Package 'CB'

July 1, 2016

Title Cole Brokamp's Personal Functions

Version 0.2

Description Cole's personal R functions

Depends R (>= 3.1.2)

Imports parallel,
XML,
RCurl,
pbapply,
plyr,
rmarkdown,
rstudioapi,
progress

License MIT + file LICENSE

LazyData true

RoxygenNote 5.0.1

R topics documented:

CBapply *deprecated! use cb_apply instead*

Description

This function is a wrapper for sapply with simplify=FALSE and USE.NAMES=TRUE. It then rbinds via do.call to return data.frame. In order for the names to work properly, a function that returns a data.frame must be used (see example).

Usage

```
CBapply(X, FUN, output = "data.frame", fill = FALSE, num.cores = 1, ...)
```

Arguments

X	List of objects to apply over
FUN	Function to apply
output	Output type. Defaults to 'data.frame', but can also be set to 'list' to suppress rbinding of the list.
fill	(defaults to FALSE) use plry::rbind.fill to fill in missing columns
num.cores	Defaults to 1 and the base 'sapply' is used. If set to greater than one, then it is the number of cores used in parallel::mclapply().
...	Additional arguments to the function

Examples

```
X <- as.data.frame(matrix(runif(100),ncol=10))
names(X) <- LETTERS[1:10]
# CBapply(X,mean) # <- will return error
# function must return a data.frame with named columns for column names to work
CBapply(X,function(x) data.frame('mean'=mean(x)))
```

cb_apply *custom *apply function*

Description

Function designed to handle anything that lapply can but can specify parallel processing, progress bars, output format and more.

Usage

```
cb_apply(X, FUN., output = "data.frame", fill = TRUE, pb = TRUE,
  parallel = FALSE, num.cores = NULL, ...)
```

Arguments

X	List of objects to apply over
FUN.	Function to apply
output	Output type. Defaults to 'data.frame', but can also be set to 'list' to suppress rbinding of the list.
fill	(defaults to FALSE) use plyr::rbind.fill to fill in missing columns when rbinding together results
pb	logical; use progress bar?
parallel	logical; use parallel processing?
num.cores	The number of cores used for parallel processing. Can be specified as an integer, or it will guess the number of cores available with detectCores(). If parallel is FALSE, the input here will be set to 1.
...	Additional arguments to the function

Details

Ideally, a function that returns a data.frame should be supplied. This gives the user the advantage of specifying the names of the columns in the resulting data.frame. If the function does not return a data.frame, then column names will be automatically generated.

Parallel processing is carried out by `pbapply::mclapply`. Use the `parallel` option to switch parallel processing on or off. Only specify the number of cores when really needed as the function will detect the maximum number of available cores. This makes it easy to rerun the script with a higher number of available cores without having to change the code.

A progress bar can be shown in the terminal using an interactive R session or in an .Rout file, if using R CMD BATCH and submitting R scripts for non-interactive completion. Although R Studio supports the progress bar for single process workers, it has a problem showing the progress bar if using parallel processing (see the discussion at http://stackoverflow.com/questions/27314011/mcfork-in-rstudio). In this specific case (R Studio + parallel processing), text updates will be printed to the file '.process'. Use a shell and 'tail -f .progress' to see the updates.

Examples

```
X <- as.data.frame(matrix(runif(100),ncol=10))

fun. <- function(x) {
   Sys.sleep(0.5)
   mean(x)
}

cb_apply(X,fun.)

fun. <- function(x) {
  Sys.sleep(0.5)
  data.frame('mean'=mean(x),'median'=median(x))
}

cb_apply(X,fun.,pb=TRUE)

# when setting names of input object, function will attempt to assign them to the output
names(X) <- LETTERS[1:10]
cb_apply(X,fun.,pb=TRUE)
```

```
cb_apply(X,fun.,output='list',pb=TRUE)

# but if function returns a data.frame with more than one row,
   # it can't assign unique row.names to a data.frame, but it can for a list
fun. <- function(x) {
  Sys.sleep(0.5)
  data.frame('summ_stat'=c(mean(x),median(x)))
}
# don't run because parallel processing
# cb_apply(X,fun.,output='data.frame',parallel=TRUE,pb=TRUE)
# cb_apply(X,fun.,output='list',parallel=TRUE,pb=FALSE)
```

`cchmc_color` *Access CCHMC color palette*

Description

Access CCHMC color palette

Usage

```
cchmc_color(n, alpha)
```

Arguments

`n`	which cchmc color (1:purple, 2:green, 3:blue, 4:grey)
`alpha`	create semi-transparent colors (0 < alpha < 1)

Value

a named character string or vector of named character strings containing RGB colors in hexadecimal

Examples

```
cchmc_color(1)
cchmc_color(2)
cchmc_color(3)
cchmc_color(4)
cchmc_color(1:4)

plot(iris$Sepal.Length,iris$Sepal.Width,pch=19,
     col=cchmc_color(sapply(iris$Species,function(x) switch(as.character(x),
                                                            'setosa' = 1,
                                                            'versicolor' = 2,
                                                            'virginica' = 3))))
```

date_print *Print the current date in a pretty format*

Description

Print the current date in a pretty format

Usage

```
date_print()
```

Value

string

Examples

```
date_print()
```

getPackages *getPackages*

Description

This function takes a package and returns a list of its dependencies. Good for downloading source files of packages to install on a R server where internet access is blocked.

Usage

```
getPackages(packs)
```

Arguments

packs	a quoted package name or list of package names

Examples

```
## Not run:
# use this to get specifically named packages and their dependencies:
packages <- getPackages('pbapply')
# use this to get all packages installed on local machine and their dependencies:
# packages <- getPackages(row.names(installed.packages()))
# then download the packages:
download.packages(packages, destdir='.',type='source')

## End(Not run)
```

LatLongToFIPS *Converting Lat/Long Coords into FIPS code*

Description

This function takes a latitude and longitude input numbers and returns the FIPS code by calling the Census Block Conversion API at the FCC.gov website. (See more details here: http://www.fcc.gov/developers/census-block-conversions-api)

Usage

```
LatLongToFIPS(latitude, longitude, census.year = "2010", showall = "false")
```

Arguments

latitude	Latitude coordinate
longitude	Longitude coordinate
census.year	Defaults to '2010'. Not tested on other years; shouldn't need to change as FIPS locations rarely change.
showall	Set to 'false' as defualt. Has to do with the FCC API; shouldn't need to change

Examples

```
LatLongToFIPS(latitude=39.135398,longitude=-84.519902)
```

mclapply_pb *Wrapper around mclapply to track progress*

Description

Doesn't work in RStudio! Based on http://stackoverflow.com/questions/10984556

Usage

```
mclapply_pb(X, FUN, ..., mc.preschedule = TRUE, mc.set.seed = TRUE,
  mc.silent = FALSE, mc.cores = getOption("mc.cores", 2L),
  mc.cleanup = TRUE, mc.allow.recursive = TRUE, mc.progress = TRUE)
```

Arguments

X	a vector (atomic or list) or an expressions vector. Other objects (including classed objects) will be coerced by <e2><80><98>as.list<e2><80><99>
FUN	the function to be applied to
...	optional arguments to <e2><80><98>FUN<e2><80><99>
mc.preschedule	see mclapply
mc.set.seed	see mclapply
mc.silent	see mclapply

mc.cores	see mclapply
mc.cleanup	see mclapply
mc.allow.recursive	see mclapply
mc.progress	track progress?

ORGetter *Retreive Odds Ratio Table from Logistic GLM Objects*

Description

This function returns a data.frame of the odds ratios and their 95% confidence intervals.

Usage

```
ORGetter(logistic.glm, digits = 2, sig.star = TRUE,
  show.intercept = FALSE)
```

Arguments

logistic.glm	A logistic GLM R object. If not an object of 'glm' and 'lm', it will stop with an error.
digits	Number of digits to round table
sig.star	Will return an extra column with a star if the confidence interval does not contain 1. Defaults to TRUE.
show.intercept	Will show the intercept and its confidence interval only if set to TRUE. Defaults to FALSE.

Examples

```
## Not run:  x1 <- rnorm(100)
x2 <- rnorm(100)
y <- rbinom(100,1,prob=0.3)
logistic.model <- glm(y ~ x1 + x2,family='binomial')
ORGetter(logistic.model)

## End(Not run)
```

save_pdf *save_pdf*

Description

Copies the graphics contents of current device to PDF (a wrapper for dev2.pdf). Default size is 8.5 x 11 in landscape mode.

Usage

```
save_pdf(file, width = 11, height = 8.5)
```

Arguments

file	filename to save the image
width	width of pdf image
height	height of pdf image

SPapply *SPapply*

Description

*apply function for spatial point objects

Usage

```
SPapply(sp.object, FUN., ..., progress.bar = TRUE, id.row.names = FALSE)
```

Arguments

sp.object	a SpatialPoints or SpatialPointsDataFrame object
FUN.	function to be applied; must take sp.object as first argument and must return a data.frame
...	additional arguments passed to function
progress.bar	logical, show progress bar?
id.row.names	if TRUE, set row.names of output data.frame from data$id of sp.object

Value

data.frame of all results from function applied to sp.object

tableSummary *Summary Table*

Description

This function summarizes numerical and dichotomous variables only. The summary number is either the mean of a numeric variable for the number and percentage of values that are the second of the two factors in a dichotomous variable. Missing values are removed before the summary statistic is calculated and the numer of missing observations is also presented in the table.

Usage

```
tableSummary(x, digits.mean = 2, digits.percentage = 0)
```

Arguments

x	Vector of data which to summarize. Should be used for numerical and dichotomous variables only.
digits.mean	The mean is rounded and displayed using this many digits.
digits.percentage	The percentage is rounded and displayed using this many digits.

Examples

```
X <- data.frame('some.continuous'=runif(300),'some.factor'=factor(rbinom(300,1,0.3)))
tableSummary(X$some.continuous)
# use CBapply to create a table
CBapply(X,tableSummary)
# specify the digits differently to change the display of the table
CBapply(X,tableSummary,digits.mean=3,digits.percentage=2)
```

tableTest *Table Test*

Description

This function summarizes and tests the differences of numerical and dichotomous variables only across some factor. The summary number is either the mean of a numeric variable for the number and percentage of values that are the second of the two factors in a dichotomous variable. Missing values are removed before the summary statistic, but the number missing is not reported. Furthermore, a p-value is reported testing the differences of the means or counts across the groups factor. The p-value is derived from an ANOVA for continuous variables or from a chi-squared test via monte-carlo simulation using 100,000 bootstrap replicates.

Usage

```
tableTest(x, group, digits.mean = 2, digits.percentage = 0)
```

Arguments

x	Vector of data which to summarize. Should be used for numerical and dichotomous variables only.
group	The factor for which to test the x variable across.
digits.mean	The mean is rounded and displayed using this many digits.
digits.percentage	The percentage is rounded and displayed using this many digits.

Examples

```
X <- data.frame('some.continuous'=runif(300),'some.factor'=factor(rbinom(300,1,0.3)))
X$some.other.factor <- factor(rbinom(300,1,0.5))
tableTest(x=X$some.continuous,group=X$some.other.factor)
tableTest(x=X$some.factor,group=X$some.other.factor)
CBapply(X[ ,c('some.continuous','some.factor')],tableTest,group=X$some.other.factor)
```

Index