Package ‘s3cR’

June 25, 2015

Type Package

Title Specific Contributions to Community Changes in R

Version 1.0

Date 2015-01-29

Author Ga<c3><bc>z<c3><a8>re Pierre & Doulcier Guilhem

Maintainer Who to complain to `<pierre.gauzere@gmail.com>`

Description s3cR is a small R package written to compute community weighted indices and specific contributions in their variations.

License CC

R topics documented:

`s3cR-package` *What the package does (short line) ~~ package title ~~*

Description

More about what it does (maybe more than one line) ~~ A concise (1-5 lines) description of the package ~~

Details

Package:	s3cR
Type:	Package
Version:	1.0
Date:	2015-06-25
License:	What license is it under?

~~ An overview of how to use the package, including the most important functions ~~

Author(s)

Who wrote it

Maintainer: Who to complain to <yourfault@somewhere.net> ~~ The author and/or maintainer of the package ~~

References

~~ Literature or other references for background information ~~

See Also

~~ Optional links to other man pages, e.g. ~~ ~~ <pkg> ~~

Examples

```
data(census)
data(species)
out<-cwi(census, species, trait_val_col = "size")
print(out)

out<-cwi(census, species, trait_val_col = "size",
         bootstrap=TRUE,
         bootstrap_n=100,
         bootstrap_ci=95)
```

`bootstrap_cwi` *Bootstrap estimator of CWM/CWV*

Description

Bootstrap is performed /at individual level/

Usage

```
bootstrap_cwi(df, trait_val_col = "trait_val", k, bootstrap_ci, bessel)
```

Arguments

`df`	Census dataframe
`trait_val_col`	column corresponding to the trait values to use
`k`	Integer number of bootstrap re-sampling
`bootstrap_ci`	Percentile of bootstrap confidence interval
`bessel`	If TRUE, use bessel correction for an unbiased variance estimator (N/(N-1))

Value

Returns data frame with the bootstrap estimators of CWM/CWV and the corresponding confidence interval.

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function (df, trait_val_col = "trait_val", k, bootstrap_ci, bessel)
{
    N = sum(df$n)
    df$rN <- df$n/N
    df <- df[order(df$rN, decreasing = T), ]
    df$proba = cumsum(df$rN)
    sp = length(df$proba)
    cwm = array(0, k)
    cwv = array(0, k)
    for (i in 1:k) {
        df$n <- hist(runif(N), breaks = c(0, df$proba), plot = F)$count
        cwm[i] = cwmean(df)
        cwv[i] = cwvar(df, cwm = cwm[i])
    }
    out <- data.frame(bootstrap_cwm = mean(cwm), bootstrap_cwv = mean(cwv),
        bootstrap_cwm_lower_ci = quantile(cwm, 1 - (bootstrap_ci/100)),
        bootstrap_cwv_lower_ci = quantile(cwv, 1 - (bootstrap_ci/100)),
        bootstrap_cwm_higher_ci = quantile(cwm, bootstrap_ci/100),
        bootstrap_cwv_higher_ci = quantile(cwv, bootstrap_ci/100))
    return(out)
  }
```

---

census *census data frame*

Usage

```
data("census")
```

Format

A data frame with 48 observations on the following 4 variables.

`site` a factor with levels `ST000` `ST001` `ST002` `ST003`

`date` a numeric vector

`species` a factor with levels `SP000` `SP001` `SP002` `SP003`

`n` a numeric vector

Examples

```
data(census)
## maybe str(census) ; plot(census) ...
```

`contrib` *Compute specific contributions to Community Weighted indexes variations*

Description

The function s3c.contrib(initial_census,final_census,species) will compute the community weighted means and variances specific contributions and their decompositions.

Usage

```
contrib(census_i, census_f, species, trait_val_col = "trait_val")
```

Arguments

`census_i`

`census_f`

`species`

`trait_val_col`

Examples

```


data(census)
data(species)

ctrb<-contrib(census_i=census[census$date==2,],
              census_f=census[census$date==3,],
              species, trait_val_col="size")
print(ctrb)

cwm2003<-cwi(census[census$date==3,], species, "size", bessel=F)
cwm2002<-cwi(census[census$date==2,], species, "size", bessel=F)
print(cwm2003-cwm2002)


##---- Should be DIRECTLY executable !! ----
```

```
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function (census_i, census_f, species, trait_val_col = "trait_val")
{
    census <- join(census_i, census_f, type = "full")
    species = species[, c("species", "trait_val")]
    species <- species[species$species %in% unique(census$species),
        ]
    species$originality = species$trait_val - mean(species$trait_val)
    species$v_originality = (species$trait_val^2) - mean(species$trait_val^2)
    census <- join(census, species)
    S = cwmean(census[census$date == 1, ]) + cwmean(census[census$date ==
        3, ])
    species$v_cross = species$originality * S
   rN_i <- daply(census[census$date == 1, ], .(species), function(x) sum(x$n))/sum(census[census$date ==
        1, "n"])
   rN_f <- daply(census[census$date == 3, ], .(species), function(x) sum(x$n))/sum(census[census$date ==
        3, "n"])
    species$dp <- rN_f - rN_i
    species$contrib = species$originality * dp
    species$v_contrib = dp * (species$v_originality - species$v_cross)
    return(species)
  }
```

cwi *Compute community weighted indexes of a community*

Description

The function s3c.cwm(census,species) will compute the community weighted means (cwm) and variances (cwv).

Usage

```
cwi(census, traits, trait_val_col = "trait_val", bootstrap = FALSE, bootstrap_n = 100, bootstrap_c
```

Arguments

`census`	census dataframe. Required columns are "n" (number of individual) & "species"
`traits`	trait value dataframe. Required columns are "species" and the trait value colums defined in trait_val_col argument.
`trait_val_col`	name of column in "traits" for which cwi will be applied.
`bootstrap`	If TRUE, perform bootstrap.
`bootstrap_n`	Number of bootstrap re-sampling.
`bootstrap_ci`	Percentile of bootstrap confidence interval.
`bessel`	If TRUE, use bessel correction for an unbiased variance estimator (N/(N-1)).

Value

A data.frame with CWM/CWV estimation and bootstraps estimators and the corresponding confidence interval if required.

Examples

```
data(census)
data(species)
out<-cwi(census, species, trait_val_col = "size")
print(out)

out<-cwi(census, species, trait_val_col = "size",
         bootstrap=TRUE,
         bootstrap_n=100,
         bootstrap_ci=95)

##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function (census, traits, trait_val_col = "trait_val", bootstrap = FALSE,
    bootstrap_n = 100, bootstrap_ci = 95, bessel = TRUE)
{
    census$species <- droplevels(census$species)
    present_sp <- unique(census$species)
    traits <- traits[traits$species %in% census$species, ]
    census = ddply(census, .(species), summarize, n = sum(n),
        .drop = F)
    merged = merge(census, traits, by = "species")
    merged$trait_val <- merged[[trait_val_col]]
    out = data.frame(cwm = 999, cwv = 999)
    out$cwm = cwmean(merged)
    out$cwv = cwvar(merged, cwm = out[["cwm"]])
    if (bootstrap == TRUE) {
        out_boot = bootstrap_cwi(merged, k = bootstrap_n, bootstrap_ci = bootstrap_ci,
            bessel = T)
        out <- cbind(out, out_boot)
    }
    return(out)
  }
```

cwi_stratified *Stratified computation of community weighted indexes of a community.*

Description

tratified computation of community weighted indexes of a community.

Usage

```
cwi_stratified(census, traits, trait_val_col = "trait_val", bootstrap = FALSE, bootstrap_n = 100,
```

Arguments

census	(data.frame) Census dataframe. Required columns are "n" (number of individual, numeric), "species", "site", "date".

`traits`	(data.frame) trait value dataframe. Required columns are "species" and the trait value colums defined in trait_val_col argument.
`trait_val_col`	(text) name of column in "traits" for which cwi will be applied.
`bootstrap`	(bool) Perform bootstrap if true.
`bootstrap_n`	(int) Number of bootstrap re-sampling.
`bootstrap_ci`	(num) Percentile of bootstrap confidence interval.
`bessel`	(logical) If TRUE, use bessel correction for an unbiased variance estimator (N/(N-1)).

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function (census, traits, trait_val_col = "trait_val", bootstrap = FALSE,
    bootstrap_n = 100, bootstrap_ci = 95, bessel = TRUE)
{
    census$species <- droplevels(census$species)
    present_sp <- unique(census$species)
    traits <- traits[traits$species %in% census$species, ]
    merged = merge(census, traits, by = "species")
    if (trait_val_col %in% colnames(traits)) {
        merged$trait_val <- merged[[trait_val_col]]
    }
    else {
        print("trait value columns is not defined")
    }
    out <- ddply(merged, .(site, date), function(x) {
        return(cbind(cwm = cwmean(x), cwv = cwvar(x, cwm = cwm),
            n = sum(x$n)))
    })
    if (bootstrap == TRUE) {
        out_boot = bootstrap_cwi(merged, k = bootstrap_n, bootstrap_ci = bootstrap_ci,
            bessel = T)
        out <- cbind(out, out_boot)
    }
    return(out)
  }
```

cwmean *Compute the community weighted mean of the dataframe*

Description

Compute the community weighted mean of the dataframe

Usage

```
cwmean(df, trait_val_col = "trait_val")
```

Arguments

`df`	
`trait_val_col`	

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function (df, trait_val_col = "trait_val")
{
    df$trait_val <- df[trait_val_col]
    return(sum(df$trait_val * df$n/sum(df$n)))
  }
```

`cwvar` *Compute the community weighted mean and community weighted variance of the dataframe.*

Description

Compute the community weighted mean and community weighted variance of the dataframe.

Usage

```
cwvar(df, trait_val_col = "trait_val", bessel = TRUE, cwm = NA)
```

Arguments

`df`	(dataframe) containing a column "n" (number of individuals) and a column referencing the trait value.
`trait_val_col`	(text) names of the column referencing trait value in df
`bessel`	(bool) If TRUE, use bessel correction for an unbiased variance estimator (N/(N-1)).
`cwm`	If Community weighted means has already been computed (it is only to speed up computations)

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function (df, trait_val_col = "trait_val", bessel = TRUE, cwm = NA)
{
    df$trait_val <- df[[trait_val_col]]
    if (is.na(cwm) == T) {
        cwm <- cwmean(df)
    }
    if (bessel == TRUE) {
        corrective_term = sum(df$n)/(sum(df$n) - 1)
    }
    else {
        corrective_term = 1
    }
    n2 <- sum(df$trait_val^2 * df$n/sum(df$n))
    return(cwv = corrective_term * (n2 - cwm^2))
  }
```

`species` *species data frame*

Description

species data frame

Usage

```
data("species")
```

Format

A data frame with 4 observations on the following 3 variables.

`species` a factor with levels `SP000` `SP001` `SP002` `SP003`

`size` a numeric vector

`size_v` a numeric vector

Examples

```
data(species)
## maybe str(species) ; plot(species) ...
```

`trend_contrib` *contribution to CWI temporal trends*

Description

Compute specific contribution to community weighted indexes variations through time using linear tendencies.

Usage

```
trend_contrib(census, traits, trait_val_col = "trait_val")
```

Arguments

`census`

`traits`

`trait_val_col`

Examples

```
data(census)
data(species)

tr_ctrb<-trend_contrib(census, species, trait_val_col="size")
print(tr_ctrb)

##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function (census, traits, trait_val_col = "trait_val")
{
    if (trait_val_col %in% colnames(traits)) {
        traits$trait_val <- traits[[trait_val_col]]
    }
    else {
        print("trait value columns is not defined")
    }
    census$species <- droplevels(census$species)
    present_sp = unique(census$species)
    species <- traits[traits$species %in% census$species, ]
    species <- species[, c("species", "trait_val")]
    species$species <- droplevels(species$species)
    census <- census[census$species %in% species$species, ]
    census$species <- droplevels(census$species)
    species$originality = species$trait_val - mean(species$trait_val)
    species$v_originality = (species$trait_val^2) - mean(species$trait_val^2)
    census <- ddply(census, .(species, date), summarize, N = sum(n))
    census <- ddply(census, .(date), mutate, n_by_date = sum(N))
    census$rN <- census$N/census$n_by_date
    check_rep_time <- ddply(census, .(species), summarize, n_year = length(unique(date)))
    print(paste("species with less than 2 time occurences were removed from analysis : ",
        as.character(check_rep_time[check_rep_time$n_year < 3,
            "species"])))
    census <- census[census$species %in% check_rep_time[check_rep_time$n_year >
        2, "species"], ]
    species$dp <- daply(census, .(species), function(x) {
        mod <- lm(rN ~ date, data = x)
        return(summary(mod)$coef[2, 1])
    })
    census = merge(census, species, by = "species")
    S = (cwmean(census[census$date == min(census$date), ]) +
        cwmean(census[census$date == max(census$date), ]))
    species$contrib = species$originality * species$dp
    species$v_contrib = species$dp * (species$v_originality -
        species$originality * S)
    return(species)
  }
```

Index