jumbled demonstrations

Sasha D. Hafner

08 October, 2022

Overview

This document demonstrates usage of some of the function in the jumbled repo, available from github.com/sashahafner/jumbled.

Load functions

```
ff <- list.files(pattern = '\\.R$')
for(i in ff) source(i)
```

aggregate2

A wrapper for `aggregate` that accepts multiple functions and simpler arguments. Does not accept formula notation.

Example from `aggregate` help file:

```
aggregate(breaks ~ wool + tension, data = warpbreaks, mean)
```

```
##   wool tension   breaks
## 1    A       L 44.55556
## 2    B       L 28.22222
## 3    A       M 24.00000
## 4    B       M 28.77778
## 5    A       H 24.55556
## 6    B       H 18.77778
```

To include sd and n, use `aggregate2`:

```
aggregate2(warpbreaks, x = 'breaks', by = c('wool', 'tension'),
           FUN = list(mean = mean, sd = sd, n = length))
```

```
##   wool tension breaks.mean breaks.sd breaks.n
## 1    A       L    44.55556 18.097729        9
## 2    B       L    28.22222  9.858724        9
## 3    A       M    24.00000  8.660254        9
## 4    B       M    28.77778  9.431036        9
## 5    A       H    24.55556 10.272671        9
## 6    B       H    18.77778  4.893306        9
```

Accepts multiple variables (as in `aggregate`).

```
aggregate2(na.omit(airquality), x = c('Ozone', 'Temp'), by = 'Month',
       FUN = list(mean = mean, sd = sd, n = length))
```

```
##   Month Ozone.mean Temp.mean Ozone.sd  Temp.sd Ozone.n Temp.n
## 1     5   24.12500  66.45833 22.88594 6.633113      24     24
## 2     6   29.44444  78.22222 18.20790 7.838651       9      9
## 3     7   59.11538  83.88462 31.63584 4.439161      26     26
## 4     8   60.00000  83.69565 41.76776 7.054559      23     23
## 5     9   31.44828  76.89655 24.14182 8.503549      29     29
```

aggregate3

Similar, but uses formula notation. Example from `aggregate` help file:

```
aggregate(breaks ~ wool + tension, data = warpbreaks, mean)
```

```
##   wool tension   breaks
## 1    A       L 44.55556
## 2    B       L 28.22222
## 3    A       M 24.00000
## 4    B       M 28.77778
## 5    A       H 24.55556
## 6    B       H 18.77778
```

To include sd and n, use `aggregate3`:

```
aggregate3(warpbreaks, breaks ~ wool + tension,
       FUN = list(mean = mean, sd = sd, n = length))
```

```
##   wool tension breaks.mean breaks.sd breaks.n
## 1    A       L    44.55556 18.097729        9
## 2    B       L    28.22222  9.858724        9
## 3    A       M    24.00000  8.660254        9
## 4    B       M    28.77778  9.431036        9
## 5    A       H    24.55556 10.272671        9
## 6    B       H    18.77778  4.893306        9
```

For multiple response variables, use `cbind()`.

```
aggregate3(airquality, cbind(Ozone, Temp) ~ Month,
       FUN = list(mean = mean, sd = sd, n = length))
```

```
##   Month Ozone.mean Temp.mean Ozone.sd  Temp.sd Ozone.n Temp.n
## 1     5   23.61538  66.73077 22.22445 6.533346      26     26
## 2     6   29.44444  78.22222 18.20790 7.838651       9      9
## 3     7   59.11538  83.88462 31.63584 4.439161      26     26
## 4     8   59.96154  83.96154 39.68121 6.666218      26     26
## 5     9   31.44828  76.89655 24.14182 8.503549      29     29
```

So `Ozone + Temp ~ Month` doesn't work, because `aggregate()` can't handle it propertly. It would be nice to address this limitation in the future.

dfcombos

Something like `expand.grid` for data frames. Can accept vectors too, but resulting name is poor.

```
d1 <- data.frame(name = letters[1:5], x = 1.1)
d2 <- data.frame(b = 1:3)
dfcombos(d1, d2)
```

```
##    name   x b
## 1     a 1.1 1
## 2     b 1.1 1
## 3     c 1.1 1
## 4     d 1.1 1
## 5     e 1.1 1
## 6     a 1.1 2
## 7     b 1.1 2
## 8     c 1.1 2
## 9     d 1.1 2
## 10    e 1.1 2
## 11    a 1.1 3
## 12    b 1.1 3
## 13    c 1.1 3
## 14    d 1.1 3
## 15    e 1.1 3
```

```
v1 <- c(TRUE, FALSE)
dfcombos(d1, d2, v1)
```

```
##    name   x b X[[i]]
## 1     a 1.1 1   TRUE
## 2     b 1.1 1   TRUE
## 3     c 1.1 1   TRUE
## 4     d 1.1 1   TRUE
## 5     e 1.1 1   TRUE
## 6     a 1.1 2   TRUE
## 7     b 1.1 2   TRUE
## 8     c 1.1 2   TRUE
## 9     d 1.1 2   TRUE
## 10    e 1.1 2   TRUE
## 11    a 1.1 3   TRUE
## 12    b 1.1 3   TRUE
## 13    c 1.1 3   TRUE
## 14    d 1.1 3   TRUE
## 15    e 1.1 3   TRUE
## 16    a 1.1 1  FALSE
## 17    b 1.1 1  FALSE
## 18    c 1.1 1  FALSE
## 19    d 1.1 1  FALSE
## 20    e 1.1 1  FALSE
## 21    a 1.1 2  FALSE
## 22    b 1.1 2  FALSE
## 23    c 1.1 2  FALSE
## 24    d 1.1 2  FALSE
## 25    e 1.1 2  FALSE
## 26    a 1.1 3  FALSE
```

```
## 27    b 1.1 3  FALSE
## 28    c 1.1 3  FALSE
## 29    d 1.1 3  FALSE
## 30    e 1.1 3  FALSE
```

dfsumm

Generate a data frame summary more detailed and compact than `summary` output.

```
dfsumm(attenu)
```

```
##
##  182 rows and 5 columns
##  182 unique rows
##                       event     mag station    dist   accel
## Class              numeric numeric  factor numeric numeric
## Minimum                  1       5    1008     0.5   0.003
## Maximum                 23     7.7    c266     370    0.81
## Mean                  14.7    6.08     262    45.6   0.154
## Unique (excld. NA)      23      17     117     153     120
## Missing values           0       0      16       0       0
## Sorted                TRUE   FALSE   FALSE   FALSE   FALSE
##
```

Compare to `summary`.

```
summary(attenu)
```

```
##      event             mag           station         dist
##  Min.   : 1.00   Min.   :5.000   117    :  5   Min.   :  0.50
##  1st Qu.: 9.00   1st Qu.:5.300   1028   :  4   1st Qu.: 11.32
##  Median :18.00   Median :6.100   113    :  4   Median : 23.40
##  Mean   :14.74   Mean   :6.084   112    :  3   Mean   : 45.60
##  3rd Qu.:20.00   3rd Qu.:6.600   135    :  3   3rd Qu.: 47.55
##  Max.   :23.00   Max.   :7.700   (Other):147   Max.   :370.00
##                                  NA's   : 16
##      accel
##  Min.   :0.00300
##  1st Qu.:0.04425
##  Median :0.11300
##  Mean   :0.15422
##  3rd Qu.:0.21925
##  Max.   :0.81000
##
```

interpm

Fill in missing observations for multiple columns via interpolation. `interpm` calls `approx`.

```
args(interpm)
```

```
## function (dat, x, ys, ...)
## NULL
```

```r
dat <- data.frame(time = 1:30, a = rnorm(30), b = rnorm(30), c = rnorm(30))
dat[5:10, -1] <- NA
dat[20:22, 'a'] <- NA

dat
```

```
##    time           a           b           c
## 1     1  0.01218747 -0.39452395 -0.44552729
## 2     2 -1.23309627  0.63425867 -1.78419366
## 3     3  0.26609512 -1.64153233  0.12041650
## 4     4  1.80050151 -0.28978956  0.17306897
## 5     5          NA          NA          NA
## 6     6          NA          NA          NA
## 7     7          NA          NA          NA
## 8     8          NA          NA          NA
## 9     9          NA          NA          NA
## 10   10          NA          NA          NA
## 11   11 -1.76939941  0.97293572 -0.17817514
## 12   12 -0.84445818 -0.04980962  0.94265301
## 13   13 -0.29326615  1.91161621  0.38331349
## 14   14  0.24095052  0.47688130 -0.59591028
## 15   15  0.47602663  0.76299572  0.38513356
## 16   16 -0.43695264  0.59217750 -0.12409123
## 17   17  0.48998380 -0.15440158 -0.52545430
## 18   18  0.24642412  0.49221836  0.93149095
## 19   19  0.97602882 -0.39469701 -0.86101581
## 20   20          NA -0.81219925 -2.25629341
## 21   21          NA  0.13443791  1.59219585
## 22   22          NA  0.67851195  0.05806476
## 23   23 -0.00241943  1.20236269  0.19811188
## 24   24  0.05492089 -1.21902276  0.25186819
## 25   25  0.71657141  0.35530595  0.02993855
## 26   26  0.59101163 -0.73463517 -0.50306126
## 27   27 -1.69192080 -1.85840100  1.16794496
## 28   28  1.31146264 -0.75432699  0.51092477
## 29   29  0.14102226  1.02738432  0.35487687
## 30   30  0.37780403  0.17711964 -0.02487856
```

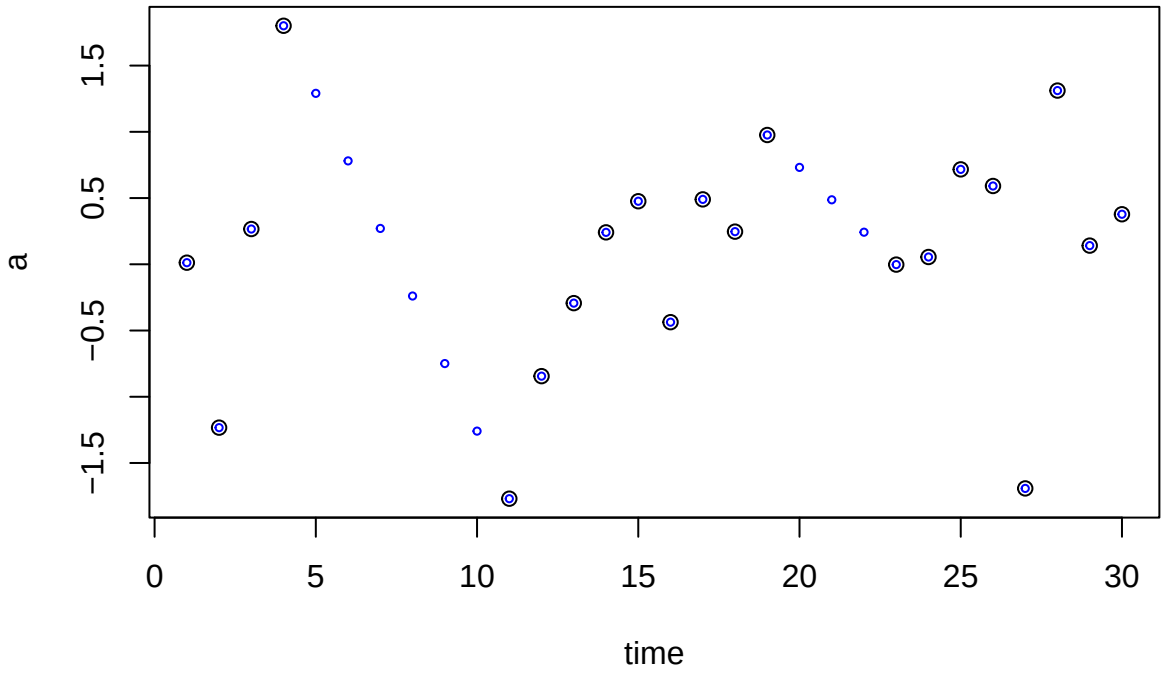
```r
dat2 <- interpm(dat, 'time', c('a', 'b', 'c'))

dat2
```

```
##    time           a           b           c
## 1     1  0.01218747 -0.39452395 -0.44552729
## 2     2 -1.23309627  0.63425867 -1.78419366
## 3     3  0.26609512 -1.64153233  0.12041650
## 4     4  1.80050151 -0.28978956  0.17306897
## 5     5  1.29051566 -0.10940023  0.12289124
## 6     6  0.78052982  0.07098909  0.07271351
## 7     7  0.27054397  0.25137842  0.02253578
## 8     8 -0.23944187  0.43176774 -0.02764195
## 9     9 -0.74942772  0.61215707 -0.07781968
## 10   10 -1.25941356  0.79254639 -0.12799741
## 11   11 -1.76939941  0.97293572 -0.17817514
## 12   12 -0.84445818 -0.04980962  0.94265301
```

```
## 13   13 -0.29326615  1.91161621  0.38331349
## 14   14  0.24095052  0.47688130 -0.59591028
## 15   15  0.47602663  0.76299572  0.38513356
## 16   16 -0.43695264  0.59217750 -0.12409123
## 17   17  0.48998380 -0.15440158 -0.52545430
## 18   18  0.24642412  0.49221836  0.93149095
## 19   19  0.97602882 -0.39469701 -0.86101581
## 20   20  0.73141676 -0.81219925 -2.25629341
## 21   21  0.48680470  0.13443791  1.59219585
## 22   22  0.24219263  0.67851195  0.05806476
## 23   23 -0.00241943  1.20236269  0.19811188
## 24   24  0.05492089 -1.21902276  0.25186819
## 25   25  0.71657141  0.35530595  0.02993855
## 26   26  0.59101163 -0.73463517 -0.50306126
## 27   27 -1.69192080 -1.85840100  1.16794496
## 28   28  1.31146264 -0.75432699  0.51092477
## 29   29  0.14102226  1.02738432  0.35487687
## 30   30  0.37780403  0.17711964 -0.02487856
```

```
plot(a ~ time, data = dat)
points(a ~ time, data = dat2, cex = 0.5, col = 'blue')
```



logaxis

Add log axis to base R plots.

logistic

The logistic function for transformations.

rbindf

Like `rbind` but data frame columns do not need to match. From monitoR package.

rounddf

Round complete data frames.

```
dat <- data.frame(a = 1:10, b = rnorm(10), c = letters[1:10])
dat
```

```
##     a          b c
## 1   1  1.9009247 a
## 2   2  0.5389815 b
## 3   3 -0.6012083 c
## 4   4  0.5160278 d
## 5   5  0.5823485 e
## 6   6  0.7041148 f
## 7   7 -0.9165045 g
## 8   8  0.4734671 h
## 9   9  0.2850358 i
## 10 10 -0.5689634 j
```

```
rounddf(dat)
```

```
##     a     b c
## 1   1  1.90 a
## 2   2  0.54 b
## 3   3 -0.60 c
## 4   4  0.52 d
## 5   5  0.58 e
## 6   6  0.70 f
## 7   7 -0.92 g
## 8   8  0.47 h
## 9   9  0.29 i
## 10 10 -0.57 j
```

```
rounddf(dat, digits = c(0, 4))
```

```
## Warning in rounddf(dat, digits = c(0, 4)): First value in digits repeated to
## match length.
```

```
##     a       b c
## 1   1  1.9009 a
## 2   2  0.5390 b
## 3   3 -0.6012 c
## 4   4  0.5160 d
## 5   5  0.5823 e
## 6   6  0.7041 f
## 7   7 -0.9165 g
## 8   8  0.4735 h
## 9   9  0.2850 i
## 10 10 -0.5690 j
```

```
rounddf(dat, digits = c(0, 4), func = signif)
```

```
## Warning in rounddf(dat, digits = c(0, 4), func = signif): First value in digits
## repeated to match length.
```

```
##     a       b c
## 1   1  1.9010 a
## 2   2  0.5390 b
## 3   3 -0.6012 c
## 4   4  0.5160 d
## 5   5  0.5823 e
## 6   6  0.7041 f
## 7   7 -0.9165 g
## 8   8  0.4735 h
## 9   9  0.2850 i
## 10 10 -0.5690 j
```

```
rounddf(dat, digits = c(2, 2), func = signif)
```

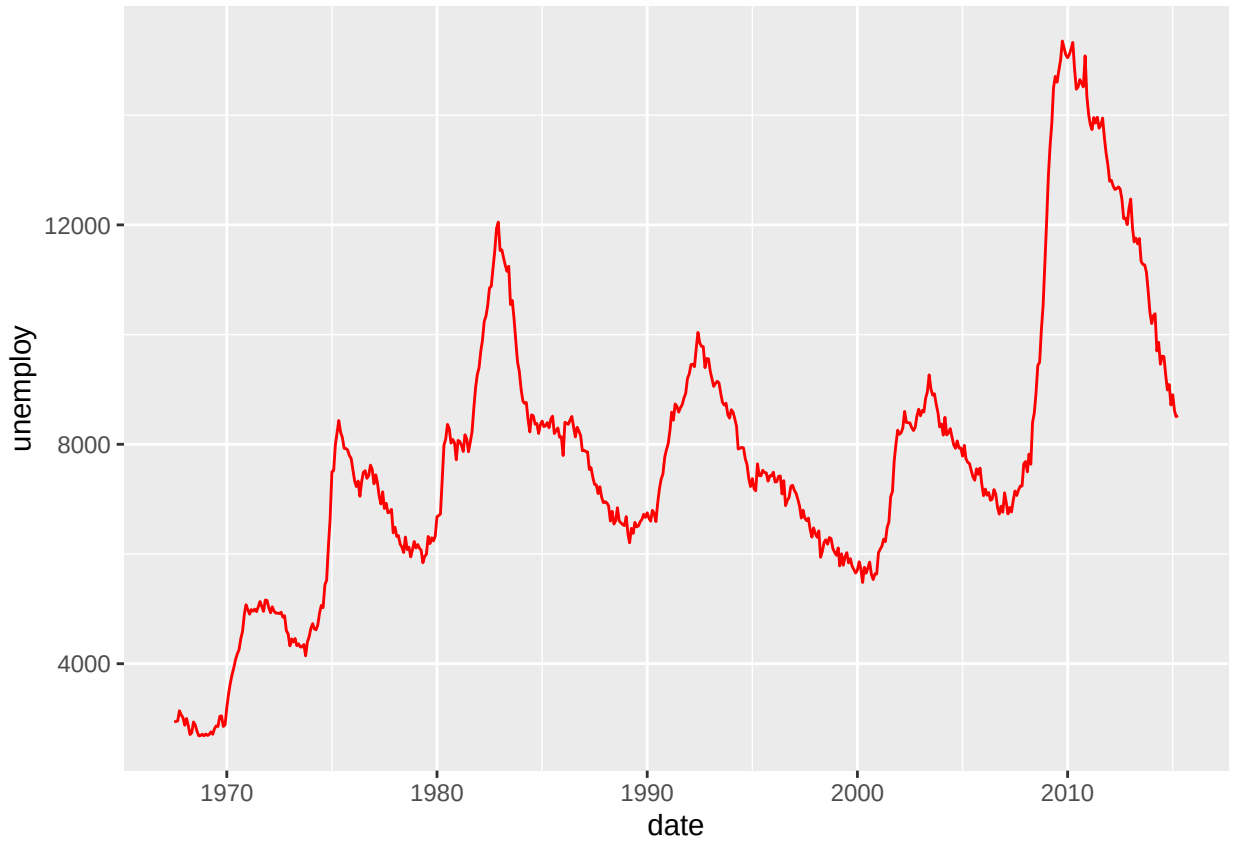
```
## Warning in rounddf(dat, digits = c(2, 2), func = signif): First value in digits
## repeated to match length.
```

```
##     a     b c
## 1   1  1.90 a
## 2   2  0.54 b
## 3   3 -0.60 c
## 4   4  0.52 d
## 5   5  0.58 e
## 6   6  0.70 f
## 7   7 -0.92 g
## 8   8  0.47 h
## 9   9  0.29 i
## 10 10 -0.57 j
```

ggsave2x

Save a ggplot2 figure in more than one format in a single call.

```
library(ggplot2)
ggplot(economics, aes(date, unemploy)) +
  geom_line(colour = "red")
```




```
ggsave2x('economics', width = 5, height = 5)
```

Saves png and pdf by default, add more with `type` argument. Use `...` optional arguments for more flexibility.