Package 'referenceIntervals'

February 20, 2015

Type Package

Title Reference Intervals

Version 1.1.1

Date 2014-09-12

Author Daniel Finnegan

Maintainer Daniel Finnegan <dan.finnegan@gmail.com>

Imports boot, extremevalues, car, outliers

LazyData no

Description This is a collection of tools to allow the medical professional to calculate appropriate reference ranges (intervals) with confidence intervals around the limits for diagnostic purposes.

License GPL-3

NeedsCompilation no

Repository CRAN

Date/Publication 2014-09-13 07:55:49

R topics documented:

referenceIntervals-package 2
cook.outliers 3
dixon.outliers 4
horn.outliers 5
nonparRanks 6
nonparRI 7
plot.interval 8
print.interval 10
print.interval.sub 11
refLimit 13
robust 15
set120 16
set20 17
set200 18

set50 18
sets.RData 19
singleRefLimit 19
tables.RData 22
vanderLoo.outliers 23

Index **25**

`referenceIntervals-package`

This package calculates reference intervals from a dataset using either parametric, non-parametric, or robust methods.

Description

This package also calculates the confidence intervals around the calculated reference intervals in order to provide a metric for how precise the calculations are. This package also contains four outlier detection functions.

Details

Package:	referenceIntervals
Type:	Package
Version:	1.1
Date:	2014-09-12
License:	GPL-3

Author(s)

Daniel Finnegan

Maintainer: Daniel Finnegan <dan.finnegan@gmail.com>

References

~~ Literature or other references for background information ~~

Examples

```
refLimit(set50, out.rm = TRUE, out.method = "cook")
refLimit(set200, out.method = "horn", RI = "n", refConf = 0.90, limitConf = 0.80)
horn.outliers(set120)
dixon.outliers(set20)

frame = data.frame(one = rnorm(30, m = 5, sd = 2), two = rnorm(30, m = 7, sd = 1),
three = rnorm(30, m = 2, sd = 0.5))
```

```
result = refLimit(frame)
plot(result)
```

`cook.outliers` *Determines outliers using Cook's Distance*

Description

A linear regression model is calculated for the data (which is the mean for one-dimensional data. From that, using the Cook Distances of each data point, outliers are determined and returned.

Usage

```
cook.outliers(data)
```

Arguments

`data`	A vector of data points.

Value

Returns a list containing a vector of outliers and a vector of the cleaned data (subset).

`outliers`	A vector of outliers from the data set
`subset`	A vector containing the remaining data, cleaned of outliers

Author(s)

Daniel Finnegan

Examples

```
cook.outliers(set50)
plot(cook.outliers(set50)$subset)

## The function is currently defined as
function (data)
{
    fit = lm(data ~ 1)
    cooks_dist = cooks.distance(fit)
    out = data[as.numeric(names(cooks_dist[cooks_dist > (4/length(cooks_dist)) |
        cooks_dist > 1]))]
    sub = data[as.numeric(names(cooks_dist[cooks_dist <= (4/length(cooks_dist)) &
        cooks_dist <= 1]))]
    return(list(outliers = out, subset = sub))
  }
```

`dixon.outliers` *Determines outliers using Dixon's Q Test method*

Description

This determines outliers of the dataset by calculating Dixon's Q statistic and comparing it to a standardized table of statistics. This method can only determine outliers for datasets of size 3 <= n <= 30. This function requires the outliers package.

Usage

```
dixon.outliers(data)
```

Arguments

`data`	A vector of data points.

Value

Returns a list containing a vector of outliers and a vector of the cleaned data (subset).

`outliers`	A vector of outliers from the data set
`subset`	A vector containing the remaining data, cleaned of outliers

Author(s)

Daniel Finnegan

References

Statistical treatment for rejection of deviant values: critical values of Dixon's "Q" parameter and related subrange ratios at the 95 (2), pp 139-146 DOI: 10.1021/ac00002a010. Publication Date: January 1991

One-sided and Two-sided Critical Values for Dixon's Outlier Test for Sample Sizes up to n = 30. Economic Quality Control, Vol 23(2008), No. 1, 5-13.

Examples

```
dixon.outliers(set20)
summary(dixon.outliers(set20)$subset)

## The function is currently defined as
function (data)
{
    if (length(data) >= 3 & length(data) <= 30) {
        d = sort(data)
        gap_high = abs(d[length(data)] - d[length(data) - 1])
        gap_low = abs(d[2] - d[1])
```

```
        range = d[length(d)] - d[1]
        end = NULL
        if (gap_high > gap_low) {
            end = "high"
            Q = gap_high/range
        }
        if (gap_low > gap_high) {
            end = "low"
            Q = gap_low/range
        }
        dixonNum = subset(dixonTableValues, Size == length(data))$Q95
        sub = data
        out = as.numeric(c())
        if (Q > dixonNum & end == "high") {
            sub = subset(data, data < d[length(d)])
            out = d[length(d)]
        }
        if (Q > dixonNum & end == "low") {
            sub = subset(data, data > d[1])
            out = d[1]
        }
        return(list(outliers = out, subset = sub))
    }
    else {
        return(list(outliers = as.numeric(c()), subset = data))
    }
}
```

`horn.outliers` *Determines outliers using Horn's method and Tukey's interquartile fences on a Box-Cox transformation of the data.*

Description

This function determines outliers in a Box-Cox transformed dataset using Horn's method of outlier detection using Tukey's interquartile fences. If a data point lies outside 1.5 * IQR from the 1st or 3rd quartile point, it is an outlier.

Usage

```
horn.outliers(data)
```

Arguments

`data`	A vector of data points.

Value

Returns a list containing a vector of outliers and a vector of the cleaned data (subset).

outliers	A vector of outliers from the data set
subset	A vector containing the remaining data, cleaned of outliers

Author(s)

Daniel Finnegan

References

ASVCP reference interval guidelines: determination of de novo reference intervals in veterinary species and other related topics. Vet Clin Pathol 41/4 (2012) 441-453, 2012 American Society for Veterinary Clinical Pathology

Horn, P. S., Feng, L., Li, Y., & Pesce, A. J. (2001). Effect of outliers and nonhealthy individuals on reference interval estimation. Clinical Chemistry, 47(12), 2137-2145.

Examples

```
horn.outliers(set200)

## The function is currently defined as
function (data)
{
    descriptives = summary(data)
    Q1 = descriptives[[2]]
    Q3 = descriptives[[5]]
    IQR = Q3 - Q1
    out = subset(data, data <= (Q1 - 1.5 * IQR) | data >= (Q3 +
        1.5 * IQR))
    sub = subset(data, data > (Q1 - 1.5 * IQR) & data < (Q3 +
        1.5 * IQR))
    return(list(outliers = out, subset = sub))
  }
```

---

nonparRanks *Table that dictate the ranks for the confidence intervals around the calculated reference interval.*

Description

This is a table that dictate the ranks for the confidence intervals around the calculated reference interval. This method is available when 120 <= n <= 1000.

Usage

```
nonparRanks
```

Format

A data frame with 882 observations on the following 3 variables.

`SampleSize` a numeric vector

`Lower` a numeric vector

`Upper` a numeric vector

References

Defining, Establishing, and Verifying Reference Intervals in the Clinical Laboratory; Approved Guideline - 3rd Edition (C28-A3)

Examples

```
data(nonparRanks)
```

`nonparRI` *Determines the reference interval using non-parametric means*

Description

This function uses the appropriate percentiles as determined by refConf to return the non-parametric reference interval. This is written as a boot function to use within the function refLimit.

Usage

```
nonparRI(data, indices = 1:length(data), refConf = 0.95)
```

Arguments

`data`	data is a vector of sample values.
`indices`	The indices of data to be used in the calculations. The default is to use the whole set.
`refConf`	refConf is a measure of the range covered by the calculation. Most often, as is the default, 95

Value

Returns a vector of two values, the lower and upper limits of the reference interval.

Author(s)

Daniel Finnegan

References

Defining, Establishing, and Verifying Reference Intervals in the Clinical Laboratory; Approved Guideline - 3rd Edition (C28-A3)

Examples

```
nonparRI(set50)
nonparRI(set50, refConf = 0.9)

## The function is currently defined as
function (data, indices = 1:length(data), refConf = 0.95)
{
    d = data[indices]
    results = c(quantile(d, (1 - refConf)/2, type = 6), quantile(d,
        1 - ((1 - refConf)/2), type = 6))
    return(results)
  }
```

`plot.interval` *Overload of plot function to include the ability to plot the results of refLimit*

Description

Plots the reference interval and confidence intervals around the limits of the reference interval.

Usage

```
## S3 method for class 'interval'
plot(x, main, ...)
```

Arguments

`x`	Object x is of type "interval".
`main`	Title of plot.
`...`	Arguments to be passed to methods, such as graphical parameters (see 'par').

Value

No return value.

Author(s)

Daniel Finnegan

Examples

```
result = refLimit(set200)
plot(result)

## The function is currently defined as
function (x, main = NULL, ...)
{
```

```
original.parameters = par()
if (class(x[[1]]) != "interval") {
    range = max(x[["Conf_Int"]][[4]]) - min(x[["Conf_Int"]][[1]])
    y_low = min(x[["Conf_Int"]][[1]]) - 0.05 * range
    y_high = max(x[["Conf_Int"]][[4]]) + 0.05 * range
    plot.new()
    plot.window(xlim = c(0, 2), ylim = c(y_low, y_high))
    segments(1, min(x[["Ref_Int"]]), 1, max(x[["Ref_Int"]]),
        col = "red")
    segments(1 - 0.05, x[["Conf_Int"]][[1]], 1 + 0.05, x[["Conf_Int"]][[1]],
        col = "blue")
    segments(1 - 0.05, x[["Conf_Int"]][[2]], 1 + 0.05, x[["Conf_Int"]][[2]],
        col = "blue")
    segments(1 - 0.05, x[["Conf_Int"]][[3]], 1 + 0.05, x[["Conf_Int"]][[3]],
        col = "blue")
    segments(1 - 0.05, x[["Conf_Int"]][[4]], 1 + 0.05, x[["Conf_Int"]][[4]],
        col = "blue")
    axis(1, at = 1:1, labels = x[["dname"]])
    axis(2)
    if (!is.null(main)) {
        title(main = main)
    }
    else {
        title(main = "Reference Range")
    }
    title(xlab = "Parameter")
    title(ylab = "Units")
    legend(x = "topright", col = c("red", "blue"), lty = 1,
        inset = c(0, -0.09), legend = c("Reference Interval",
            "Confidence Intervals"), cex = 0.75, xpd = TRUE)
    box()
}
if (class(x[[1]]) == "interval") {
    numRanges = length(x)
    intervals = unlist(sapply(x, "[", "Conf_Int"))
    labels = unlist(sapply(x, "[", "dname"))
    range = max(intervals) - min(intervals)
    y_low = min(intervals) - 0.05 * range
    y_high = max(intervals) + 0.05 * range
    plot.new()
    plot.window(xlim = c(0, numRanges + 1), ylim = c(y_low,
        y_high))
    for (i in 1:numRanges) {
        segments(i, x[[i]]$Ref_Int[1], i, x[[i]]$Ref_Int[2],
            col = "red")
        segments(i - 0.05, x[[i]]$Conf_Int[[1]], i + 0.05,
            x[[i]]$Conf_Int[[1]], col = "blue")
        segments(i - 0.05, x[[i]]$Conf_Int[[2]], i + 0.05,
            x[[i]]$Conf_Int[[2]], col = "blue")
        segments(i - 0.05, x[[i]]$Conf_Int[[3]], i + 0.05,
            x[[i]]$Conf_Int[[3]], col = "blue")
        segments(i - 0.05, x[[i]]$Conf_Int[[4]], i + 0.05,
            x[[i]]$Conf_Int[[4]], col = "blue")
```

```
        }
        axis(1, at = 1:numRanges, labels = FALSE)
        text(x = seq(1, numRanges, by = 1), par("usr")[3] - 0.2,
            labels = labels, cex = 0.75, srt = 90, pos = 1, offset = 2,
            xpd = TRUE)
        axis(2)
        if (!is.null(main)) {
            title(main = main)
        }
        else {
            title(main = "Reference Range")
        }
        title(xlab = "Parameter")
        title(ylab = "Units")
        legend(x = "topright", col = c("red", "blue"), lty = 1,
            inset = c(0, -0.09), legend = c("Reference Interval",
                "Confidence Intervals"), cex = 0.75, xpd = TRUE)
        box()
    }
    par(original.parameters[-c(13, 19, 21:23)])
}
```

`print.interval` *Overload of print in order to concisely print the results of refLimit*

Description

This function allows for the pretty-printing of a large list object created by calling the refLimit function.

Usage

```
## S3 method for class 'interval'
print(x, digits = 4L, quote = TRUE, prefix = "", ...)
```

Arguments

`x`	x is an object of type "interval"
`digits`	minimal number of _significant_ digits. See 'print.default'.
`quote`	logical, indicating whether or not strings should be printed with surrounding quotes.
`prefix`	Option to specify a formatting prefix.
`...`	further arguments passed to or from other methods.

Value

No return value.

Author(s)

Daniel Finnegan

Examples

```
result = refLimit(set120)
result

## The function is currently defined as
function (x, digits = 4L, quote = TRUE, prefix = "", ...)
{
    if (class(x[[1]]) == "interval") {
        lapply(x, print.interval.sub)
    }
    else {
        print.interval.sub(x)
    }
  }
```

---

print.interval.sub *Overload of print in order to concisely print the results of refLimit*

Description

This function allows for the pretty-printing of a large list object created by calling the refLimit function.

Usage

```
## S3 method for class 'interval.sub'
print(x, digits = 4L, quote = TRUE, prefix = "", ...)
```

Arguments

x	x is an object of type "interval"
digits	minimal number of _significant_ digits. See 'print.default'.
quote	logical, indicating whether or not strings should be printed with surrounding quotes.
prefix	Option to specify a formatting prefix.
...	further arguments passed to or from other methods.

Value

No return value.

Author(s)

Daniel Finnegan

Examples

```
## The function is currently defined as
function (x, digits = 4L, quote = TRUE, prefix = "", ...)
{
    cat("\n")
    cat(strwrap(x$methodRI, prefix = "\t"), sep = "\n")
    cat(strwrap(x$methodCI, prefix = "\t"), sep = "\n")
    cat("\n")
    cat("data:  ", x$dname, "\n", sep = "")
    cat("N: ", x$size, "\n", sep = "")
    if (!is.null(x$refConf)) {
        cat(format(100 * x$refConf), "% Reference Interval",
            "\n", sep = "")
    }
    if (!is.null(x$limitConf)) {
        cat(format(100 * x$limitConf), "% Confidence Intervals\n",
            "\n", sep = "")
    }
    if (!is.null(x$out.method)) {
        cat("Outlier detection method:", x$out.method, "\n")
    }
    if (!is.null(x$outliers) & length(x$outliers) > 0) {
        if (x$out.rm) {
            cat("Removed outliers: ")
        }
        else {
            cat("Suspected outliers: ")
        }
        cat(strwrap(paste(format(x$outliers, digits = 6), collapse = ", ")),
            sep = "\n")
    }
    else {
        cat("No outliers detected\n")
    }
    if (!is.null(x$norm)) {
        cat(strwrap(x$norm, prefix = "\n"), "\n\n")
    }
    if (!is.null(x$Ref_Int)) {
        cat("\nReference Interval: ")
        cat(strwrap(paste(format(x$Ref_Int, digits = 6), collapse = ", ")),
            sep = "\n")
    }
    if (!is.null(x$Conf_Int)) {
        cat("Lower Confidence Interval: ")
        cat(strwrap(paste(format(x$Conf_Int[1:2], digits = 6),
            collapse = ", ")), sep = "\n")
        cat("Upper Confidence Interval: ")
        cat(strwrap(paste(format(x$Conf_Int[3:4], digits = 6),
            collapse = ", ")), sep = "\n")
    }
    cat("\n")
```

```
  invisible(x)
}
```

`refLimit` *Calculates and returns reference and confidence intervals for a dataset*

Description

This function calculates a reference interval from a dataset using parametric, non-parametric, or robust methods.

Usage

```
refLimit(data, out.method = "horn", out.rm = FALSE, RI = "p", CI = "p",
refConf = 0.95, limitConf = 0.9)
```

Arguments

`data`	A vector of data points.
`out.method`	The outlier detection method. Valid options include "horn", "cook", "dixon", and "vanderLoo".
`out.rm`	Remove outliers. If value is TRUE, outliers will be automatically removed prior to calculations. If FALSE (default), outliers will be detected but not removed.
`RI`	Method for reference interval calculations. Valid options include "p" (default) for parametric, "n" for non-parametric, and "r" for robust method.
`CI`	Method for confidence interval calculations. Valid options include "p" for parametric (default), "n" for non-parametric, and "boot" for bootstrapping method. The minimum samplesize for non-parametric confidence interval calculations is 120. With smaller samples, bootstrapping methods are used.
`refConf`	Desired coverage for the calculated reference interval. The default is a 95 interval.
`limitConf`	Desired confidence interval level. The default is a 90 reference interval limits.

Details

A confidence interval around each limit of the reference interval is calculated as a metric for determining the validity of the result. Outliers can be detected in one of four different methods and automatically eliminated.

Value

Returns a list of necessary information.

`size`	Size of dataset
`dname`	Name of dataset

out.method	Outlier detection method
out.rm	Boolean indicating whether outliers are automatically removed
outliers	Vector of detected outliers
methodRI	Method for reference interval calculations (p, n, or r)
methodCI	Method for confidence interval calculations (p, n, boot)
norm	Results of running Shapiro-Wilk and Kolmorgorov-Smirnov normacy tests
refConf	Desired coverage of reference interval
limitConf	Desired confidence interval level
Ref_Int	List containing the reference interval and confidence interval values

Author(s)

Daniel Finnegan

References

ASVCP reference interval guidelines: determination of de novo reference intervals in veterinary species and other related topics. Vet Clin Pathol 41/4 (2012) 441-453, 2012. American Society for Veterinary Clinical Pathology

Examples

```
refLimit(set20, out.method = "dixon")
refLimit(set200, out.method = "cook", out.rm = TRUE, RI = "n", refConf = 0.9)

## The function is currently defined as
function (data, out.method = "horn", out.rm = FALSE, RI = "p",
    CI = "p", refConf = 0.95, limitConf = 0.9)
{
    cl = class(data)
    if (cl == "data.frame") {
        frameLabels = colnames(data)
        dname = deparse(substitute(data))
        result = lapply(data, singleRefLimit, dname, out.method,
            out.rm, RI, CI, refConf, limitConf)
        for (i in 1:length(data)) {
            result[[i]]$dname = frameLabels[i]
        }
        class(result) = "interval"
    }
    else {
        frameLabels = NULL
        dname = deparse(substitute(data))
        result = singleRefLimit(data, dname, out.method, out.rm,
            RI, CI, refConf, limitConf)
    }
    return(result)
  }
```

robust *Algorithm that implements the robust method for reference interval calculations*

Description

The robust method is an iterative method that determines the most appropriate weighted mean of the data and then calculates the desired reference interval.

Usage

```
robust(data, indices = c(1:length(data)), refConf = 0.95)
```

Arguments

data	Vector of data.
indices	Indices of data to use for calculations.
refConf	Desired coverage of the reference interval. Default is 95 interval.

Value

Returns a vector containing the lower and upper limits of the reference interval.

Author(s)

Daniel Finnegan

References

Defining, Establishing, and Verifying Reference Intervals in the Clinical Laboratory; Approved Guideline - 3rd Edition (C28-A3)

Examples

```
robust(set50)
robust(horn.outliers(set20)$subset)

## The function is currently defined as
function (data, indices = c(1:length(data)), refConf = 0.95)
{
    data = sort(data[indices])
    n = length(data)
    median = summary(data)[[3]]
    Tbi = median
    TbiNew = 10000
    c = 3.7
    MAD = summary(abs(data - median))[[3]]
    MAD = MAD/0.6745
    smallDiff = FALSE
```

```
    repeat {
        ui = (data - Tbi)/(c * MAD)
        ui[ui < -1] = 1
        ui[ui > 1] = 1
        wi = (1 - ui^2)^2
        TbiNew = (sum(data * wi)/sum(wi))
        if ((abs(TbiNew - Tbi)) < 1e-06) {
            break
        }
        Tbi = TbiNew
    }
    ui = NULL
    ui = (data - median)/(205.6 * MAD)
    sbi205.6 = 205.6 * MAD * sqrt((n * sum(((1 - ui[ui > -1 &
        ui < 1]^2)^4) * ui[ui > -1 & ui < 1]^2))/(sum((1 - ui[ui >
        -1 & ui < 1]^2) * (1 - 5 * ui[ui > -1 & ui < 1]^2)) *
        max(c(1, -1 + sum((1 - ui[ui > -1 & ui < 1]^2) * (1 -
            5 * ui[ui > -1 & ui < 1]^2))))))
    ui = NULL
    ui = (data - median)/(3.7 * MAD)
    sbi3.7 = 3.7 * MAD * sqrt((n * sum(((1 - ui[ui > -1 & ui <
        1]^2)^4) * ui[ui > -1 & ui < 1]^2))/(sum((1 - ui[ui >
        -1 & ui < 1]^2) * (1 - 5 * ui[ui > -1 & ui < 1]^2)) *
        max(c(1, -1 + sum((1 - ui[ui > -1 & ui < 1]^2) * (1 -
            5 * ui[ui > -1 & ui < 1]^2))))))
    ui = NULL
    ui = (data - Tbi)/(3.7 * sbi3.7)
    St3.7 = 3.7 * sbi3.7 * sqrt((sum(((1 - ui[ui > -1 & ui <
        1]^2)^4) * ui[ui > -1 & ui < 1]^2))/(sum((1 - ui[ui >
        -1 & ui < 1]^2) * (1 - 5 * ui[ui > -1 & ui < 1]^2)) *
        max(c(1, -1 + sum((1 - ui[ui > -1 & ui < 1]^2) * (1 -
            5 * ui[ui > -1 & ui < 1]^2))))))
    tStatistic = qt(1 - ((1 - refConf)/2), (n - 1))
    margin = tStatistic * sqrt(sbi205.6^2 + St3.7^2)
    robustLower = Tbi - margin
    robustUpper = Tbi + margin
    RefInterval = c(robustLower, robustUpper)
    return(RefInterval)
}
```

`set120` *Dataset containing 120 values*

Description

Small dataset containing 120 samples. The mean is centered on 27 with a standard deviation of 7.

Usage

```
set120
```

Format

The format is: num [1:120] 38.1 12.6 31.3 35.5 22.6 ...

Source

```
rnorm(120, m = 27, sd = 7)
```

Examples

```
data(set120)
```

---

set20 *Small dataset containing 20 samples*

Description

Small dataset containing 20 samples. The mean is centered on 42 with a standard deviation of 5.

Usage

```
set20
```

Format

The format is: num [1:20] 35 32.9 43.6 44.6 35.9 ...

Source

```
rnorm(20, m = 42, sd = 6)
```

Examples

```
data(set20)
```

set200 *Dataset containing 200 values*

Description

Small dataset containing 200 samples. The mean is centered on 5 with a standard deviation of 1.

Usage

```
set200
```

Format

The format is: num [1:200] 3.95 5.16 5.32 3.86 3.54 ...

Source

```
rnorm(200, m = 5, sd = 1)
```

Examples

```
data(set200)
```

set50 *Dataset containing 50 values*

Description

Small dataset containing 50 samples. The mean is centered on 14 with a standard deviation of 3.

Usage

```
set50
```

Format

The format is: num [1:50] 16.61 20.43 7.91 15.19 14.77 ...

Source

```
rnorm(50, m = 14, sd = 3)
```

Examples

```
data(set50)
```

sets.RData *Collection of sets*

Description

Four different datasets of sizes 20, 50, 120, and 200.

Usage

```
sets.RData
```

Format

The format is: chr "sets.RData"

Examples

```
data(sets.RData)
```

singleRefLimit *This is the workhorse of the refLimit function*

Description

This is the function called to work on each individual vector of data.

Usage

```
singleRefLimit(data, dname = "default", out.method = "horn", out.rm = FALSE,
RI = "p", CI = "p", refConf = 0.95, limitConf = 0.9)
```

Arguments

data	A vector of data points.
dname	Name of dataset.
out.method	The outlier detection method. Valid options include "horn", "cook", "dixon", and "vanderLoo".
out.rm	Remove outliers. If value is TRUE, outliers will be automatically removed prior to calculations. If FALSE (default), outliers will be detected but not removed.
RI	Method for reference interval calculations. Valid options include "p" (default) for parametric, "n" for non-parametric, and "r" for robust method.
CI	Method for confidence interval calculations. Valid options include "p" for parametric (default), "n" for non-parametric, and "boot" for bootstrapping method. The minimum sample size for non-parametric confidence interval calculations is 120. With smaller samples, bootstrapping methods are used.

refConf	Desired coverage for the calculated reference interval. The default is a 95
limitConf	Desired confidence interval level. The default is a 90 confidence interval around the reference interval limits.

Value

Returns a list of necessary information.

size	Size of dataset
dname	Name of dataset
out.method	Method of outlier detection
out.rm	Boolean indicating whether outliers are automatically removed
outliers	Vector of detected outliers
methodRI	Method for reference interval calculations (p, n, or r)
methodCI	Method for confidence interval calculations (p, n, boot)
norm	Results of running Shapiro-Wilk and Kolmorgorov-Smirnov normacy tests
refConf	Desired coverage of reference interval
limitConf	Desired confidence interval level
Ref_Int	List containing the reference interval and confidence interval values

Author(s)

Daniel Finnegan

Examples

```
singleRefLimit(set200, out.method = "horn", out.rm = TRUE)

## The function is currently defined as
function (data, dname = "default", out.method = "horn", out.rm = FALSE,
    RI = "p", CI = "p", refConf = 0.95, limitConf = 0.9)
{
    if (out.method == "dixon") {
        output = dixon.outliers(data)
    }
    else if (out.method == "cook") {
        output = cook.outliers(data)
    }
    else if (out.method == "vanderLoo") {
        output = vanderLoo.outliers(data)
    }
    else {
        output = horn.outliers(data)
    }
    if (out.rm == TRUE) {
        data = output$subset
    }
    outliers = output$outliers
```

```
n = length(data)
mean = mean(data, na.rm = TRUE)
sd = sd(data, na.rm = TRUE)
norm = NULL
if (RI == "n") {
    methodRI = "Reference Interval calculated nonparametrically"
    data = sort(data)
    holder = nonparRI(data, indices = 1:length(data), refConf)
    lowerRefLimit = holder[1]
    upperRefLimit = holder[2]
    if (CI == "p") {
        CI = "n"
    }
}
if (RI == "r") {
    methodRI = "Reference Interval calculated using Robust algorithm"
    holder = robust(data, 1:length(data), refConf)
    lowerRefLimit = holder[1]
    upperRefLimit = holder[2]
    CI = "boot"
}
if (RI == "p") {
    methodRI = "Reference Interval calculated parametrically"
    methodCI = "Confidence Intervals calculated parametrically"
    refZ = qnorm(1 - ((1 - refConf)/2))
    limitZ = qnorm(1 - ((1 - limitConf)/2))
    lowerRefLimit = mean - refZ * sd
    upperRefLimit = mean + refZ * sd
    se = sqrt(((sd^2)/n) + (((refZ^2) * (sd^2))/(2 * n)))
    lowerRefLowLimit = lowerRefLimit - limitZ * se
    lowerRefUpperLimit = lowerRefLimit + limitZ * se
    upperRefLowLimit = upperRefLimit - limitZ * se
    upperRefUpperLimit = upperRefLimit + limitZ * se
    shap_normalcy = shapiro.test(data)
    shap_output = paste(c("Shapiro-Wilk: W = ", format(shap_normalcy$statistic,
        digits = 6), ", p-value = ", format(shap_normalcy$p.value,
        digits = 6)), collapse = "")
    ks_normalcy = suppressWarnings(ks.test(data, "pnorm",
        m = mean, sd = sd))
    ks_output = paste(c("Kolmorgorov-Smirnov: D = ", format(ks_normalcy$statistic,
        digits = 6), ", p-value = ", format(ks_normalcy$p.value,
        digits = 6)), collapse = "")
    if (shap_normalcy$p.value < 0.05 | ks_normalcy$p.value <
        0.05) {
        norm = list(shap_output, ks_output)
    }
    else {
        norm = list(shap_output, ks_output)
    }
}
if (CI == "n") {
    if (n < 120) {
        cat("\nSample size too small for non-parametric confidence intervals,
```

```
            bootstrapping instead\n")
            CI = "boot"
        }
        else {
            methodCI = "Confidence Intervals calculated nonparametrically"
            ranks = subset(nonparRanks, subset = (nonparRanks$SampleSize ==
                n))
            lowerRefLowLimit = data[ranks$Lower]
            lowerRefUpperLimit = data[ranks$Upper]
            upperRefLowLimit = data[(n + 1) - ranks$Upper]
            upperRefUpperLimit = data[(n + 1) - ranks$Lower]
        }
    }
    if (CI == "boot" & (RI == "n" | RI == "r")) {
        methodCI = "Confidence Intervals calculated by bootstrapping, R = 5000"
        if (RI == "n") {
            bootresult = boot(data = data, statistic = nonparRI,
                refConf = refConf, R = 5000)
        }
        if (RI == "r") {
            bootresult = boot(data = data, statistic = robust,
                refConf = refConf, R = 5000)
        }
        bootresultlower = boot.ci(bootresult, conf = limitConf,
            type = "basic", index = 1)
        bootresultupper = boot.ci(bootresult, conf = limitConf,
            type = "basic", index = 2)
        lowerRefLowLimit = bootresultlower$basic[4]
        lowerRefUpperLimit = bootresultlower$basic[5]
        upperRefLowLimit = bootresultupper$basic[4]
        upperRefUpperLimit = bootresultupper$basic[5]
    }
    RVAL = list(size = n, dname = dname, out.method = out.method,
        out.rm = out.rm, outliers = outliers, methodRI = methodRI,
        methodCI = methodCI, norm = norm, refConf = refConf,
        limitConf = limitConf, Ref_Int = c(lowerRefLimit = lowerRefLimit,
            upperRefLimit = upperRefLimit),
            Conf_Int = c(lowerRefLowLimit = lowerRefLowLimit,
            lowerRefUpperLimit = lowerRefUpperLimit,
            upperRefLowLimit = upperRefLowLimit,
            upperRefUpperLimit = upperRefUpperLimit))
    class(RVAL) = "interval"
    return(RVAL)
}
```

`tables.RData` *Two reference tables necessary for reference interval calculations*

Description

Table for non-parametric ranks of confidence intervals.

Usage

```
tables.RData
```

Format

The format is: chr "tables.RData"

Examples

```
data(tables.RData)
```

`vanderLoo.outliers` *Mark van der Loo's outlier detection method in the extremevalues package*

Description

Separates data into vectors of outliers and a cleaned subset of the data.

Usage

```
vanderLoo.outliers(data)
```

Arguments

`data`	Vector of data values.

Value

Returns a list containing a vector of outliers and a vector of the cleaned data (subset).

`outliers`	A vector of outliers from the data set
`subset`	A vector containing the remaining data, cleaned of outliers

Note

Requires extremevalues package.

Author(s)

Daniel Finnegan

References

http://cran.r-project.org/web/packages/extremevalues/extremevalues.pdf

Examples

```
vanderLoo.outliers(set50)
vanderLoo.outliers(set200)

## The function is currently defined as
function (data)
{
    result = getOutliers(data, method = "I")
    indices = c(result$iLeft, result$iRight)
    out = data[indices]
    sub = data[!data %in% out]
    return(list(outliers = out, subset = sub))
  }
```

Index

∗Topic **\textasciitildeCook**
cook.outliers, 3
∗Topic **\textasciitildeDixon**
dixon.outliers, 4
∗Topic **\textasciitildeHorn**
horn.outliers, 5
∗Topic **\textasciitildeinterval**
nonparRI, 7
plot.interval, 8
print.interval, 10
refLimit, 13
robust, 15
singleRefLimit, 19
∗Topic **\textasciitildekwd1**
print.interval.sub, 11
∗Topic **\textasciitildekwd2**
print.interval.sub, 11
∗Topic **\textasciitildenonparametric**
nonparRI, 7
∗Topic **\textasciitildeoutliers**
vanderLoo.outliers, 23
∗Topic **\textasciitildeoutlier**
cook.outliers, 3
dixon.outliers, 4
horn.outliers, 5
∗Topic **\textasciitildeplot**
plot.interval, 8
∗Topic **\textasciitildeprint**
print.interval, 10
∗Topic **\textasciitilderange**
nonparRI, 7
∗Topic **\textasciitildereference**
refLimit, 13
singleRefLimit, 19
∗Topic **\textasciitilderobust**
robust, 15
∗Topic **\textasciitildevanderLoo**
vanderLoo.outliers, 23
∗Topic **datasets**
nonparRanks, 6
set120, 16
set20, 17
set200, 18
set50, 18
sets.RData, 19
tables.RData, 22
∗Topic **package**
referenceIntervals-package, 2

cook.outliers, 3

dixon.outliers, 4

horn.outliers, 5

nonparRanks, 6
nonparRI, 7

plot.interval, 8
print.interval, 10
print.interval.sub, 11

referenceIntervals
(referenceIntervals-package), 2
referenceIntervals-package, 2
refLimit, 13
robust, 15

set120, 16
set20, 17
set200, 18
set50, 18
sets.RData, 19
singleRefLimit, 19

tables.RData, 22

vanderLoo.outliers, 23