## Loading required package: doBy
Groupwise computations and other utilities in the doBy package

Søren Højsgaard

doBy version 4.6.20 as of 2023-11-01

Contents

1 Introduction 2

2 Data used for illustration 3

3 Groupwise computations 3
  3.1 The summaryBy and summary_by functions 3
  3.2 The orderBy function and order_by functions 4
  3.3 The splitBy and split_by functions 4
  3.4 The subsetBy and subset_by functions 6
  3.5 The transformBy and transform_by functions 6
  3.6 The lapplyBy and lapply_by function 7

4 Miscellaneous utilities 7
  4.1 The firstobs() / lastobs() functions 7
  4.2 The which.maxn() and which.minn() functions 7
  4.3 Subsequences - subSeq() 8
  4.4 Recoding values of a vector - recodeVar() 8
  4.5 Renaming columns of a dataframe or matrix - renameCol() 8
  4.6 Time since an event - timeSinceEvent() 9
  4.7 Example: Using subSeq() and timeSinceEvent() 11

5 Acknowledgements 15

1 Introduction

The doBy package contains a variety of utility functions. This working document describes some of these functions. The package originally grew out of a need to calculate groupwise summary statistics (much in the spirit of PROC SUMMARY of the SAS system), but today the package contains many different utilities.
2 Data used for illustration

The description of the doBy package is based on the mtcars dataset.

```r
> head(mtcars)
## mpg cyl disp hp drat wt qsec vs am gear carb
## Mazda RX4 21.0  6 160 110 3.90 2.620 16.46 0  1  4  4
## Mazda RX4 Wag 21.0  6 160 110 3.90 2.875 17.02 0  1  4  4
## Datsun 710 22.8  4 108  93 3.85 2.320 18.61 1  1  4  1
## Hornet 4 Drive 21.4  6 258 110 3.08 3.215 19.44 1  0  3  1
## Hornet Sportabout 18.7  8 360 175 3.15 3.440 17.02 0  0  3  2
## Valiant 18.1  6 225 105 2.76 3.460 20.22 1  0  3  1
```

A description of the variable names can be found here: [https://rstudio-pubs-static.s3.amazonaws.com/61800_faea93548c6b49cc91cd0c5ef5059894.html](https://rstudio-pubs-static.s3.amazonaws.com/61800_faea93548c6b49cc91cd0c5ef5059894.html); see also [https://www.jstor.org/stable/2529336?seq=12](https://www.jstor.org/stable/2529336?seq=12).

3 Groupwise computations

3.1 The `summaryBy` and `summary_by` functions

The `summaryBy` function is used for calculating quantities like “the mean and variance of numerical variables $x$ and $y$ for each combination of two factors $A$ and $B$”. Notice: A functionality similar to `summaryBy` is provided by `aggregate()` from base R.

```r
> myfun1 <- function(x){
  c(m=mean(x), s=sd(x))
}
> summaryBy(cbind(mpg, cyl, lh=log(hp)) ~ vs,
  data=mtcars, FUN=myfun1)
## vs mpg.m mpg.s cyl.m cyl.s lh.m lh.s
## 1 0 16.62 3.861 7.444 1.1490 5.196 0.3299
## 2 1 24.56 5.379 4.571 3.340 4.478 0.2894
```

A simpler call is

```r
> summaryBy(mpg ~ vs, data=mtcars, FUN=mean)
## vs mpg.mean
## 1 0 16.62
## 2 1 24.56
```

Instead of formula we may specify a list containing the left hand side and the right hand side
of a formula[1] but that is possible only for variables already in the dataframe:

```r
> summaryBy(list(c("mpg", "cyl"), "vs"),
  data=mtcars, FUN=myfun1)
```

```
## vs mpg.m mpg.s cyl.m cyl.s
## 1 0 16.62 3.861 7.444 1.1490
## 2 1 24.56 5.379 4.571 0.9376
```

Inspired by the **dplyr** package, there is a `summary_by` function which does the same as `summaryBy` but with the data argument being the first so that one may write

```r
> mtcars %>% summary_by(cbind(mpg, cyl, lh=log(hp)) ~ vs,
  FUN=myfun1)
```

```
## vs mpg.m mpg.s cyl.m cyl.s lh.m lh.s
## 1 0 16.62 3.861 7.444 1.1490 5.196 0.3299
## 2 1 24.56 5.379 4.571 0.9376 4.478 0.2894
```

### 3.2 The `orderBy` function and `order_by` functions

Ordering (or sorting) a data frame is possible with the `orderBy` function. For example, we order the rows according to `gear` and `carb` (within `gear`):

```r
> x1 <- orderBy(~ gear + carb, data=mtcars)
> head(x1, 4)
```

```
## mpg cyl disp hp drat wt qsec vs am gear carb
## Hornet 4 Drive 21.4 6 258.0 110 3.08 3.215 19.44 1 0 3 1
## Valiant 18.1 6 225.0 105 2.76 3.460 20.22 1 0 3 1
## Toyota Corona 21.5 4 120.1 97 3.70 2.465 20.01 1 0 3 1
## Hornet Sportabout 18.7 8 360.0 175 3.15 3.440 17.02 0 0 3 2
```

```r
> tail(x1, 4)
```

```
## mpg cyl disp hp drat wt qsec vs am gear carb
## Lotus Europa 30.4 4 95.1 113 3.77 1.513 16.9 1 1 5 2
## Ford Pantera L 15.8 8 351.0 264 4.22 3.170 14.5 0 1 5 4
## Ferrari Dino 19.7 6 145.0 175 3.62 2.770 15.5 0 1 5 6
## Maserati Bora 15.0 8 301.0 335 3.54 3.570 14.6 0 1 5 8
```

If we want the ordering to be by decreasing values of one of the variables, we can do

```r
> x2 <- orderBy(~ -gear + carb, data=mtcars)
```

Alternative forms are:

```r
> x3 <- orderBy(c("gear", "carb"), data=mtcars)
> x4 <- orderBy(~ -"gear" ~ "carb", data=mtcars)
> x5 <- mtcars %>% order_by(c("gear", "carb"))
> x6 <- mtcars %>% order_by(~ -gear + carb)
```

### 3.3 The `splitBy` and `split_by` functions

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:

---

1. This is a feature of `summaryBy` and it does not work with `aggregate`.
```r
> x <- splitBy(~ Month, data=airquality)
> x <- splitBy(~ vs, data=mtcars)
> lapply(x, head, 4)

## $'0'
##
## Mazda RX4 21.0 6 160 110 3.90 2.620 16.46 0 1 4 4
## Mazda RX4 Wag 21.0 6 160 110 3.90 2.875 17.02 0 1 4 4
## Hornet Sportabout 18.7 8 360 175 3.15 3.440 17.02 0 0 3 2
## Duster 360 14.3 8 360 245 3.21 3.570 15.84 0 0 3 4

## $'1'
##
## Datsun 710 22.8 4 108.0 93 3.85 2.320 18.61 1 1 4 1
## Hornet 4 Drive 21.4 6 258.0 110 3.08 3.215 19.44 1 0 3 1
## Valiant 18.1 6 225.0 105 2.76 3.460 20.22 1 0 3 1
## Merc 240D 24.4 4 146.7 62 3.69 3.190 20.00 1 0 4 2

> attributes(x)

## $names
## [1] "0" "1"

## $groupid
## vs
## 1 0
## 2 1

## $idxvec
## $idxvec$'0'
## [1] 1 2 5 7 12 13 14 15 16 17 22 23 24 25 29 30 31

## $idxvec$'1'
## [1] 3 4 6 8 9 10 11 18 19 20 21 26 28 32

## $grps
## [1] "0" "0" "1" "1" "0" "0" "1" "1" "0" "0" "0" "0" "1" "1" "1" "1" "1" "1" "1" "1" "1" "1"

## [22] "0" "0" "0" "0" "0" "1" "0" "0" "0" "1" "0" "0" "0" "0" "0" "0" "1"

## $class
## [1] "splitByData" "list"

Alternative forms are:

> splitBy("vs", data=mtcars)

## listentry vs
## 1 0 0
## 2 1 1

> mtcars %>% split_by(~ vs)

## listentry vs
## 1 0 0
## 2 1 1
```
3.4 The subsetBy and subset_by functions

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

```r
> x <- subsetBy(~am, subset=mpg > mean(mpg), data=mtcars)
> head(x)
```

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

Alternative forms are

```r
> x <- subsetBy("am", subset=mpg > mean(mpg), data=mtcars)
> x <- mtcars %>% subset_by("vs", subset=mpg > mean(mpg))
> x <- mtcars %>% subset_by(~vs, subset=mpg > mean(mpg))
```

3.5 The transformBy and transform_by functions

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

```r
> head(x)
```

```r
> x <- transformBy(~vs, data=mtcars, 
                   min.mpg=min(mpg), max.mpg=max(mpg))
> head(x)
```

Alternative forms:

```r
> x <- transformBy("vs", data=mtcars, 
                   min.mpg=min(mpg), max.mpg=max(mpg))
> x <- mtcars %>% transform_by("vs", 
                           min.mpg=min(mpg), max.mpg=max(mpg))
```
3.6 The `lapplyBy` and `lapply_by` function

This `lapplyBy` function is a wrapper for first splitting data into a list according to the formula (using `splitBy`) and then applying a function to each element of the list (using `lapply`).

```r
> lapplyBy(~vs, data=mtcars,
       FUN=function(d) lm(mpg~cyl, data=d) %>% summary %>% coef)
```

```
## $'0'
## Estimate Std. Error t value Pr(>|t|)
## (Intercept) 36.927  3.6908 10.005 2.728e-08
## cyl         -2.728  0.4903 -5.564 4.273e-05

## $'1'
## Estimate Std. Error t value Pr(>|t|)
## (Intercept) 41.940  5.778  7.258 1.004e-05
## cyl         -3.803  1.240 -3.066 9.782e-03
```

4 Miscellaneous utilities

4.1 The `firstobs()` / `lastobs()` functions

To obtain the indices of the first/last occurrences of an item in a vector do:

```r
> x <- c(1, 1, 2, 2, 1, 1, 1, 3)
> firstobs(x)
## [1] 1 4 10

> lastobs(x)
## [1] 6 9 10
```

The same can be done on variables in a data frame, e.g.

```r
> firstobs(~vs, data=mtcars)
## [1] 1 3

> lastobs(~vs, data=mtcars)
## [1] 31 32
```

4.2 The `which.maxn()` and `which.minn()` functions

The location of the \( n \) largest / smallest entries in a numeric vector can be obtained with

```r
> x <- c(1:4, 0:5, 11, NA, NA)
> which.maxn(x, 3)
## [1] 11 10 4

> which.minn(x, 5)
## [1] 5 1 6 2 7
```
4.3 Subsequences - subSeq()

Find (sub) sequences in a vector:

```r
> x <- c(1, 1, 2, 2, 1, 1, 3, 3, 3, 1, 1, 1)
> subSeq(x)
## first last slength midpoint value
## 1 1 2 2 2 1
## 2 3 5 3 4 2
## 3 6 7 2 7 1
## 4 8 11 4 10 3
## 5 12 14 3 13 1

> subSeq(x, item=1)
## first last slength midpoint value
## 1 1 2 2 2 1
## 2 6 7 2 7 1

> subSeq(letters[x])
## first last slength midpoint value
## 1 1 2 2 2 a
## 2 3 5 3 4 b
## 3 6 7 2 7 a
## 4 8 11 4 10 c
## 5 12 14 3 13 a

> subSeq(letters[x], item="a")
## first last slength midpoint value
## 1 1 2 2 2 a
## 2 6 7 2 7 a
```

4.4 Recoding values of a vector - recodeVar()

```r
> x <- c("dec", "jan", "feb", "mar", "apr", "may")
> src1 <- list(c("dec", "jan", "feb"), c("mar", "apr", "may"))
> tgt1 <- list("winter", "spring")
> recodeVar(x, src=src1, tgt=tgt1)
## [1] "winter" "winter" "winter" "spring" "spring" "spring"
```

4.5 Renaming columns of a dataframe or matrix – renameCol()

```r
> head(renameCol(mtcars, c("vs", "mpg"), c("vs_", "mpg_")))
## mpg_ cyl disp hp drat wt qsec vs_ am gear carb
## Mazda RX4 21.0 6 160 110 3.90 2.620 16.46 0 1 4 4
## Mazda RX4 Wag 21.0 6 160 110 3.90 2.875 17.02 0 1 4 4
## Datsun 710 22.8 4 108 93 3.85 2.320 18.61 1 0 3 1
## Hornet 4 Drive 21.4 6 258 110 3.08 3.440 17.02 0 0 3 2
## Hornet Sportabout 18.7 8 360 175 3.15 3.440 17.02 0 0 3 2
## Valiant 18.1 6 225 105 2.76 3.460 20.22 1 0 3 1
```
### 4.6 Time since an event - timeSinceEvent()

Consider the vector

```r
> yvar <- c(0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0)
```

Imagine that "1" indicates an event of some kind which takes place at a certain time point. By default time points are assumed equidistant but for illustration we define time variable

```r
> tvar <- seq_along(yvar) + c(0.1, 0.2)
```

Now we find time since event as

```r
> tse <- timeSinceEvent(yvar, tvar)
> tse
```

<table>
<thead>
<tr>
<th></th>
<th>yvar</th>
<th>tvar</th>
<th>abs.tse</th>
<th>sign.tse</th>
<th>ewin</th>
<th>run</th>
<th>tae</th>
<th>tbe</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>1.1</td>
<td>3.1</td>
<td>-3.1</td>
<td>1</td>
<td>NA</td>
<td>NA</td>
<td>-3.1</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>2.0</td>
<td>2.0</td>
<td>-2.0</td>
<td>1</td>
<td>NA</td>
<td>NA</td>
<td>-2.0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>1.1</td>
<td>1.1</td>
<td>-1.1</td>
<td>1</td>
<td>NA</td>
<td>NA</td>
<td>-1.1</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>4.2</td>
<td>0.0</td>
<td>0.0</td>
<td>1</td>
<td>1.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>5.1</td>
<td>0.9</td>
<td>0.9</td>
<td>1</td>
<td>1.0</td>
<td>0.9</td>
<td>-5.1</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>6.2</td>
<td>2.0</td>
<td>2.0</td>
<td>1</td>
<td>1.2</td>
<td>0.0</td>
<td>-4.0</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>7.1</td>
<td>2.9</td>
<td>2.9</td>
<td>1</td>
<td>1.2</td>
<td>0.9</td>
<td>-3.1</td>
</tr>
<tr>
<td>8</td>
<td>0</td>
<td>8.2</td>
<td>2.0</td>
<td>-2.0</td>
<td>2</td>
<td>1.4</td>
<td>0.0</td>
<td>-2.0</td>
</tr>
<tr>
<td>9</td>
<td>0</td>
<td>9.1</td>
<td>1.1</td>
<td>-1.1</td>
<td>2</td>
<td>1.4</td>
<td>-1.1</td>
<td>-1.1</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
<td>10.2</td>
<td>0.0</td>
<td>0.0</td>
<td>2</td>
<td>2.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>11</td>
<td>0</td>
<td>11.1</td>
<td>0.9</td>
<td>0.9</td>
<td>2</td>
<td>2.0</td>
<td>-3.1</td>
<td>-3.1</td>
</tr>
<tr>
<td>12</td>
<td>0</td>
<td>12.2</td>
<td>2.0</td>
<td>2.0</td>
<td>2</td>
<td>2.2</td>
<td>0.0</td>
<td>-2.0</td>
</tr>
<tr>
<td>13</td>
<td>0</td>
<td>13.1</td>
<td>1.1</td>
<td>-1.1</td>
<td>3</td>
<td>2.2</td>
<td>-1.1</td>
<td>-1.1</td>
</tr>
<tr>
<td>14</td>
<td>1</td>
<td>14.2</td>
<td>0.0</td>
<td>0.0</td>
<td>3</td>
<td>3.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>15</td>
<td>1</td>
<td>15.1</td>
<td>0.0</td>
<td>0.0</td>
<td>4</td>
<td>4.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>16</td>
<td>0</td>
<td>16.2</td>
<td>1.1</td>
<td>1.1</td>
<td>4</td>
<td>4.1</td>
<td>NA</td>
<td>NA</td>
</tr>
<tr>
<td>17</td>
<td>0</td>
<td>17.1</td>
<td>2.0</td>
<td>2.0</td>
<td>4</td>
<td>4.2</td>
<td>NA</td>
<td>NA</td>
</tr>
<tr>
<td>18</td>
<td>0</td>
<td>18.2</td>
<td>3.1</td>
<td>3.1</td>
<td>4</td>
<td>4.3</td>
<td>NA</td>
<td>NA</td>
</tr>
<tr>
<td>19</td>
<td>0</td>
<td>19.1</td>
<td>4.0</td>
<td>4.0</td>
<td>4</td>
<td>4.4</td>
<td>NA</td>
<td>NA</td>
</tr>
<tr>
<td>20</td>
<td>0</td>
<td>20.2</td>
<td>5.1</td>
<td>5.1</td>
<td>4</td>
<td>5.1</td>
<td>NA</td>
<td>NA</td>
</tr>
</tbody>
</table>

The output reads as follows:

- **abs.tse**: Absolute time since (nearest) event.
- **sign.tse**: Signed time since (nearest) event.
- **ewin**: Event window: Gives a symmetric window around each event.
- **run**: The value of run is set to 1 when the first event occurs and is increased by 1 at each subsequent event.
- **tae**: Time after event.
- **tbe**: Time before event.

```r
> plot(sign.tse ~ tvar, data=tse, type="b")
> grid()
> rug(tse$yvar[tse$yvar == 1], col="blue", lwd=4)
> points(scale(tse$run), col=tse$run, lwd=2)
> lines(abs.tse + .2 ~ tvar, data=tse, type="b", col=3)
```
> plot(tae ~ tvar, data=tse, ylim=c(-6,6), type="b")
> grid()
> lines(tbe ~ tvar, data=tse, type="b", col="red")
> rug(tse$tvar[tse$yvar==1], col="blue", lwd=4)
> lines(run ~ tvar, data=tse, col="cyan", lwd=2)

> plot(ewin ~ tvar, data=tse, ylim=c(1,4))
> rug(tse$tvar[tse$yvar==1], col="blue", lwd=4)
We may now find times for which time since an event is at most 1 as

```r
> tse$tvar[tse$abs <= 1]
```

```
## [1] 4.2 5.1 10.2 11.1 14.2 15.1
```

### 4.7 Example: Using `subSeq()` and `timeSinceEvent()`

Consider the lynx data:

```r
> lynx <- as.numeric(lynx)
> tvar <- 1821:1934
> plot(tvar, lynx, type="l")
```
Suppose we want to estimate the cycle lengths. One way of doing this is as follows:

```r
> yyy <- lynx > mean(lynx)
> head(yyy)
## [1] FALSE FALSE FALSE FALSE FALSE TRUE
> sss <- subSeq(yyy, TRUE)
> sss
## first last slength midpoint value
## 1 6 10 5 8 TRUE
## 2 16 19 4 18 TRUE
## 3 27 28 2 28 TRUE
## 4 35 38 4 37 TRUE
## 5 44 47 4 46 TRUE
## 6 53 55 3 54 TRUE
## 7 63 66 4 65 TRUE
## 8 75 76 2 76 TRUE
## 9 83 87 5 85 TRUE
## 10 92 96 5 94 TRUE
## 11 104 106 3 105 TRUE
## 12 112 114 3 113 TRUE
```

```r
> plot(tvar, lynx, type="l")
> rug(tvar[sss$midpoint], col="blue", lwd=4)
```
Create the "event vector"

```r
> yvar <- rep(0, length(lynx))
> yvar[sse$midpoint] <- 1
> str(yvar)
```

```
## num [1:114] 0 0 0 0 0 0 0 1 0 0 ...
```

```r
> tse <- timeSinceEvent(yvar, tvar)
> head(tse, 20)
```

<table>
<thead>
<tr>
<th></th>
<th>yvar</th>
<th>tvar</th>
<th>abs.tse</th>
<th>sign.tse</th>
<th>ewin</th>
<th>run</th>
<th>tae</th>
<th>tbe</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>1821</td>
<td>7</td>
<td>-7</td>
<td>1</td>
<td>NA</td>
<td>NA</td>
<td>-7</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>1822</td>
<td>6</td>
<td>-6</td>
<td>1</td>
<td>NA</td>
<td>NA</td>
<td>-6</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>1823</td>
<td>5</td>
<td>-5</td>
<td>1</td>
<td>NA</td>
<td>NA</td>
<td>-5</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>1824</td>
<td>4</td>
<td>-4</td>
<td>1</td>
<td>NA</td>
<td>NA</td>
<td>-4</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>1825</td>
<td>3</td>
<td>-3</td>
<td>1</td>
<td>NA</td>
<td>NA</td>
<td>-3</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>1826</td>
<td>2</td>
<td>-2</td>
<td>1</td>
<td>NA</td>
<td>NA</td>
<td>-2</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>1827</td>
<td>1</td>
<td>-1</td>
<td>1</td>
<td>NA</td>
<td>NA</td>
<td>-1</td>
</tr>
<tr>
<td>8</td>
<td>1</td>
<td>1828</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>9</td>
<td>0</td>
<td>1829</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>-9</td>
</tr>
<tr>
<td>10</td>
<td>0</td>
<td>1830</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>-8</td>
</tr>
<tr>
<td>11</td>
<td>0</td>
<td>1831</td>
<td>3</td>
<td>3</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>-7</td>
</tr>
<tr>
<td>12</td>
<td>0</td>
<td>1832</td>
<td>4</td>
<td>4</td>
<td>1</td>
<td>4</td>
<td>1</td>
<td>-6</td>
</tr>
<tr>
<td>13</td>
<td>0</td>
<td>1833</td>
<td>5</td>
<td>5</td>
<td>1</td>
<td>5</td>
<td>1</td>
<td>-5</td>
</tr>
<tr>
<td>14</td>
<td>0</td>
<td>1834</td>
<td>4</td>
<td>-4</td>
<td>2</td>
<td>1</td>
<td>6</td>
<td>-4</td>
</tr>
<tr>
<td>15</td>
<td>0</td>
<td>1835</td>
<td>3</td>
<td>-3</td>
<td>2</td>
<td>1</td>
<td>7</td>
<td>-3</td>
</tr>
<tr>
<td>16</td>
<td>0</td>
<td>1836</td>
<td>2</td>
<td>-2</td>
<td>2</td>
<td>1</td>
<td>8</td>
<td>-2</td>
</tr>
<tr>
<td>17</td>
<td>0</td>
<td>1837</td>
<td>1</td>
<td>-1</td>
<td>2</td>
<td>1</td>
<td>9</td>
<td>-1</td>
</tr>
<tr>
<td>18</td>
<td>1</td>
<td>1838</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>19</td>
<td>0</td>
<td>1839</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>-9</td>
</tr>
<tr>
<td>20</td>
<td>0</td>
<td>1840</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>-8</td>
</tr>
</tbody>
</table>
```

We get two different (not that different) estimates of period lengths:
> len1 <- tapply(tse$ewin, tse$ewin, length)
> len2 <- tapply(tse$run, tse$run, length)
> c(median(len1), median(len2), mean(len1), mean(len2))


We can overlay the cycles as:

> tse$lynx <- lynx
> tse2 <- na.omit(tse)
> plot(lynx ~ tae, data=tse2)

![Graph showing lynx vs. tae](image)

> plot(tvar, lynx, type="l", lty=2)
> mm <- lm(lynx ~ tae + I(tae^2) + I(tae^3), data=tse2)
> lines(fitted(mm) ~ tvar, data=tse2, col="red")
5 Acknowledgements

Credit is due to Dennis Chabot, Gabor Grothendieck, Paul Murrell and Jim Robison-Cox for reporting various bugs and making various suggestions to the functionality in the doBy package.