R Package `miscset` User Manual

Sven E. Templer
sven.templer@gmail.com

March 27, 2015

Contents

1 Introduction

miscset

A GNU R package with miscellaneous tools.

The package contains methods to simplify workspace handling, sort, reshape and apply functions on grids of data.frames, scale numeric values, extract unique and duplicate values, perform regular expression based string operations and ease plotting. Many methods are implemented with multi-core support from the parallel package or written in C++ interfaced with the Rcpp library.

2 Installation & Development

You can install the package to use it in any `R` session. The stable versions are published on the comprehensive R archive network (CRAN) servers. The web description of this and previous versions can be found on the page cran.r-project.org/web/packages/miscset. The installation takes place in an `R` session or script. Run the code:

```
install.packages("miscset")
```

A more administrative way is to use the command line in a shell environment and install the package from a downloaded `.tar.gz` archive. Invoking `R` from the shell provides the syntax `R CMD INSTALL /path/to/package.tar.gz` therefore.

The most recent version can be found on github.com/setempler/miscset. With the `devtools` package, it is easy to install it:

```
install.packages("devtools")
library(devtools)
install_github(repo='setempler/miscset')
```

Once installed, you need to load the package into the current session to access the functions and help pages. With the functions `library` or `require` you perform this step:

```
library(miscset)
```

In the following sections the usage of the functions is shown with example data and the output. Each function is accomanied by a help page, which is typically invoked on the `R` shell with `?` in front of the function's name.

3 Numeric Functions

3.1 Generate triangular numbers - `ntri`

The function generates a series of triangular numbers of length `n` according to oeis.org/A000217. The series for 12 rows of a triangle, for example, can be returned as in the following example:

```
ntri(12)

##  [1]  0  1  3  6 10 15 21 28 36 45 55 66
```

The function takes one argument `n` as in the syntax `ntri(n)`, which can be a positive integer value for the length of the sequence.

3.2 Scale numeric vectors - `scale0, scaler`

The function `scale0` scales all values in a numeric vector from 0 to 1 according to their minimum and maximum values. It allows numeric vector input for the argument `x` as in it's usage:

```
scale0(x)
```

By it's default argument, `scaler` behaves the same as `scale0`. It offers two more arguments to modify the range (`r`) of the output and borders (`b`) of the input as in:

```
scaler(x, r = c(0, 1), b = range(x, na.rm = TRUE))
```

See the following examples for the effect:

```
scale0(0:5)

## [1] 0.0 0.2 0.4 0.6 0.8 1.0

scale0(-2:3)

## [1] 0.0 0.2 0.4 0.6 0.8 1.0

scaler(0:5)

## [1] 0.0 0.2 0.4 0.6 0.8 1.0

scaler(0:5, r = c(1, 9.9))

## [1] 1.00 2.78 4.56 6.34 8.12 9.90

scaler(0:5, b = c(0, 10))

## [1] 0.0 0.1 0.2 0.3 0.4 0.5
```

3.3 p values to symbol (***) converting - `p2star`

The function `p2star` converts numeric values to character symbol representing the range in which the value resides. This feature is provided in `R` by the function `symnum`. The default settings in `p2star` are setting breaks for the ranges useful to convert p-values from statistical tests to significance indicators:

```
p2star(p, breaks = c(0, 0.001, 0.01, 0.05, 0.1, 1), symbols = c("***", "**", "*", ".",
"n.s."))
```

The argument `symbols` can be modified to change the output symbols. The two lowest values in `breaks` define the range for the first value in `symbol`. See the following example:

```
p2star(c(1e-5,.1,.9))

## [1] "***"  "."    "n.s."
```

4 Data Manipulation

4.1 Sort a data frame - `sort`

The base function `sort` provides a way to sort data vectors in increasing or decreasing order. Despite being implemented as S3 generic, it does not contain a lot of methods specific for special data types. Therefore, to sort data arranged in a `data.frame` class object, one needs to use the `order` function, and create complex function calls, for example:

```
d <- data.frame(a=c(1,1,1,2,NA),b=c(2,1,3,1,1),c=5:1)
d

##    a b c
## 1  1 2 5
## 2  1 1 4
## 3  1 3 3
## 4  2 1 2
## 5 NA 1 1

# classical ordering of data.frame objects
d[order(d$b, d$c),]

##    a b c
## 5 NA 1 1
## 4  2 1 2
## 2  1 1 4
## 1  1 2 5
## 3  1 3 3
```

In the package `miscset` the method for this class is implemented (as `sort.data.frame`) and can be called with the generic name on `data.frame` objects. The order and set of the columns to sort by can be provided as character vector or expression with the `by` or `bye` arguments. Increasing or decreasing order can be chosen with the `decreasing` argument (right now only one direction is possible for all chosen columns.)

```
# sort by every column, decreasing
sort(d, decreasing = TRUE)

##   a b c
## 4 2 1 2
## 3 1 3 3
## 1 1 2 5
## 2 1 1 4

# ... by column 'c'
sort(d, by="c")

##    a b c
## 5 NA 1 1
## 4  2 1 2
## 3  1 3 3
## 2  1 1 4
## 1  1 2 5

# ... by columns 'a' and then 'c'
sort(d, bye=.(a,c))

##   a b c
## 3 1 3 3
## 2 1 1 4
## 1 1 2 5
## 4 2 1 2
```

Rows with missing values, if present in one of the columns to sort by, will be dropped with the default setting of the argument `na.last = NA`. To keep rows with missing values on top or bottom of the `data.frame`, use `FALSE` or `TRUE` as value for the argument, respectively.

4.2 Row-bind data.frames - `do.rbind`

...

4.3 Duplicates and unique values - `duplicates, duplicatei, uniquei, nunique`

...

4.4 Create a factor containing missing values as level - `factorNA`

...

4.5 Apply a function on a data.frame by a grid - `gapply`

To apply a function on a subset of a dataset, all named columns are used to create a grid for which each unique combination is used to extract the rows in the `data.frame`. Multicore support is implemented by `mclapply`. The grid can be extracted by the function `levels` and a row binding of elements that can be coerced to data.frames is implemented in the method `as.data.frame`. If a `data.table` is preferred, the method `as.data.table` is implemented and works only if the package is installed.

```
f <- function (x) c(conc.diff = diff(range(x$conc)), uptake.sum=sum(x$uptake))
d <- gapply(CO2, c('Type', 'Treatment'), f)
levels(d)

##          Type  Treatment
## 1      Quebec nonchilled
## 2 Mississippi nonchilled
## 3      Quebec    chilled
## 4 Mississippi    chilled

head(as.data.frame(d))

##   conc.diff uptake.sum        Type  Treatment
## 1       905      742.0      Quebec nonchilled
## 2       905      545.0 Mississippi nonchilled
## 3       905      666.8      Quebec    chilled
## 4       905      332.1 Mississippi    chilled
```

4.6 Transform to squared matrix - `squarematrix`

The function `squarematrix` can generate a symmetric (square) matrix from an unsymmetric matrix by using the column and row names and filling empty pairs with `NA`.

```
M <- matrix(1:6, 2, dimnames = list(2:3,1:3))
M

##   1 2 3
## 2 1 3 5
## 3 2 4 6

squarematrix(M)

##    1  2  3
## 1 NA NA NA
## 2  1  3  5
## 3  2  4  6
```

4.7 Generate a pairwise list - `enpaire`

The function **enpaire** creates a pairwise list of matrix values. The result is a **data.frame** that containsa column for the names of each dimension and the upper and lower triangle values. Unsymmetric matrices are transformed by **squarematrix** (see previous section).

```
M <- matrix(letters[1:9], 3, 3, dimnames = list(1:3,1:3))
M

##   1   2   3
## 1 "a" "d" "g"
## 2 "b" "e" "h"
## 3 "c" "f" "i"

enpaire(M)

##   row col lower upper
## 1   1   2     b     d
## 2   1   3     c     g
## 3   2   3     f     h
```

4.8 Create a latex document containing a table - `textable`

This function enhances the functionality of the **xtable** function from the similar named package. The output of **xtable** is captured, processed and then written to a file. The file may contain also latex header for an A4 portrait or landscape article. The function is called with the following syntax: **textable(d, file, caption = NULL, label = NULL, align = NULL, rownames = FALSE, topcapt = TRUE, digits = NULL, as.document = FALSE, landscape = FALSE, margin = 2, pt.size = 10, cmd = NULL)**

file is a character string with the name to the file of the function output. **caption** is a character string with the table's title. It is aligned to the top of the table, when **topcapt** is **TRUE**, otherwise to the table bottom. **rownames** is logical and allows to switch printing of row names on and off. when **as.document** is **TRUE**, a document header is added. Then, **landscape** defines the orientation of the page, **pt.size** the size of the characters and **margin** the table borders in cm. **digits** sets the number of digits to print for numeric values. With **align** the column alignements can be set. It is either one of **'r', 'c', 'l'**, or a vector of those elements for alignemt to the right, center or left, repsectively (use **rep()** or **strplit()** as support). **label** allows to supply a latex label for reference in form of a vector of length one.

```
textable(head(trees,3), rownames = TRUE, digits=4, align=strsplit("llrr","")[[1]],
         as.document = TRUE, label='tab:trees', caption='R dataset "trees".')

## % output by function 'textable' from package miscset 0.6
## % latex table generated in R 3.1.3 by xtable 1.7-4 package
## % Fri Mar 27 15:17:06 2015
##
## \documentclass[a4paper,10pt]{article}
## \usepackage[a4paper,margin=2cm]{geometry}
## \begin{document}
##
## \begin{table}[ht]
## \centering
## \caption{R dataset "trees".}
## \begin{tabular}{llrr}
##   \hline
##  & Girth & Height & Volume \\
##   \hline
## 1 & 8.3000 & 70.0000 & 10.3000 \\
##   2 & 8.6000 & 65.0000 & 10.3000 \\
```

```
##   3 & 8.8000 & 63.0000 & 10.2000 \\
##    \hline
## \end{tabular}
## \label{tab:trees}
## \end{table}
##
## \end{document}
```

In addition, a system command can be provided as a character string with `cmd` to create a pdf for example. An example therefore might be `cmd = "pdflatex"`.

5 Workspace and System

5.1 List details from and remove all objects - `lsall, rmall`

With `lsall(envir, ...)` all object names, their length, class, mode and size is returned in a data.frame from a specified environment. `rmall(...)` removes the complete list of objects at the global environment.

```
lsall()

## Environment: R_GlobalEnv
## Objects:
##   Name Length    Class      Mode Size Unit
## 1    d      4   gapply      list  4.9   Kb
## 2    f      1 function  function  2.5   Kb
## 3    M      9   matrix character  1.3   Kb

rmall()
lsall()

## Environment: R_GlobalEnv
## Objects:
## NULL
```

5.2 Load objects from R data files into a list - `lload`

`lload` provides a way to load R objects from multiple R data files and stores all objects in a list. Thereby, the list names are respective to the R data files. Each entry consists of a sublist with as many entries as objects were loaded from the according R data file. The sublist names are the same as the object names that are stored. If `simplify=TRUE` is given, all object (sublist) names are checked for duplicates, and if none are found, the list is reduced in one level, dropping the file level.

5.3 Quickly view package (help) index - `help.index`

...

6 Strings and Patterns

6.1 Prepend zeroes to unify number lengths - `leading0`

The function `leading0` aims to create e.g. index names with a common string length. It creates character strings from numeric values while attaching `0` in front of the number up to a certain length of total digits of each string.

```
paste0("page", leading0(8:10, 3))

## [1] "page008" "page009" "page010"
```

6.2 Extract substrings by pattern - strextr

The function **strextr** splits strings in a character vector by **sep** and extracts all substrings matching a given **pattern**.

```
s <- c("a1 b1 c1", "a2 b2", "aa a1", "aa", "b1 a1", "bb ab a1")
strextr(s, "^[ab][[:digit:]]$")

## [1] NA   NA   "a1" NA   NA   "a1"

strextr(s, "^[ab][[:digit:]]$", mult = T)

## [[1]]
## [1] "a1" "b1"
##
## [[2]]
## [1] "a2" "b2"
##
## [[3]]
## [1] "a1"
##
## [[4]]
## [1] NA
##
## [[5]]
## [1] "b1" "a1"
##
## [[6]]
## [1] "a1"

strextr(s, "^[ab][[:digit:]]$", mult = T, unlist = T)

## [1] "a1" "b1" "a2" "b2" "a1" NA   "b1" "a1" "a1"

strextr(s, "^[c][[:digit:]]$")

## [1] "c1" NA   NA   NA   NA   NA
```

6.3 Extract substrings by splitting - strpart

Similar to **strextr** the function **strpart** supplies a method to extract a substring, but by defining the **n**th part of the string split by the separator given in **sep**.

```
s

## [1] "a1 b1 c1" "a2 b2"    "aa a1"    "aa"       "b1 a1"    "bb ab a1"

strpart(s, " ", 2)

## [1] "b1" "b2" "a1" NA   "a1" "ab"
```

6.4 Reverse strings - strrev

With **strrev** you can create the reversed version of strings.

```
strrev(c("olleH", "!dlroW"))

## [1] "Hello"  "World!"
```

6.5 Multiple pattern replacement - `msub, mgsub`

`msub` and `mgsub` behave like `sub` and `gsub` but they replace multiple patterns. Replacement is done in order of the pattern input, and multicore support is enabled by `mclapply` from the parallel package.

```
s

## [1] "a1 b1 c1" "a2 b2"    "aa a1"    "aa"       "b1 a1"    "bb ab a1"

msub("A", "X", s)

## [1] "a1 b1 c1" "a2 b2"    "aa a1"    "aa"       "b1 a1"    "bb ab a1"

mgsub("A", "X", s)

## [1] "a1 b1 c1" "a2 b2"    "aa a1"    "aa"       "b1 a1"    "bb ab a1"
```

6.6 Get index of expression - `gregexprind`

```
s

## [1] "a1 b1 c1" "a2 b2"    "aa a1"    "aa"       "b1 a1"    "bb ab a1"

gregexprind("a", s, 1)

## [1] 1 1 1 1 4 4

gregexprind("a", s, 2)

## [1] NA NA  2  2 NA  7

gregexprind("a", s, "last")

## [1] 1 1 4 2 4 7
```

6.7 Multiple pattern search - `mgrepl`

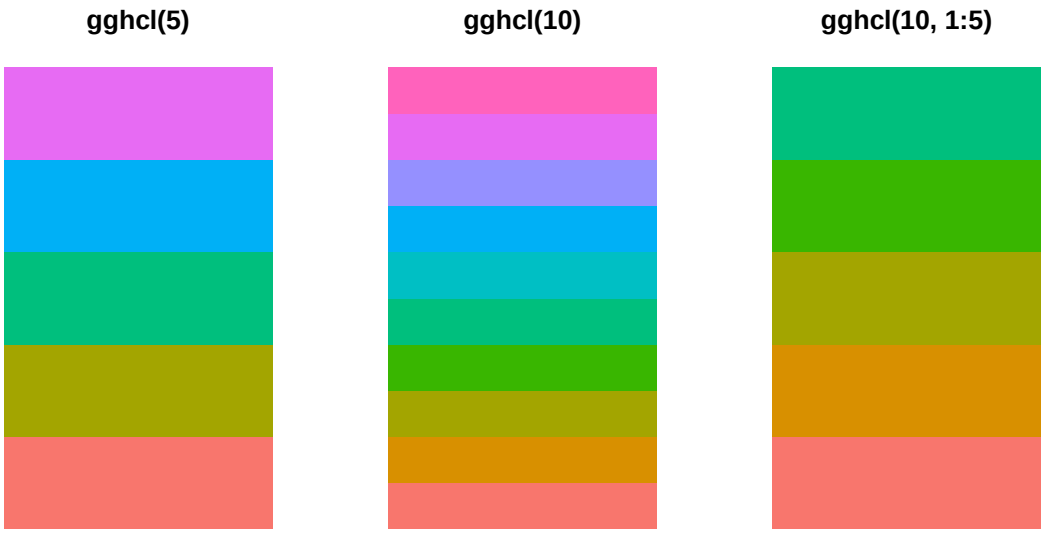
With `mgrepl(patterns, text, ...)` you can search for more than one regular expression, and use a logical function to combine the results for each single expression. It returns a logical vector, or when `use.which = TRUE` an integer vector with the indices as in the base function `which`.

```
s

## [1] "a1 b1 c1" "a2 b2"    "aa a1"    "aa"       "b1 a1"    "bb ab a1"

mgrepl(c("a","b"), s, any)

## [1] TRUE TRUE TRUE TRUE TRUE TRUE

mgrepl(c("a","b"), s, all)

## [1]  TRUE  TRUE FALSE FALSE  TRUE  TRUE

mgrepl(c("a","b"), s, all, use.which = TRUE)

## [1] 1 2 5 6
```

7 Graphics

7.1 Create a color palette - gghcl

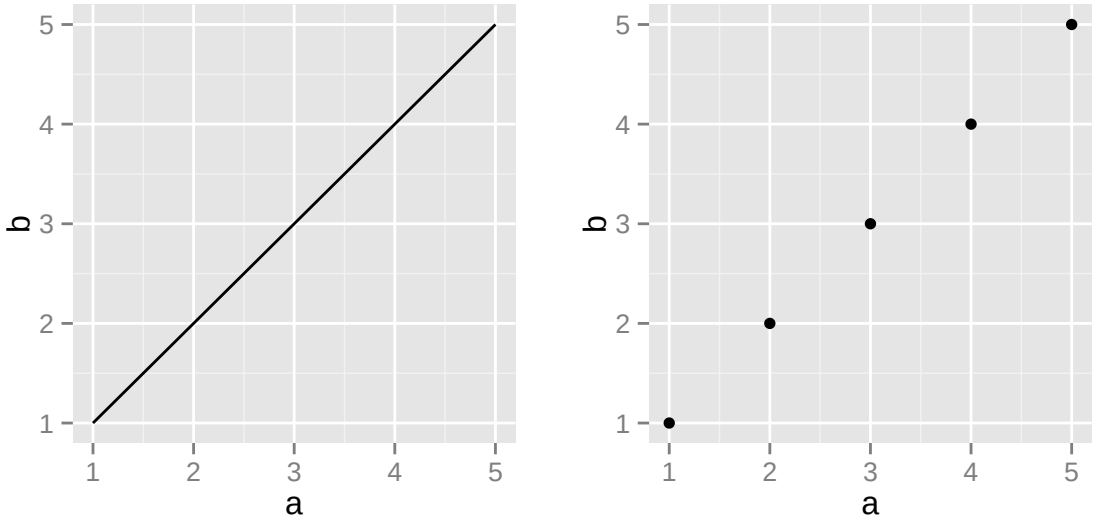
`gghcl()` creates color palettes. It enhances the `hcl` function. See some examples:



7.2 Arrange a list of ggplots in a grid - ggplotGrid, ggplotGridA4

`ggplotGrid` arranges your `ggplot` objects in a grid and optionally sends them to a graphics device such as `pdf/ps/svg/png/eps` for export to a file.

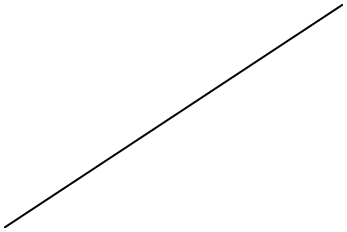
```
library(ggplot2)
d <- data.frame(a=1:5, b=1:5)
gg1 <- ggplot(d, aes(x=a, y=b)) + geom_line()
gg2 <- ggplot(d, aes(x=a, y=b)) + geom_point()
ggplotGrid(list(gg1, gg2), ncol = 2)
```



7.3 Create an empty plot - plotn

To produce nothing but a plot, use `plotn`:

```
plotn(1)
abline(0,1)
```



8 Deprecated

8.1 Arrange a list of ggplots in a grid - `ggplotlist`

The function `ggplotlist` is deprecated and will be dropped in future releases. Please use `ggplotGrid` as replacement.