# Package 'RcmdrPlugin.TeachStat'

November 14, 2023

**Type** Package

**Title** R Commander Plugin for Teaching Statistical Methods

**Version** 1.1.3

**Description** R Commander plugin for teaching statistical methods.
It adds a new menu for making easier the teaching of the main concepts about the main statistical methods.

**License** GPL (>= 2)

**Depends** R (>= 3.5.0)

**Imports** Rcmdr (>= 2.5-1), tcltk, Hmisc, tcltk2, tseries, IndexNumR, lme4, distr, distrEx

**Suggests** tkrplot, e1071, RcmdrMisc, randtests

**LazyData** TRUE

**Encoding** UTF-8

**NeedsCompilation** no

**Author** Tomás R. Cotos Yañez [aut] (<https://orcid.org/0000-0002-7732-6565>), Manuel A. Mosquera Rodríguez [aut, cre] (<https://orcid.org/0000-0002-4769-6119>), Ana Pérez González [aut] (<https://orcid.org/0000-0003-4706-7125>), Benigno Reguengo Lareo [aut]

**Maintainer** Manuel A. Mosquera Rodríguez <mamrguez@uvigo.es>

**Repository** CRAN

**Date/Publication** 2023-11-14 13:53:28 UTC

## R topics documented:

---

`RcmdrPlugin.TeachStat-package`
*R Commander plugin for teaching statistical methods.*

---

## Description

It adds a new menu for making easier the teaching of the main concepts about the main statistical methods.

## Details

| | |
|---|---|
| Package: | `RcmdrPlugin.TeachStat` |
| Type: | Package |
| Version: | 1.1.2 |
| Date: | 2023-11-13 |
| License: | GPL version 2 or newer |

## Author(s)

Tomás R. Cotos Yañez <cotos@uvigo.gal>
Manuel A. Mosquera Rodríguez <mamrguez@uvigo.gal>
Ana Pérez González <anapg@uvigo.gal>
Benigno Reguengo Lareo <benireguengo@gmail.com>

## See Also

Rcmdr.

---

`Agrupadas` *Grouped or tabulated data set*

---

## Description

Grouped or tabulated data set, given by lower and upper limits and frequency. It is used as an example for the use of the *Numerical Summaries - Tabulated data* window of the `RcmdrPlugin.TeachStat` package

## Usage

```
data("Agrupadas")
```

## Format

Data frame with 4 cases (rows) and 3 variables (columns).

`Linf` Numeric value, the lower limit of the tabulated data.

`Lsup` Numeric value, the upper limit of the tabulated data.

`ni` Numeric value, the frequency of the tabulated data.

## Examples

```
data(Agrupadas)
calcularResumenDatosTabulados(l_inf=Agrupadas$Linf, l_sup=Agrupadas$Lsup,
  ni=Agrupadas$ni, statistics =c("mean", "sd", "IQR", "quantiles"), quantiles
  = c(0,0.25,0.5,0.75,1), tablaFrecuencia=FALSE)
```

aovreml *ANOVA with random effects using the (REstricted) Maximum Likelihood method.*

## Description

Print the ANOVA table with random effects and compute the point estimations of the variance components using the maximum likelihood method or the REstricted Maximum Likelihood (REML) method. It also provides some confidence intervals.

## Usage

```
aovreml(formula, data = NULL, Lconfint = FALSE, REML = TRUE, ...)
```

## Arguments

| | |
|---|---|
| formula | A formula specifying the model. Random-effects terms are distinguished by vertical bars (\|) separating expressions for design matrices from grouping factors. Two vertical bars (\|\|) can be used to specify multiple uncorrelated random effects for the same grouping variable. (Because of the way it is implemented, the \|\|-syntax works only for design matrices containing numeric (continuous) predictors.) |
| data | an optional data frame containing the variables named in formula. By default the variables are taken from the environment of formula. |
| Lconfint | logical scalar - Should the confidence intervals be printed? |
| REML | logical scalar - Should the estimates be chosen to optimize the REML criterion (as opposed to the log-likelihood)? |
| ... | Arguments to be passed to other functions. |

## Value

A list

## See Also

aov, lmer, aovremm.

## 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 (formula, data = NULL, Lconfint = FALSE, REML = TRUE,
    ...)
{
```

```
  vars <- all.vars(formula)
  formulaaov <- as.formula(paste(vars[1], "~", vars[2]))
  ANOV <- aov(formulaaov, data, ...)
  .ANOV <- summary(ANOV)
  cat("-------------------------------")
  cat("\n", gettext("ANOVA table", domain = "R-RcmdrPlugin.TeachStat"),
      ":\n", sep = "")
  print(.ANOV)
  cat("\n-------------------------------\n")
  .sol <- lme4::lmer(formula, data = data, REML = REML, ...)
  .varcor <- lme4::VarCorr(.sol)
  .sighat2 <- unname(attr(.varcor, "sc"))^2
  .sighatalph2 <- unname(attr(.varcor[[vars[2]]], "stddev"))^2
  .prop <- .sighatalph2/(.sighatalph2 + .sighat2)
  estim <- c(.sighat2, .sighatalph2, .prop)
  names(estim) <- c("var (Error)", "var (Effect)", "% var (Effect)")
  cat("\n", gettext("Components of Variance", domain = "R-RcmdrPlugin.TeachStat"),
      " (", lme4::methTitle(.sol@devcomp$dims), "):\n", sep = "")
  print(estim)
  if (Lconfint) {
      cat("\n", gettext("Confidence intervals", domain = "R-RcmdrPlugin.TeachStat"),
          ":\n", sep = "")
      print(confint(.sol, oldNames = FALSE))
  }
  return(invisible(list(model = .sol, estimation = estim)))
}
```

---

aovremm *ANOVA with random effects using the Moments method.*

---

## Description

Print the ANOVA table with random effects and compute the classical point estimations of the variance components using the Moments method.

## Usage

```
aovremm(formula, data = NULL, ...)
```

## Arguments

| | |
|---|---|
| formula | A formula specifying the model. |
| data | A data frame in which the variables specified in the formula will be found. If missing, the variables are searched for in the standard way. |
| ... | Arguments to be passed to other functions. |

## Value

A list

## See Also

aov, lmer, aovreml.

## 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 (formula, data = NULL, ...)
{
    ANOV <- aov(formula, data, ...)
    .ANOV <- summary(ANOV)
    cat("--------------------------------")
    cat("\n", gettext("ANOVA table", domain = "R-RcmdrPlugin.TeachStat"),
        ":\n", sep = "")
    print(.ANOV)
    cat("\n--------------------------------\n\n")
    .sighat2 <- .ANOV[[1]]$`Mean Sq`[2]
    .vars <- all.vars(formula)
    .groups <- data[[.vars[2]]][!is.na(data[[.vars[1]]])]
    .n <- length(.groups)
    .ni <- table(.groups)
    .c <- (.n^2 - sum(.ni^2))/(.n * (length(.ni) - 1))
    .sighatalph2 <- (.ANOV[[1]]$`Mean Sq`[1] - .sighat2)/.c
    if (.sighatalph2 < 0)
        warning("Estimation of any variance component is not positive. The variance
                component model is inadequate.")
    .prop <- .sighatalph2/(.sighatalph2 + .sighat2)
    estim <- c(.sighat2, .sighatalph2, .prop)
    names(estim) <- c("var (Error)", "var (Effect)", "% var (Effect)")
    cat("\n", gettext("Components of Variance", domain = "R-RcmdrPlugin.TeachStat"),
        ":\n", sep = "")
    print(estim)
    return(invisible(list(model = ANOV, estimation = estim)))
  }
```

---

`calcularResumenDatosTabulados`

*Summary statistics for tabulated data*

---

## Description

`calcularResumenDatosTabulados` performs the main statistical summary for tabulated data (mean, standard deviation, coefficient of variation, skewness, kurtosis, quantile and mode) are calculated. Also it allows to obtain the frequency table (with classmark, amplitude and density).

## Usage

```
calcularResumenDatosTabulados(l_inf, l_sup, ni,
                              statistics = c("mean", "sd", "se(mean)", "IQR",
                              "quantiles", "cv", "skewness", "kurtosis"),
                              quantiles = c(0, 0.25, 0.5, 0.75, 1),
                              tablaFrecuencia = FALSE)
```

## Arguments

| | |
|---|---|
| `l_inf` | numeric vector with the lower limit of each interval. |
| `l_sup` | numeric vector with the upper limit of each interval. |
| `ni` | numeric vector with the frequency of occurrence of values in the range between the lower limit and upper limit [l_inf [i-1], l_sup [i]). |
| `statistics` | any of `"mean"`, `"sd"`, `"se(mean)"`, `"quantiles"`, `"cv"` (coefficient of variation - sd/mean), `"skewness"`, `"kurtosis"` or `"mode"`; defaulting to `c("mean", "sd", "quantiles", "IQR")`. |
| `quantiles` | quantiles to report; by default is `c(0, 0.25, 0.5, 0.75, 1)`. |
| `tablaFrecuencia` | logical value indicating whether or not to display the frequency table, by default is `FALSE`. |

## Details

`calcularResumenDatosTabulados` performs an analysis of **tabulated data** (frequently used in statistics when the number of distinct values is large or when dealing with continuous quantitative variables), represented by a table of statistics (arithmetic mean, standard deviation, interquartile range, coefficient of variation, asymmetry, kurtosis, and quantile).

It also allows to show the frequency table of the tabulated variable by selecting `tablaFrecuencia=TRUE`. The class mark, amplitude and density are added to the frequency table.

The `LOWER LIMIT` or `L[i-1]` and `UPPER LIMIT` or `L[i]` vectors, represent the data of continuous quantitative variables in class intervals of the form [L[i-1], L[i]) where i = 1, .. ., k.

## Value

`calcularResumenDatosTabulados()` returns a list of two elements:

| | |
|---|---|
| `.numsummary` | an object of class `"numSummary"` containing the numerical summary of the tabulated variable. |
| `.table` | a matrix containing the values of the frequency table. |

## See Also

`cut`

## Examples

```
data(cars93)
cortes <- seq(from=1500, to=4250, by=250)
aa <- cut( cars93$Weight, breaks=cortes, dig.lab=4)
ni <- table(aa)
l_inf <- cortes[-length(cortes)]
l_sup <- cortes[-1]
agrup <- data.frame(l_inf,l_sup,ni)
head(agrup)

calcularResumenDatosTabulados(agrup$l_inf, agrup$l_sup, agrup$Freq)
calcularResumenDatosTabulados(agrup$l_inf, agrup$l_sup, agrup$Freq, tabla=TRUE)

bb <- calcularResumenDatosTabulados(agrup$l_inf, agrup$l_sup, agrup$Freq,
                                    statistics=c("mean","mode") )
bb
str(bb)
class(bb$.summary)
class(bb$.table)
```

---

# calcularResumenVariablesContinuas

*Summary statistics for continuous variables*

---

## Description

calcularResumenVariablesContinuas gives the main statistical summary for continuous variables (mean, standard deviation, coefficient of variation, skewness, kurtosis and quantiles). Also builds the frequency table (with classmark, amplitude and density).

## Usage

```
calcularResumenVariablesContinuas(data,
                                  statistics = c("mean", "sd", "se(mean)", "IQR",
                                    "quantiles", "cv", "skewness", "kurtosis"),
                                quantiles = c(0, 0.25, 0.5, 0.75, 1), groups = NULL,
                                  tablaFrecuencia = FALSE, cortes="Sturges", ...)
```

## Arguments

| | |
|---|---|
| data | data.frame with the continuous variables. |
| statistics | any of "mean", "sd", "se(mean)", "quantiles", "cv" (coefficient of variation - sd/mean), "skewness" or "kurtosis"; defaulting to c("mean", "sd", "quantiles", "IQR"). |
| quantiles | quantiles to report; by default is c(0, 0.25, 0.5, 0.75, 1). |
| groups | optional variable, typically a factor, to be used to partition the data. By default is NULL. |

`tablaFrecuencia`
: logical value indicating whether or not to display the frequency table, by default is `FALSE`.

`cortes`
: one of:

- a numeric vector of two or more unique cut points,
- a single number (greater than or equal to 2) giving the number of intervals into which data is to be cut,
- a character string naming an algorithm to compute the number of intervals into which data is to be cut (see 'Details')

by default is `Sturges`.

`...`
: further arguments to be passed to `numSummary`.

## Details

`calcularResumenVariablesContinuas` performs a descriptive analysis of continuous variables (quantitative variables that take infinite distinct values into an interval), generating a table of statistics (arithmetic mean, standard deviation, interquartile range, coefficient of variation, skewness, kurtosis, and quantiles) optionally allowing the partition of the data by a factor variable (`groups`).

It also allows to show the frequency table of selected continuous variables by selecting `tablaFrecuencia=TRUE`. Moreover it also allows to divide the range of the variables into intervals given by the argument `cortes` (breaks). See more info in `cut` and in `hist`.

## Value

`calcularResumenVariablesContinuas` returns a list of two elements:

`.numsummary`
: an object of class `"numSummary"` containing the numerical summary of the continuous variables.

`.table`
: a matrix containing the values of the frequency table.

## See Also

`numSummary`, `cut`, `hist`

## Examples

```
## Not run:
data(cars93)
calcularResumenVariablesContinuas(data=cars93["FuelCapacity"],group=NULL)
calcularResumenVariablesContinuas(data=cars93["FuelCapacity"],group=cars93$Airbags)
bb <- calcularResumenVariablesContinuas(data=cars93["FuelCapacity"],group=cars93$Airbags,
                                        tablaFrecuencia=TRUE)
str(bb)
bb
bb$.summary
class(bb$.summary)

calcularResumenVariablesContinuas(data=cars93["MidPrice"], tablaFrecuencia=TRUE)
calcularResumenVariablesContinuas(data=cars93["MidPrice"], tablaFrecuencia=TRUE, cortes=5)
```

```
calcularResumenVariablesContinuas(data=cars93["MidPrice"], tablaFrecuencia=TRUE,
                                  cortes=c(7,14,21,28,63))
calcularResumenVariablesContinuas(data=cars93["MidPrice"], tablaFrecuencia=TRUE,
                                  cortes="Scott")
calcularResumenVariablesContinuas(data=cars93["MidPrice"], groups=cars93$Airbags,
                                  tablaFrecuencia=TRUE, cortes=5)

## End(Not run)
```

---

`calcularResumenVariablesDiscretas`

*Summary statistics for discrete variables*

---

## Description

`calcularResumenVariablesDiscretas` gives the main statistical summary for discrete variables (mean, standard deviation, coefficient of variation, skewness, kurtosis and quantiles). Also builds the frequency table

## Usage

```
calcularResumenVariablesDiscretas(data,
                                  statistics = c("mean", "sd", "se(mean)", "IQR",
                                    "quantiles", "cv", "skewness", "kurtosis"),
                                quantiles = c(0, 0.25, 0.5, 0.75, 1), groups = NULL,
                                    tablaFrecuencia = FALSE, cortes=NULL)
```

## Arguments

| | |
|---|---|
| `data` | `data.frame` with the discrete variables. |
| `statistics` | any of `"mean"`, `"sd"`, `"se(mean)"`, `"quantiles"`, `"cv"` (coefficient of variation - sd/mean), `"skewness"` or `"kurtosis"`; defaulting to `c("mean", "sd", "quantiles", "IQR")`. |
| `quantiles` | quantiles to report; by default is `c(0, 0.25, 0.5, 0.75, 1)`. |
| `groups` | optional variable, typically a factor, to be used to partition the data. By default is `NULL`. |
| `tablaFrecuencia` | logical value indicating whether or not to display the frequency table, by default is `FALSE`. |
| `cortes` | one of:<br>• a numeric vector of two or more unique cut points,<br>• a single number (greater than or equal to 2) giving the number of intervals into which data is to be cut,<br>• a character string naming an algorithm to compute the number of intervals into which data is to be cut (see 'Details')<br>by default is `NULL`. |

## Details

calcularResumenVariablesDiscretas performs a descriptive analysis of discrete variables (quantitative variables that take as a finite or infinite numerable distinct values), generating a table of statistics (arithmetic mean, standard deviation, interquartile range, coefficient of variation, skewness, kurtosis, and quantiles) optionally allowing the partition of the data by a factor variable (groups).

It also allows to show the frequency table of selected discrete variables by selecting tablaFrecuencia=TRUE. Moreover it also allows to divide the range of the variables into intervals given by the argument cortes (breaks). See more info in cut and in hist.

## Value

calcularResumenVariablesDiscretas returns a list of two elements:

| | |
|---|---|
| .numsummary | an object of class "numSummary" containing the numerical summary of the discrete variables. |
| .table | a matrix containing the values of the frequency table. |

## See Also

cut, hist

## Examples

```
## Not run:
data(cars93)
calcularResumenVariablesDiscretas(data=cars93["Cylinders"],group=NULL)
calcularResumenVariablesDiscretas(data=cars93["Cylinders"],group=cars93$Airbags)
bb <- calcularResumenVariablesDiscretas(data=cars93["Cylinders"],group=cars93$Airbags,
                                        tablaFrecuencia=TRUE)
str(bb)
bb
bb$.summary
class(bb$.summary)

calcularResumenVariablesDiscretas(data=cars93["Horsepower"], tablaFrecuencia=TRUE)
calcularResumenVariablesDiscretas(data=cars93["Horsepower"], tablaFrecuencia=TRUE, cortes=5)
calcularResumenVariablesDiscretas(data=cars93["Horsepower"], tablaFrecuencia=TRUE,
                                  cortes=c(50,100,200,250,300))
calcularResumenVariablesDiscretas(data=cars93["Horsepower"], tablaFrecuencia=TRUE,
                                  cortes="Sturges")
calcularResumenVariablesDiscretas(data=cars93["Horsepower"], groups=cars93$Airbags,
                                  tablaFrecuencia=TRUE, cortes=5)

## End(Not run)
```

`calcular_frecuencia` *Frequency distributions for qualitative variables*

## Description

Performs frequency distribution for qualitative, nominal and/or ordinal variables. For ordinal variables, the requested quantile is calculated.

## Usage

```
calcular_frecuencia(df.nominal, ordenado.frec = FALSE,df.ordinal,
                    cuantil.p = 0.5,iprint = TRUE, ...)
```

## Arguments

| | |
|---|---|
| `df.nominal` | `data.frame` with factor type components (including character type) that are interpreted as nominal variables. |
| `ordenado.frec` | table ordered frequencies depending on their frequency (only used for nominal variables). |
| `df.ordinal` | `data.frame` with factor type components (including character type) that are interpreted as ordinal variables. |
| `cuantil.p` | requested quantile value (only used for ordinal variables). |
| `iprint` | logical value indicating whether or not to display the frequency table. |
| `...` | further arguments to be passed to or from methods. |

## Value

`calcular_frecuencia` returns a list of three elements:

| | |
|---|---|
| `.nominal` | a matrix containing the table of frequency distribution for nominal variables (`ni` = absolute frequencies and `fi` = relative frequencies). |
| `.ordinal` | a matrix containing the table of frequency distribution for ordinal variables (`ni` = absolute frequencies, `fi` = relative frequencies, `Ni` = absolute cumulative frequency, `Fi` = cumulative absolute frequencies). |
| `df.cuantil` | data frame containing the quantiles. |

## See Also

`table` , `cumsum`

## Examples

```
data(cars93)
aa <- calcular_frecuencia(df.nominal=cars93["Type"], ordenado.frec=TRUE, df.ordinal=NULL,
                          cuantil.p=0.5, iprint = TRUE)
calcular_frecuencia(df.nominal=NULL, ordenado.frec=TRUE, df.ordinal=cars93["Airbags"],
                    cuantil.p=0.25, iprint = TRUE)
bb <- calcular_frecuencia(df.nominal=cars93["Type"], ordenado.frec=TRUE,
                          df.ordinal=cars93["Airbags"], cuantil.p=0.25, iprint = FALSE)
str(bb)
bb
```

---

# cars93 *Data from 93 Cars on Sale in the USA in 1993*

---

## Description

The cars93 data frame has 93 rows and 26 columns.

## Usage

```
cars93
```

## Format

This data frame contains the following columns:

`Manufacturer` Manufacturer.

`Model` Model.

`Type` Type: a factor with levels `"Small"`, `"Sporty"`, `"Compact"`, `"Midsize"`, `"Large"` and `"Van"`.

`MinPrice` Minimum Price (in $1,000): price for a basic version.

`MidPrice` Midrange Price (in $1,000): average of `Min.Price` and `Max.Price`.

`MaxPrice` Maximum Price (in $1,000): price for "a premium version".

`CityMPG` City MPG (miles per US gallon by EPA rating).

`HighwayMPG` Highway MPG.

`Airbags` Air Bags standard. Factor: none, driver only, or driver & passenger.

`DriveTrain` Drive train type: rear wheel, front wheel or 4WD; (factor).

`Cylinders` Number of cylinders (missing for Mazda RX-7, which has a rotary engine).

`EngineSize` Engine size (litres).

`Horsepower` Horsepower (maximum).

`RPM` RPM (revs per minute at maximum horsepower).

`EngineRevol` Engine revolutions per mile (in highest gear).

`Manual` Is a manual transmission version available? (yes or no, Factor).

`FuelCapacity` Fuel tank capacity (US gallons).

`Passengers` Passenger capacity (persons)

`Length` Length (inches).

`Wheelbase` Wheelbase (inches).

`Width` Width (inches).

`UTurnSpace` U-turn space (feet).

`RearSeatRoom` Rear seat room (inches) (missing for 2-seater vehicles).

`LuggageCapacity` Luggage capacity (cubic feet) (missing for vans).

`Weight` Weight (pounds).

`USA` Of non-USA or USA company origins? (factor).

## Details

Cars were selected at random from among 1993 passenger car models that were listed in both the *Consumer Reports* issue and the *PACE Buying Guide*. Pickup trucks and Sport/Utility vehicles were eliminated due to incomplete information in the *Consumer Reports* source. Duplicate models (e.g., Dodge Shadow and Plymouth Sundance) were listed at most once.

Further description can be found in Lock (1993).

## Source

Lock, R. H. (1993) 1993 New Car Data. *Journal of Statistics Education* **1**(1). doi:10.1080/10691898.1993.11910459.

## References

Venables, W. N. and Ripley, B. D. (1999) *Modern Applied Statistics with S-PLUS.* Third Edition. Springer.

---

`characRV` *Characteristics of Random Variables.*

---

## Description

This function computes the main characteristics of a random variable (expectation, median, standard deviation, ...)

## Usage

```
characRV(D, charact = c("expectation", "median", "sd", "IQR", "skewness", "kurtosis",
         "moment", "cmoment"), moment = 1, cmoment = 2)
```

## Arguments

| | |
|---|---|
| D | An object of the class Distribution. |
| charact | any of "expectation", "median", "sd", "IQR", "skewness", "kurtosis", "moment", "cmoment" |
| moment | an integer indicating the moment with respect to the origin to be calculated. |
| cmoment | an integer indicating the moment with respect to the expectation to be calculated. |

## Value

'characRV' returns a table containing the selected characteristics.

## See Also

E, median, sd, IQR, skewness, kurtosis

## 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 (D, charact = c("expectation", "median", "sd", "IQR",
    "skewness", "kurtosis", "moment", "cmoment"), moment = 1,
    cmoment = 2)
{
    if (missing(charact))
        charact <- c("expectation", "sd")
    charact <- match.arg(charact, c("expectation", "median",
        "sd", "IQR", "skewness", "kurtosis", "moment", "cmoment"),
        several.ok = TRUE)
    moment <- if ("moment" %in% charact)
        moment
    else NULL
    cmoment <- if ("cmoment" %in% charact)
        cmoment
    else NULL
    mom <- if (!is.null(moment))
        paste("alpha_", moment, sep = "")
    else NULL
    cmom <- if (!is.null(cmoment))
        paste("mu_", cmoment, sep = "")
    else NULL
    chars <- c(c("expectation", "median", "sd", "IQR", "skewness",
        "kurtosis")[c("expectation", "median", "sd", "IQR", "skewness",
        "kurtosis") %in% charact], mom, cmom)
    nchars <- length(chars)
    table <- matrix(0, 1, nchars)
    rownames(table) <- gsub("[[:space:]]", "", deparse(substitute(D)))
    colnames(table) <- chars
```

```
  if ("expectation" %in% chars)
      table[, "expectation"] <- distrEx::E(D)
  if ("median" %in% chars)
      table[, "median"] <- distrEx::median(D)
  if ("sd" %in% chars)
      table[, "sd"] <- distrEx::sd(D)
  if ("IQR" %in% chars)
      table[, "IQR"] <- distrEx::IQR(D)
  if ("skewness" %in% chars)
      table[, "skewness"] <- distrEx::skewness(D)
  if ("kurtosis" %in% chars)
      table[, "kurtosis"] <- distrEx::kurtosis(D)
  if ("moment" %in% charact)
      table[, mom] <- distrEx::E(D, fun = function(x) {
          x^moment
      })
  if ("cmoment" %in% charact)
      table[, cmom] <- distrEx::E(D, fun = function(x) {
          (x - distrEx::E(D))^cmoment
      })
  print(table)
  return(invisible(table))
}
```

---

ComplexIN *Complex index numbers*

---

## Description

ComplexIN computes the aggregation of a set of index numbers using the arithmetic, geometric or harmonic means.

## Usage

```
ComplexIN(data, means = c("arithmetic", "geometric", "harmonic"), zero.rm = TRUE,
          na.rm = TRUE, ...)
```

## Arguments

| | |
|---|---|
| data | Data frame containing, the index numbers to aggregate. |
| means | Character vector with the name of the mean to compute. mean can be arithmetic, geometric or harmonic. |
| zero.rm | Logical string for geometric and harmonic means. TRUE (default) indicates that negative and zero values should be deleted before computing the geometric mean and that zero values should be deleted before computing the harmonic mean. |
| na.rm | Logical value indicating whether NA values should be stripped before the computation proceeds. It is TRUE by default. |
| ... | Further arguments passed to or from other methods. |

### Details

Sindex, Deflat, priceIndexNum.

### Value

Matrix with as many rows as columns of x and as many columns as means selected.

### Examples

```
df <- data.frame(Index=round(runif(12,80,105),2))
ComplexIN(df, means = c("arithmetic", "geometric", "harmonic"))
```

---

## ConvertVariables *Modify Variable Types*

---

### Description

In this graphical interface, the user can modify the variable type into nominal (factor), ordinal (ordered factor) or numeric type.

---

## Cprop.test *Test for proportions of one or two samples*

---

### Description

Performs hypothesis testing and confidence interval for a proportion or difference of two proportions. The values of the samples necessary to perform the function are the number of successes and the number of trails.

### Usage

```
Cprop.test(ex, nx, ey = NULL, ny = NULL, p.null = 0.5,
           alternative = c("two.sided", "less", "greater"), conf.level = 0.95,
           ...)
```

### Arguments

| | |
|---|---|
| ex | numeric value that represents the number of successes of the first sample (see Details). |
| nx | numerical value representing the total number of trails of the first sample. |
| ey | (optional) numerical value representing the number of success of the second sample (see Details). |
| ny | (optional) numerical value representing the total number of trails of the second sample. |

| | |
|---|---|
| p.null | numeric value that represents the value of the population proportion or the difference between the two population proportions, depending on whether there are one or two samples (see Details). |
| alternative | a character string specifying the alternative hypothesis, must be one of "two.sided" (default), "greater" or "less". You can specify just the initial letter. |
| conf.level | confidence level of the interval. |
| ... | further arguments to be passed to or from methods. |

## Details

So that the contrast can be made must be fulfilled that at least 1 hit. That is, in the case of a sample ex must be greater than or equal to 1 and in the case of two samples, ex or ey must be greater than or equal to 1.

Furthermore, for the case of a sample value p.null must be strictly positive.

## Value

A list with class "htest" containing the following components:

| | |
|---|---|
| statistic | the value of the test statistic. |
| parameter | number of trails and value of the population proportion or the difference in population proportions. |
| p.value | the p-value for the test. |
| conf.int | a confidence interval for the proportion or for the difference in proportions, appropriate to the specified alternative hypothesis. |
| estimate | a value with the sample proportions. |
| null.value | the value of the null hypothesis. |
| alternative | a character string describing the alternative. |
| method | a character string indicating the method used, and whether Yates' continuity correction was applied. |
| data.name | a character string giving the names of the data. |

## See Also

prop.test

## Examples

```
## Proportion for a sample
Cprop.test(1,6) # 1 success in 6 attempts

#### With a data set: proportion of cars not manufactured in US
data(cars93) #data set provided with the package
exitos<-sum(cars93$USA == "nonUS")
total<-length(cars93$USA)
Cprop.test(ex=exitos, nx=total)
```

```
## Difference of proportions
Cprop.test(1,6,3,15)
  # Sample 1: 1 success in 6 attempts
  # Sample 2: 3 success in 15 attempts

#### With a data set: difference of proportions of cars not manufactured in US
#### between manual and automatic
exitosx<-sum(cars93$USA == "nonUS" & cars93$Manual == "Yes" )
totalx<-sum(cars93$Manual == "Yes")
exitosy<-sum(cars93$USA == "nonUS" & cars93$Manual == "No" )
totaly<-sum(cars93$Manual == "No")
Cprop.test(ex=exitosx, nx=totalx,ey=exitosy, ny=totaly)
```

---

`Deflat` *Deflation of an economic series*

---

## Description

`Deflat` deflates a current value variable into a constant value variable.

## Usage

```
Deflat(x, pervar, cvar, defl, base)
```

## Arguments

| | |
|---|---|
| `x` | Data frame containing, at least, the characteristics (time, location, ...), the current value and the deflator variables. |
| `pervar` | Character string for the name of the factor variable with the characteristics. |
| `cvar` | Character string for the name of the numeric variable with the current values. |
| `defl` | Character string for the name of the numeric variable with the index number used as deflator. |
| `base` | Character string for the name of the base characteristic. |

## Value

`Deflat` returns a data frame with one column:

| | |
|---|---|
| `const_base` | The variable with constant values at base `base` |

## See Also

`Sindex`, `ComplexIN`, `priceIndexNum`.

## Examples

```
data(Depositos, package = "RcmdrPlugin.TeachStat")
Deflat(Depositos, "year", "quantity", "G_IPC_2016", "2018")
```

`Depositos` *Deposits with credit institutions in Ourense*

## Description

Private sector deposits (in millions of euro) with credit institutions in the province of Ourense (Spain) in 2002-2018.

## Usage

```
data("Depositos")
```

## Format

A data frame with 17 observations on the following 4 variables.

`year` a factor, year

`quantity` a numeric vector, deposit (in millions of euro) with credit institutions

`E_IPC_2016` a numeric vector, Consumer Price Index (CPI) with base 2016 in Spain

`G_IPC_2016` a numeric vector, Consumer Price Index (CPI) with base 2016 in Galicia

## Source

Galician Institute of Statistics (2019):

- https://www.ige.gal/igebdt/esqv.jsp?ruta=verTabla.jsp?OP=1&B=1&M=&COD=462&R=2%5B2002:2003:2004:2005:2006:2007:2008:2009:2010:2011:2012:2013:2014:2015:2016:2017:2018%5D&C=9928%5B32%5D;0%5B3%5D;1%5B3%5D&F=&S=&SCF=#

- https://www.ige.gal/igebdt/esqv.jsp?ruta=verTabla.jsp?OP=1&B=1&M=&COD=8547&R=0%5Ball%5D&C=2%5B0%5D;1%5B0%5D;9928%5B108:12%5D&F=&S=&SCF=

## Examples

```
data(Depositos)

.Sindex <- Sindex(Depositos, "year", "quantity", "2010")*100
print(.Sindex)

Deflat(Depositos, "year", "quantity", "E_IPC_2016", "2011")
```

## distrDefine — *Definition of Random Variables*

### Description

In this window the user can define any random variable by providing its parameters (for a well-known random variable), its distribution function, its density function (for a generic absolutely continuous random variable), or its mass probability function (for a generic discrete random variable).

## DMKV.test — *Z-test for the difference of means of two independent Normal variables with known population variances.*

### Description

Under the assumption that the data come from two independent Normal distributions, it performs the hypothesis test and the confidence interval for the difference of means with known population variances.

### Usage

```
DMKV.test(x, y, difmu = 0, sdx, sdy,
          alternative = c("two.sided", "less", "greater"), conf.level = 0.95,
          ...)
```

### Arguments

| | |
|---|---|
| x | numerical vector (non-empty) that contains the data of the first sample. |
| y | numerical vector (non-empty) containing the data of the second sample. |
| difmu | numeric value indicating the value of the difference in population means between the two samples. |
| sdx | numerical value indicating the population standard deviation of the first sample, which is assumed to be known (mandatory). |
| sdy | numeric value indicating the population standard deviation of the second sample, which is assumed to be known (mandatory). |
| alternative | a character string specifying the alternative hypothesis, must be one of "two.sided" (default), "greater" or "less". You can specify just the initial letter. |
| conf.level | confidence level of the interval. |
| ... | further arguments to be passed to or from methods. |

## Value

A list with class "htest" containing the following components:

| | |
|---|---|
| statistic | the value of the test statistic. |
| parameter | sample lengths and population standard deviations. |
| p.value | the p-value for the test. |
| conf.int | confidence interval for the difference of means with known population variances associated with the specified alternative hypothesis. |
| estimate | the estimated difference in means. |
| null.value | the specified hypothesized value of the mean difference. |
| alternative | a character string describing the alternative hypothesis. |
| method | a character string indicating what type of statistical method was performed. |
| data.name | a character string giving the name(s) of the data. |

## See Also

t.test

## Examples

```
data(cars93) # Data set provided with the package
# Maximum price difference (MaxPrice) in means between cars manufactured in the
# US and those manufactured outside, assuming that the variances are known and
# equal to 64 and 169, respectively
var1<-subset(cars93, USA=="nonUS", select=MaxPrice)
var2<-subset(cars93, USA=="US", select=MaxPrice)
DMKV.test(var1, var2, sdx=13, sdy=8, difmu=0,
alternative="greater", conf.level=0.95)
```

---

# IndexNumbers *Index Numbers menu*

---

## Description

In this menu the user can perform some calculations related to index numbers.

This menu will call the functions for calculating simple index numbers and making base changes (Sindex), for calculating complex index numbers (ComplexIN), for calculating price indices (priceIndexNum), and for deflation of economic series (Deflat).

`intervaloConfianzaMedia`

*Confidence interval or hypothesis testing for the mean of a Normal variable*

## Description

In this graphical interface, the data selection is made to perform the calculation of the confidence interval or the hypothesis testing for the mean of a Normal variable.

This interface will call the statistical functions `MKV.test` and `t.test`, depending, respectively, on whether the population variance is known or not.

`intervaloConfianzaMediasIndependientes`

*Confidence interval or hypothesis testing for the difference in means of two independent Normal variables*

## Description

In this graphical interface, the data selection is made to perform the calculation of the confidence interval or the hypothesis testing for the difference in means of two independent Normal variables.

This interface will call the statistical functions `DMKV.test` and `t.test`, depending, respectively, on whether the population variances are known or not.

`intervaloConfianzaVarianza`

*Confidence interval or hypothesis testing for the Variance*

## Description

In this graphical interface, the data selection is made to perform the calculation of the confidence interval or the hypothesis testing for the variance of a Normal variable.

This interface will call the statistical functions `VKM.test` and `VUM.test`, depending, respectively, on whether the population mean is known or not.

## listTypesVariables *List of variables and types of a Data Frame*

### Description

`listTypesVariables` returns a vector with the names and types of the variables of a data frame.

### Usage

```
listTypesVariables(dataSet)
```

### Arguments

| | |
|---|---|
| `dataSet` | the quoted name of a data frame in memory. |

### Value

A character vector

### See Also

`names`

### Examples

```
require(datasets)
listTypesVariables("iris")
```

## MKV.test *Z-test for the mean of a Normal variable with known population variance.*

### Description

Under the assumption that the data come from a Normal distribution, it makes the hypothesis testing and the confidence interval for the mean with known population variance.

### Usage

```
MKV.test(x, mu = 0, sd, alternative = c("two.sided", "less", "greater"),
         conf.level = 0.95, ...)
```

## Arguments

| | |
|---|---|
| x | a (non-empty) numeric vector of data values. |
| mu | a number indicating the true value of the mean - Null hypothesis. |
| sd | numerical value indicating the population standard deviation assumed to be known (mandatory). |
| alternative | a character string specifying the alternative hypothesis, must be one of "two.sided" (default), "greater" or "less". You can specify just the initial letter. |
| conf.level | confidence level of the interval. |
| ... | further arguments to be passed to or from methods. |

## Value

A list with class "htest" l containing the following components:

| | |
|---|---|
| statistic | the value of the test statistic. |
| parameter | sample length, population standard deviation and sample standard deviation. |
| p.value | the p-value for the test. |
| conf.int | a confidence interval for the mean appropriate to the specified alternative hypothesis. |
| estimate | the estimated mean. |
| null.value | the specified hypothesized value of the mean. |
| alternative | a character string describing the alternative hypothesis. |
| method | a character string indicating what type of statistical test was performed. |
| data.name | a character string giving the name of the data. |

## See Also

t.test

## Examples

```
data(cars93) # Dataset provided with the package
# Mean maximum price (MaxPrice) less than 20 thousand $ assuming that the
# variance is known and equal to 11
MKV.test(cars93$MaxPrice, sd=11, alternative="less", mu=20, conf.level=0.95)
```

plotRegions *Plot regions in probability mass or density functions.*

## Description

This function plot regions in probability mass or density functions.

## Usage

```
plotRegions(D, add = FALSE, regions = NULL, col = "gray", legend = TRUE,
        legend.pos = "topright", to.draw.arg = 1, verticals = FALSE, ngrid = 1000,
            cex.points = par("cex"), mfColRow = FALSE, lwd = par("lwd"), ...)
```

## Arguments

| | |
|---|---|
| D | object of class "AffLinUnivarLebDecDistribution" or class "UnivarLebDecDistribution" or class "AbscontDistribution" or class "DiscreteDistribution" or class "DistrList": (list of) distribution(s) to be plotted |
| add | logical; if TRUE only *add* to an existing plot. |
| regions | a list of regions to fill with color col; each element of the list is a pair of x values with the minimum and maximum horizontal coordinates of the corresponding region. |
| col | may be a single value or a vector indicating the colors of the regions. |
| legend | plot a legend of the regions (default TRUE). |
| legend.pos | position for the legend (see legend, default "topright"). |
| to.draw.arg | Either NULL (default; everything is plotted) or a vector of either integers (the indices of the subplots to be drawn) or characters - the names of the subplots to be drawn: in case of an object x of class "DiscreteDistribution" or "AbscontDistribution" c("d","p","q") for density, c.d.f. and quantile function; in case of x a proper "UnivarLebDecDistribution" (with pos. weights for both discrete and abs. continuous part) names are c("p","q","d.c","p.c","q.c","d.d","p.d","q.d")) for c.d.f. and quantile function of the composed distribution and the respective three panels for the absolutely continuous and the discrete part, respectively; |
| verticals | logical: if TRUE, draw vertical lines at steps; as in plot.stepfun |
| ngrid | integer: number of grid points used for plots of absolutely continuous distributions |
| cex.points | numeric; character expansion factor; as in plot.stepfun |
| mfColRow | shall default partition in panels be used – defaults to TRUE |
| lwd | a vector of line widths, see par. |
| ... | arguments to be passed to plot. |

## Value

invisible

## See Also

plot

## 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 (D, add = FALSE, regions = NULL, col = "gray", legend = TRUE,
    legend.pos = "topright", to.draw.arg = 1, verticals = FALSE,
    ngrid = 1000, cex.points = par("cex"), mfColRow = FALSE,
    lwd = par("lwd"), ...)
{
    dots <- match.call(call = sys.call(0), expand.dots = FALSE)$...
    if (!is.null(dots[["panel.first"]])) {
        pF <- .panel.mingle(dots, "panel.first")
    }
    else if (to.draw.arg == 1) {
        pF <- quote(abline(h = 0, col = "gray"))
    }
    else if (to.draw.arg == 2) {
        pF <- quote(abline(h = 0:1, col = "gray"))
    }
    else {
        pF <- NULL
    }
    dots$panel.first <- pF
    if (!add) {
        do.call(plot, c(list(D, to.draw.arg = to.draw.arg, cex.points = cex.points,
            mfColRow = mfColRow, verticals = verticals), dots))
    }
    discrete <- is(D, "DiscreteDistribution")
    if (discrete) {
        x <- support(D)
        if (hasArg("xlim")) {
            if (length(xlim) != 2)
                stop("Wrong length of Argument xlim")
            x <- x[(x >= xlim[1]) & (x <= xlim[2])]
        }
        if (!is.null(regions)) {
            col <- rep(col, length = length(regions))
            for (i in 1:length(regions)) {
                region <- regions[[i]]
                which.xs <- (x > region[1] & x <= region[2])
                xs <- x[which.xs]
                ps <- d(D)(x)[which.xs]
                lines(xs, ps, type = "h", col = col[i], lwd = 3 *
                  lwd, ...)
                points(xs, ps, pch = 16, col = col[i], cex = 2 *
                  cex.points, ...)
```

```
            }
            if (legend) {
                if (length(unique(col)) > 1) {
                  legend(legend.pos, title = if (length(regions) >
                    1)
                    "Regions"
                  else "Region", legend = sapply(regions, function(region) {
                    paste(round(region[1], 2), "to", round(region[2],
                      2))
                  }), col = col, pch = 15, pt.cex = 2.5, inset = 0.02)
                }
                else {
                  legend(legend.pos, title = if (length(regions) >
                    1)
                    "Regions"
                  else "Region", legend = sapply(regions, function(region) {
                    paste(round(region[1], 2), "to", round(region[2],
                      2))
                  }), inset = 0.02)
                }
            }
        }
    }
    else {
        lower0 <- getLow(D, eps = getdistrOption("TruncQuantile") *
            2)
        upper0 <- getUp(D, eps = getdistrOption("TruncQuantile") *
            2)
        me <- (distr::q.l(D))(1/2)
        s <- (distr::q.l(D))(3/4) - (distr::q.l(D))(1/4)
        lower1 <- me - 6 * s
        upper1 <- me + 6 * s
        lower <- max(lower0, lower1)
        upper <- min(upper0, upper1)
        dist <- upper - lower
        if (hasArg("xlim")) {
            if (length(xlim) != 2)
                stop("Wrong length of Argument xlim")
            x <- seq(xlim[1], xlim[2], length = ngrid)
        }
        else x <- seq(from = lower - 0.1 * dist, to = upper +
            0.1 * dist, length = ngrid)
        if (!is.null(regions)) {
            col <- rep(col, length = length(regions))
            for (i in 1:length(regions)) {
                region <- regions[[i]]
                which.xs <- (x >= region[1] & x <= region[2])
                xs <- x[which.xs]
                ps <- d(D)(x)[which.xs]
                xs <- c(xs[1], xs, xs[length(xs)])
                ps <- c(0, ps, 0)
                polygon(xs, ps, col = col[i])
            }
```

```
            if (legend) {
                if (length(unique(col)) > 1) {
                  legend(legend.pos, title = if (length(regions) >
                    1)
                    "Regions"
                  else "Region", legend = sapply(regions, function(region) {
                    paste(round(region[1], 2), "to", round(region[2],
                      2))
                  }), col = col, pch = 15, pt.cex = 2.5, inset = 0.02)
                }
                else {
                  legend(legend.pos, title = if (length(regions) >
                    1)
                    "Regions"
                  else "Region", legend = sapply(regions, function(region) {
                    paste(round(region[1], 2), "to", round(region[2],
                      2))
                  }), inset = 0.02)
                }
            }
        }
    }
    return(invisible(NULL))
}
```

---

`priceIndexNum` *Price index numbers*

---

## Description

`priceIndexNum` computes price indices given data on products over time (prices and quantities)

## Usage

```
priceIndexNum(x, prodID, pervar, pvar, qvar, base, indexMethod = "laspeyres",
              output = "fixedBase", ...)
```

## Arguments

| | |
|---|---|
| `x` | Data frame containing, at least, the characteristics (time, location, ...), the product identifiers, the prices and the quantities. |
| `prodID` | Character string for the name of the product identifier. |
| `pervar` | Character string for the name of the factor variable with the characteristics. |
| `pvar` | Character string for the name of the price variable. |
| `qvar` | Character string for the name of the quantity variable. |
| `base` | Character string for the name of the base characteristic. |

| | |
|---|---|
| indexMethod | Character vector to select the price index method. Tipical price index methods are laspeyres (default), paasche, and fisher, but it can also be use those in function priceIndex from package IndexNumR (dutot, carli, jevons, cswd, harmonic, tornqvist, satovartia, walsh and CES). |
| output | A character string specifying whether a chained (output="chained") , fixed base (output="fixedBase") or period-on-period (output="pop") price index numbers should be returned. Default is fixed base. |
| ... | Further arguments passed to or from other methods. |

## Details

priceIndexNum uses the function priceIndex from package IndexNumR without restricting the argument pervar from being integers starting at period 1 (base) and increasing in increments of 1 period.

## Value

priceIndexNum returns a data frame with one column with the characteristic variable plus as many columns as indexMethod selected:

| | |
|---|---|
| period | The characteristic variable. |
| laspeyres | The price index computed by the Laspeyres method. |
| paasche | The price index computed by the Paasche method. |
| fisher | The price index computed by the Fisher method. |
| ... | |

## See Also

priceIndex, Sindex, Deflat, ComplexIN.

## Examples

```
library(IndexNumR)
data(Prices, package = "RcmdrPlugin.TeachStat")

priceIndexNum(Prices, prodID = "prodID", pervar = "year", pvar = "price",
              qvar = "quantity", base = "2003",
              indexMethod = c("laspeyres", "paasche", "fisher"))
```

## Prices *Data for computing price indices.*

### Description

Data on the sold quantity and sale price of several products through some years.

It is used as an example for the use of the *Price index* window of the `RcmdrPlugin.TeachStat` package

### Usage

```
data("Prices")
```

### Format

A data frame with 15 observations on the following 4 variables.

`year` a factor representing the year

`prodID` a factor with the ID of the products

`price` the sale price

`quantity` the sold quantity

### Examples

```
data(Prices)
priceIndexNum (Prices, prodID ="prodID", pervar ="year", pvar="price",
          qvar ="quantity", base="2001", indexMethod =c("laspeyres", "paasche", "fisher"))
```

## RandomANOVA *One-Way ANOVA with random effects.*

### Description

In this menu the user can perform some calculations related to One-Way ANOVA with random effects.

This menu will call the functions for calculating the ANOVA table (`aov` from package `stats`) and the estimations of variance components using the maximum likelihood method and the REstricted Maximum Likelihood (REML) method (`lmer` from package `lme4`).

randomnessMenu *Randomness test*

## Description

In the "Nonparametric Tests" menu, two new entries are provided to perform the randomness test.

The first "Randomness test for two level factor..." can be used to contrast the randomness of a factor with two levels. This option use the function `runs.test` from `tseries` package. `runs.test`.

The second entry in the menu "Randomness test for numeric variable..." is used to test the randomness of a numerical variable. This option use the function `runs.test` from `randtest` package.

## Details

Here is an example of "Randomness test for a two level factor..." menu entry.

Load data "AMSsurvey" selecting from Rcmdr menu: "Data" -> "Data in packages" -> "Read data set from an attached package..." then double-click on "car", click on "AMSsurvey" and on "OK". Rcmdr reply with the following command in source pane (R Script)

```
data(AMSsurvey, package="car")
```

To make randomness test on variable "sex", select from Rcmdr menu: "Statistics" -> "Nonparametric tests" -> "Randomness test for two level factor..." select "sex" and "OK". Rcmdr reply with the following command in source pane (R Script)

```
with(AMSsurvey, twolevelfactor.runs.test(sex))
```

Here is an example of "Randomness test for a numeric variable..." menu entry.

Load data "sweetpotato" selecting from Rcmdr menu: "Data" -> "Data in packages" -> "Read data set from an attached package..." then double-click on "randtests", click on "sweetpotato" and on "OK". Rcmdr reply with the following command in source pane (R Script)

```
data(sweetpotato, package="randtests")
```

```
sweetpotato <- as.data.frame(sweetpotato)
```

To make randomness test on variable "yield", select from Rcmdr menu: "Statistics" -> "Nonparametric tests" -> "Randomness test for numeric variable..." select "yield" and "OK". Rcmdr reply with the following command in source pane (R Script)

```
with(sweetpotato, numeric.runs.test(yield))
```

## Author(s)

Manuel Munoz-Marquez <manuel.munoz@uca.es>

## See Also

For more information see `Rcmdr-package`.

Sindex *Simple index numbers*

## Description

Sindex returns a data frame with the index numbers with a given base. An index number measures changes in a variable with respect to a characteristic (time, location, ...)

Sindex can also be used for computing the base change of an index number.

## Usage

```
Sindex(x, pervar, vvar, base)
```

## Arguments

| | |
|---|---|
| x | Data frame containing, at least, a factor and a numeric variables. |
| pervar | Character string for the name of the factor variable with the characteristics. |
| vvar | Character string for the name of the numeric variable for which you want to calculate the index number or representing the index number for which you want to compute a base change. |
| base | Character string for the name of the base characteristic. |

## Value

Sindex returns a data frame with one column:

| | |
|---|---|
| index_base | The index number with base base |

## See Also

Deflat, ComplexIN, priceIndexNum.

## Examples

```
data(Depositos, package = "RcmdrPlugin.TeachStat")
Sindex(Depositos, "year", "quantity", "2006")
```

## Utilities RcmdrPlugin.TeachStat *Utility Functions*

### Description

`twoOrMoreLevelFactorsP()` returns TRUE if there is at least one factor in the active dataset that has two or more levels.

`twoOrMoreLevelFactors()` returns the object name of those factors that are active in the dataset that have at least two levels.

`listDistrs(class, env)` returns the object name of those distributions of class `class` (see package `distr`) in the `env` environment.

`DiscreteDistrsP()` returns TRUE if there is at least one distribution of class `DiscreteDistribution` (see `DiscreteDistribution`).

`AbscontDistrsP()` returns TRUE if there is at least one distribution of class `AbscontDistribution` (see `AbscontDistribution`).

### Usage

```
twoOrMoreLevelFactors()
twoOrMoreLevelFactorsP()
listDistrs(class = "UnivariateDistribution", envir = .GlobalEnv, ...)
DiscreteDistrsP()
AbscontDistrsP()
```

### Arguments

| | |
|---|---|
| `class` | string with the name of the class to be listed. |
| `envir` | string with the name of the environment. |
| `...` | further arguments. |

## VKM.test *Chi-square test for the variance of a Normal variable with known population mean.*

### Description

Under the assumption that the data come from a Normal distribution, it performs the hypothesis testing and the confidence interval for the variance with known population mean.

### Usage

```
VKM.test(x, sigma = 1, sigmasq = sigma^2, mu,
         alternative = c("two.sided", "less", "greater"), conf.level = 0.95,
         ...)
```

## Arguments

| | |
|---|---|
| x | a (non-empty) numeric vector of data values. |
| sigma | a number indicating the true value of the population standard deviation - Null hypothesis. |
| sigmasq | control argument. |
| mu | numerical value indicating the population mean assumed to be known (mandatory). |
| alternative | a character string specifying the alternative hypothesis, must be one of "two.sided" (default), "greater" or "less". You can specify just the initial letter. |
| conf.level | confidence level of the interval. |
| ... | further arguments to be passed to or from methods. |

## Value

A list with class "htest" containing the following components:

| | |
|---|---|
| statistic | the value of the ctest statistic. |
| parameter | the degrees of freedom for the test statistic. |
| p.value | the p-value for the test. |
| conf.int | confidence interval for variance with known population mean associated with the specified alternative hypothesis. |
| estimate | the estimated variance. |
| null.value | the specified hypothesized value of the variance. |
| alternative | a character string describing the alternative hypothesis. |
| method | a character string indicating what type of statistical test was performed. |
| data.name | a character string giving the name of the data. |

## See Also

VUM.test , var.test

## Examples

```
data(cars93) # Dataset provided with the package
# Variance of the maximum price (MaxPrice) assuming that the population mean
# price is known and equal to 22
VKM.test(cars93$MaxPrice, alternative="two.sided", sigma=11, mu=22, conf.level=0.95)
```

VUM.test *Chi-square test for the variance of a Normal variable with unknown population mean.*

## Description

Under the assumption that the data come from a Normal distribution, it performs the hypothesis testing and the confidence interval for the variance with unknown population mean.

## Usage

```
VUM.test(x, sigma = 1, sigmasq = sigma^2,
         alternative = c("two.sided", "less", "greater"), conf.level = 0.95,
         ...)
```

## Arguments

| | |
|---|---|
| x | a (non-empty) numeric vector of data values. |
| sigma | a number indicating the true value of the population standard deviation - Null hypothesis. |
| sigmasq | control argument. |
| alternative | a character string specifying the alternative hypothesis, must be one of "two.sided", "greater" o "less". You can specify just the initial letter. |
| conf.level | confidence level of the interval. |
| ... | further arguments to be passed to or from methods. |

## Value

A list with class "htest" containing the following components:

| | |
|---|---|
| statistic | the value of the test statistic |
| parameter | the degrees of freedom for the test statistic |
| p.value | the p-value for the test. |
| conf.int | confidence interval for variance with unknown population mean associated with the specified alternative hypothesis. |
| estimate | the estimated variance. |
| null.value | the specified hypothesized value of the variance. |
| alternative | a character string describing the alternative hypothesis. |
| method | a character string indicating what type of statistical test was performed. |
| data.name | a character string giving the name of the data. |

## See Also

VKM.test , var.test

## Examples

```
data(cars93) # Dataset provided with the package
# Variance of the maximum price (MaxPrice) assuming that the population mean
# price is unknown
VUM.test(cars93$MaxPrice, alternative="two.sided", sigma=11, conf.level=0.95)
```

---

W.numSummary *Summary statistics for weighted variables*

---

## Description

W.numSummary gives the main statistical summary for weighted variables (mean, standard deviation, coefficient of variation, skewness, kurtosis and quantiles). It also allows the partition of the data by a factor variable.

## Usage

```
W.numSummary(data,
             statistics = c("mean", "sd", "se(mean)", "IQR",
            "quantiles", "cv", "skewness", "kurtosis"),type = c("2", "1", "3"),
             quantiles = c(0, 0.25, 0.5, 0.75, 1),groups = NULL, weights)
```

## Arguments

| | |
|---|---|
| data | data.frame with the variables. |
| statistics | any of "mean", "sd", "se(mean)", "quantiles", "cv" (coefficient of variation - sd/mean), "skewness" or "kurtosis"; defaulting to c("mean", "sd", "quantiles", "IQR"). |
| type | definition to use in computing skewness and kurtosis; see the skewness and kurtosis functions in the **e1071** package. The default is "2". |
| quantiles | quantiles to report; by default is c(0, 0.25, 0.5, 0.75, 1). |
| groups | optional variable, typically a factor, to be used to partition the data. By default is NULL. |
| weights | numeric vector of weights. Zero values are allowed. |

## Details

W.numSummary performs a descriptive analysis of quantitative variables weighted (or not) by a numeric variable which determines the importance of each subject in the data frame. Optionally it allows the partition of the data by a factor variable (groups).

Note that, unlike the numSummary function, the sample standard deviation is calculated instead of the sample standard quasideviation.

## Value

An object with class "numSummary".

## See Also

numSummary, skewness, kurtosis.

## Examples

```
data(cars93)

# no weighted
W.numSummary(data=cars93[,c("CityMPG")], statistics =c("mean", "sd", "IQR", "quantiles"),
   quantiles = c(0,0.25,0.5,0.75,1), weights=NULL, groups=NULL)
# weighted
W.numSummary(data=cars93[,c("CityMPG")], statistics =c("mean", "sd", "IQR", "quantiles"),
   quantiles = c(0,0.25,0.5,0.75,1), weights=cars93$FuelCapacity, groups=NULL)
# no weighted
W.numSummary(data=cars93[,c("CityMPG")], statistics =c("mean", "sd", "IQR", "quantiles"),
   quantiles = c(0,0.25,0.5,0.75,1), weights=NULL, groups=cars93$Manual)
# weighted
bb <- W.numSummary(data=cars93[,c("CityMPG")], statistics =c("mean", "sd", "IQR", "quantiles"),
   quantiles = c(0,0.25,0.5,0.75,1), weights=cars93$FuelCapacity, groups=cars93$Manual)

bb
str(bb)
class(bb)
```

# Index