

The doBy package

Søren Højsgaard

December 8, 2007

Contents

1	Introduction	2
2	Data	2
3	The summaryBy function	2
3.1	Basic usage	2
3.2	Using a list of functions	3
3.3	Naming output variables with the <code>postfix</code> argument	3
3.4	Copying variables out with the <code>id</code> argument	4
3.5	Statistics on functions of data	4
3.6	Using <code>”.</code> on the left hand side of a formula	5
3.7	Using <code>”.</code> on the right hand side of a formula	5
3.8	Using <code>”1</code> on the right hand side of the formula	5
3.9	Preserving names of variables using <code>keep.names</code>	5
4	The orderBy function	6
5	The splitBy function	6
6	The sampleBy function	6
7	The subsetBy function	7
8	The transformBy function	7
9	The lapplyBy function	7
10	Miscellaneous	7
10.1	The <code>esticon</code> function	7
11	Final remarks	9

1 Introduction

The doBy package grew out of a need to calculate groupwise summary statistics in a simple way, much in the spirit of PROC SUMMARY of the SAS system. We have tried to keep the interface to the functions based on specifying formulas.

```
> library(doby)
```

2 Data

The usage of the doBy package is based on the following datasets.

CO2 data The CO2 data frame comes from an experiment on the cold tolerance of the grass species *Echinochloa crus-galli*. To limit the amount of output we modify names and levels of variables as follows

```
> data(CO2)
> CO2 <- transform(CO2, Treat = Treatment, Treatment = NULL)
> levels(CO2$Treat) <- c("nchil", "chil")
> levels(CO2$Type) <- c("Que", "Mis")
> CO2 <- subset(CO2, Plant %in% c("Qn1", "Qc1", "Mn1", "Mc1"))
```

Airquality data The airquality dataset contains air quality measurements in New York, May to September 1973. The months are coded as 5, ..., 9. To limit the output we only consider data for two months:

```
> airquality <- subset(airquality, Month %in% c(5, 6))
```

Dietox data The dietox data are provided in the doBy package and result from a study of the effect of adding vitamin E and/or copper to the feed of slaughter pigs.

3 The summaryBy function

The summaryBy function is used for calculating quantities like “the mean and variance of x and y for each combination of two factors A and B ”. Examples are based on the CO2 data.

3.1 Basic usage

For example, the mean and variance of uptake and conc for each value of Plant is obtained by:

```
> summaryBy(conc + uptake ~ Plant, data = CO2, FUN = function(x) {
+   c(m = mean(x), v = var(x))
+ })
```

	Plant	conc.m	conc.v	uptake.m	uptake.v
1	Qn1	435	100950	33.23	67.48
2	Qc1	435	100950	29.97	69.47
3	Mn1	435	100950	26.40	75.59
4	Mc1	435	100950	18.00	16.96

Defining the function to return named values as above is the recommended use of `summaryBy`. The function can also be defined outside the call to `summaryBy`:

```
> myfun1 <- function(x) {
+   c(m = mean(x), v = var(x))
+ }
> summaryBy(conc + uptake ~ Plant, data = CO2, FUN = myfun1)
```

Note that the values returned by the function has been named as `m` and `v`. If the result of the function(s) are not named, then the names in the output data in general become less intuitive:

```
> myfun2 <- function(x) {
+   c(mean(x), var(x))
+ }
> summaryBy(conc + uptake ~ Plant, data = CO2, FUN = myfun2)
```

	Plant	conc.myfun21	conc.myfun22	uptake.myfun21	uptake.myfun22
1	Qn1	435	100950	33.23	67.48
2	Qc1	435	100950	29.97	69.47
3	Mn1	435	100950	26.40	75.59
4	Mc1	435	100950	18.00	16.96

3.2 Using a list of functions

It is possible use a list of functions. A typical usage will be by invoking a list of predefined functions:

```
> summaryBy(uptake ~ Plant, data = CO2, FUN = c(mean, var, median))
```

	Plant	uptake.mean	uptake.var	uptake.median
1	Qn1	33.23	67.48	35.3
2	Qc1	29.97	69.47	32.5
3	Mn1	26.40	75.59	30.0
4	Mc1	18.00	16.96	18.9

Slightly more elaborate is

```
> mymed <- function(x) c(med = median(x))
> summaryBy(uptake ~ Plant, data = CO2, FUN = c(mean, var, mymed))
```

	Plant	uptake.mean	uptake.var	uptake.med
1	Qn1	33.23	67.48	35.3
2	Qc1	29.97	69.47	32.5
3	Mn1	26.40	75.59	30.0
4	Mc1	18.00	16.96	18.9

The naming of the output variables determined from what the functions returns. The names of the last two columns above are imposed by `summaryBy` because `myfun2` does not return named values.

3.3 Naming output variables with the postfix argument

The `postfix` argument gives an alternative way of naming the output variables: For example, the functions `myfun1` and `myfun2` both returns two values. These can be named as:

```
> summaryBy(conc + uptake ~ Plant, data = CO2, postfix = list(c("mean1",
+ "var1"), c("mean2", "var2")), FUN = c(myfun1, myfun2))
```

	Plant	conc.mean1	conc.var1	uptake.mean1	uptake.var1	conc.mean2	conc.var2
1	Qn1	435	100950	33.23	67.48	435	100950
2	Qc1	435	100950	29.97	69.47	435	100950
3	Mn1	435	100950	26.40	75.59	435	100950
4	Mc1	435	100950	18.00	16.96	435	100950

	uptake.mean2	uptake.var2
1	33.23	67.48
2	29.97	69.47
3	26.40	75.59
4	18.00	16.96

3.4 Copying variables out with the id argument

To get the value of the `Type` and `Treat` in the first row of the groups (defined by the values of `Plant`) copied to the output dataframe we use the `id` argument: as:

```
> summaryBy(conc + uptake ~ Plant, data = CO2, FUN = myfun1, id = ~Type +
+ Treat)
```

	Plant	conc.m	conc.v	uptake.m	uptake.v	Type	Treat
1	Qn1	435	100950	33.23	67.48	Que	nchil
2	Qc1	435	100950	29.97	69.47	Que	chil
3	Mn1	435	100950	26.40	75.59	Mis	nchil
4	Mc1	435	100950	18.00	16.96	Mis	chil

3.5 Statistics on functions of data

We may want to calculate the mean and variance for the logarithm of `uptake`, for `uptake+conc` (not likely to be a useful statistic) as well as for `uptake` and `conc`. This can be achieved as:

```
> summaryBy(log(uptake) + I(conc + uptake) + conc + uptake ~ Plant,
+ data = CO2, FUN = myfun1)
```

	Plant	log(uptake).m	log(uptake).v	conc+uptake.m	conc+uptake.v	conc.m	conc.v
1	Qn1	3.467	0.10168	468.2	104747	435	100950
2	Qc1	3.356	0.11873	465.0	105297	435	100950
3	Mn1	3.209	0.17928	461.4	105642	435	100950
4	Mc1	2.864	0.06874	453.0	103157	435	100950

	uptake.m	uptake.v
1	33.23	67.48
2	29.97	69.47
3	26.40	75.59
4	18.00	16.96

If one does not want output variables to contain parentheses then setting `p2d=TRUE` causes the parentheses to be replaced by dots (“.”).

```
> summaryBy(log(uptake) + I(conc + uptake) ~ Plant, data = CO2,
+ p2d = TRUE, FUN = myfun1)
```

	Plant	log.uptake..m	log.uptake..v	conc+uptake.m	conc+uptake.v
1	Qn1	3.467	0.10168	468.2	104747
2	Qc1	3.356	0.11873	465.0	105297
3	Mn1	3.209	0.17928	461.4	105642
4	Mc1	2.864	0.06874	453.0	103157

3.6 Using "." on the left hand side of a formula

It is possible to use the dot (".") on the left hand side of the formula. The dot means "all numerical variables which do not appear elsewhere" (i.e. on the right hand side of the formula and in the `id` statement):

```
> summaryBy(log(uptake) + I(conc + uptake) + . ~ Plant, data = CO2,
+ FUN = myfun1)

  Plant log(uptake).m log(uptake).v conc+uptake.m conc+uptake.v conc.m conc.v
1  Qn1          3.467      0.10168          468.2      104747    435 100950
2  Qc1          3.356      0.11873          465.0      105297    435 100950
3  Mn1          3.209      0.17928          461.4      105642    435 100950
4  Mc1          2.864      0.06874          453.0      103157    435 100950
 uptake.m uptake.v
1      33.23      67.48
2      29.97      69.47
3      26.40      75.59
4       18.00      16.96
```

3.7 Using "." on the right hand side of a formula

The dot (".") can also be used on the right hand side of the formula where it refers to "all non-numerical variables which are not specified elsewhere":

```
> summaryBy(log(uptake) ~ Plant + ., data = CO2, FUN = myfun1)

  Plant Type Treat log(uptake).m log(uptake).v
1  Qn1  Que  nchil          3.467      0.10168
2  Qc1  Que   chil          3.356      0.11873
3  Mn1  Mis  nchil          3.209      0.17928
4  Mc1  Mis   chil          2.864      0.06874
```

3.8 Using "1" on the right hand side of the formula

Using 1 on the right hand side means no grouping:

```
> summaryBy(log(uptake) ~ 1, data = CO2, FUN = myfun1)

  log(uptake).m log(uptake).v
1          3.224          0.1577
```

3.9 Preserving names of variables using `keep.names`

If the function applied to data only returns one value, it is possible to force that the summary variables retain the original names by setting `keep.names=TRUE`. A typical use of this could be

```
> summaryBy(conc + uptake + log(uptake) ~ Plant, data = CO2, FUN = mean,
+ id = ~Type + Treat, keep.names = TRUE)

  Plant conc uptake log(uptake) Type Treat
1  Qn1  435  33.23      3.467  Que  nchil
2  Qc1  435  29.97      3.356  Que   chil
3  Mn1  435  26.40      3.209  Mis  nchil
4  Mc1  435  18.00      2.864  Mis   chil
```

4 The orderBy function

Ordering (or sorting) a data frame is possible with the `orderBy` function. Suppose we want to order the rows of the `airquality` data by `Temp` and by `Month` (within `Temp`). This can be achieved by:

```
> x <- orderBy(~Temp + Month, data = airquality)
```

The first lines of the result are:

```
> head(x)

  Ozone Solar.R Wind Temp Month Day
5     NA      NA 14.3   56     5   5
18     6      78 18.4   57     5  18
25     NA      66 16.6   57     5  25
27     NA      NA  8.0   57     5  27
15    18      65 13.2   58     5  15
26     NA     266 14.9   58     5  26
```

If we want the ordering to be by decreasing values of one of the variables, we change the sign, e.g.

```
> x <- orderBy(~-Temp + Month, data = airquality)
> head(x)
```

```
  Ozone Solar.R Wind Temp Month Day
42     NA     259 10.9   93     6  11
43     NA     250  9.2   92     6  12
40     71     291 13.8   90     6   9
39     NA     273  6.9   87     6   8
41     39     323 11.5   87     6  10
36     NA     220  8.6   85     6   5
```

5 The splitBy function

Suppose we want to split the `airquality` data into a list of dataframes, e.g. one dataframe for each month. This can be achieved by:

```
> x <- splitBy(~Month, data = airquality)
```

Information about the grouping is stored as a dataframe in an attribute called `groupid` and can be retrieved with:

```
> attr(x, "groupid")

  Month
1     5
2     6
```

6 The sampleBy function

Suppose we want a random sample of 50 % of the observations from a dataframe. This can be achieved with:

```
> sampleBy(~1, frac = 0.5, data = airquality)
```

Suppose instead that we want a systematic sample of every fifth observation within each month. This is achieved with:

```
> sampleBy(~Month, frac = 0.2, data = airquality, systematic = T)
```

7 The subsetBy function

Suppose we want to take out those rows within each month for which the the wind speed is larger than the mean wind speed (within the month). This is achieved by:

```
> subsetBy(~Month, subset = Wind > mean(Wind), data = airquality)
```

Note that the statement `Wind>mean(Wind)` is evaluated within each month.

8 The transformBy function

The `transformBy` function is analogous to the `transform` function except that it works within groups. For example:

```
> transformBy(~Month, data = airquality, minW = min(Wind), maxW = max(Wind),
+   chg = sum(range(Wind) * c(-1, 1)))
```

9 The lapplyBy function

This `lapplyBy` function is a wrapper for calling `lapply` on the list resulting from first calling `splitBy`.

Suppose we want to calculate the weekwise feed efficiency of the pigs in the `dietox` data, i.e. weight gain divided by feed intake.

```
> data(dietox)
> dietox <- orderBy(~Pig + Time, data = dietox)
> v <- lapplyBy(~Pig, data = dietox, function(d) c(NA, diff(d$Weight)/diff(d$Feed)))
> dietox$FE <- unlist(v)
```

Technically, the above is the same as

```
> dietox <- orderBy(~Pig + Time, data = dietox)
> wdata <- splitBy(~Pig, data = dietox)
> v <- lapply(wdata, function(d) c(NA, diff(d$Weight)/diff(d$Feed)))
> dietox$FE <- unlist(v)
```

10 Miscellaneous

10.1 The esticon function

Consider a linear model which explains `Ozone` as a linear function of `Month` and `Wind`:

```
> data(airquality)
> airquality <- transform(airquality, Month = factor(Month))
> m <- lm(Ozone ~ Month * Wind, data = airquality)
> coefficients(m)
```

(Intercept)	Month6	Month7	Month8	Month9	Wind
50.748	-41.793	68.296	82.211	23.439	-2.368
Month6:Wind	Month7:Wind	Month8:Wind	Month9:Wind		
4.051	-4.663	-6.154	-1.874		

When a parameter vector β of (systematic) effects have been estimated, interest is often in a particular estimable function, i.e. linear combination $\lambda^\top \beta$ and/or testing the hypothesis $H_0 : \lambda^\top \beta = \beta_0$ where λ is a specific vector defined by the user.

Suppose for example we want to calculate the expected difference in ozone between consecutive months at wind speed 10 mph (which is about the average wind speed over the whole period).

The `esticon` function provides a way of doing so. We can specify several λ vectors at the same time. For example

```
> Lambda

      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
[1,]    0   -1    0    0    0    0   -10    0    0    0
[2,]    0    1   -1    0    0    0    10   -10    0    0
[3,]    0    0    1   -1    0    0    0    10   -10    0
[4,]    0    0    0    1   -1    0    0    0    10   -10

> esticon(m, Lambda)

Confidence interval ( WALD ) level = 0.95
  beta0 Estimate Std.Error t.value  DF Pr(>|t|) Lower.CI Upper.CI
1     0   1.2871   10.238  0.1257 106  0.90019  -19.010   21.585
2     0 -22.9503   10.310 -2.2259 106  0.02814  -43.392   -2.509
3     0   0.9954    7.094  0.1403 106  0.88867  -13.069   15.060
4     0  15.9651    6.560  2.4337 106  0.01662    2.959   28.971
```

In other cases, interest is in testing a hypothesis of a contrast $H_0 : \Lambda\beta = \beta_0$ where Λ is a matrix. For example a test of no interaction between `Month` and `Wind` can be made by testing jointly that the last four parameters in `m` are zero (observe that the test is a Wald test):

```
> Lambda

      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
[1,]    0    0    0    0    0    0    1    0    0    0
[2,]    0    0    0    0    0    0    0    1    0    0
[3,]    0    0    0    0    0    0    0    0    1    0
[4,]    0    0    0    0    0    0    0    0    0    1

> esticon(m, Lambda, joint.test = T)

X2.stat DF Pr(>|X^2|)
1   22.11  4  0.0001906
```

For a linear normal model, one would typically prefer to do a likelihood ratio test instead. However, for generalized estimating equations of glm-type (as dealt with in the packages `geepack` and `gee`) there is no likelihood. In this case `esticon` function provides an operational alternative.

Observe that another function for calculating contrasts as above is the `contrast` function in the `Design` package but it applies to a narrower range of models than `esticon` does.

11 Final remarks

The `esticon` functions and other smaller functions are likely to be removed from the `doBy` package in the future. Credit is due to Dennis Chabot, Gabor Grothendieck, Paul Murrell and Erik Jørgensen for reporting various bugs and making various suggestions to the functionality in the `doBy` package.