000
1 98 41000 40000
2 96 45000 42000
2 97 45400 43000
2 98 45800 44000
3 96 75000 70000
3 97 76000 71000
3 98 77000 72000
```

Summarize the data by year for each family as a data frame:

```
     name 96_income 96_spend 97_income 97_spend 98_income 98_spend
1 Jackson     75000    70000     76000    71000     77000    72000
2   Jones     40000    38000     40500    39000     41000    40000
3   Smith     45000    42000     45400    43000     45800    44000
```

## 5.9 Exercises

Load swimming.txt file, and reshape it to get a data frame with the following columns (you may need to assign name to collumns manually): treatment, subject, time, response.