Package 'CB'

May 8, 2016

Title Cole Brokamp's Personal Functions

Version 0.1

Description Cole's personal R functions

Depends R (>= 3.1.2)

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

License GPL

LazyData true

RoxygenNote 5.0.1

R topics documented:

`CBapply` *CB's Custom Apply Function*

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 *cb_apply*

Description

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

Usage

```
cb_apply(X, FUN., output = "data.frame", fill = TRUE, names = "row.names",
  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 binding together results
names	how to record the names of X in the resulting list or data.frame. see details
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

TODO: make progress bar for parallel processings;

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 ('V1','V2',...).

Three options exist for the names parameter: (1) 'row.names' will attempt to assign the names of X to the row names of the resulting data.frame. If the names are not unique, then they will instead be assigned to a column in the resulting data.frame called "names". (2) any other character string (i.e. "names" or "id") will assign the names of X to a new column in the resulting data.frame with that character string. If X does not have a names attribute, then names will be automatically generated ('V1','V2',...). Lastly, (3) a vector of character strings will assign this vector as the row.names; it must have as many elements as the number of rows in the resulting data frame.

If not running an interactive R session, then no progress bar is printed. However, if parallel processing is being used and progress bar is requested, the text progress bar will be printed with each interation. This is useful for tracking progress of functions while they are running in batch mode on a server, for example.

`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)
```

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