Package 'Ecfun'

July 2, 2014

Version 0.1-0

Date 2013-10-13

Title Functions for Ecdat

Author Spencer Graves <spencer.graves@effectivedefense.org>

Maintainer Spencer Graves <spencer.graves@effectivedefense.org>

Depends R (>= 3.0.1)

Suggests car, systemfit, sem, lmtest, sandwich, gdata, RCurl, XML,tis, Ecdat, fda

Description Functions to update datasets in Ecdat and to create,manipulate, plot and analyze those and similar datasets.

LazyData true

License GPL (>= 2)

URL http://www.r-project.org

Repository CRAN

Repository/R-Forge/Project ecdat

Repository/R-Forge/Revision 157

Repository/R-Forge/DateTimeStamp 2014-04-07 04:08:00

Date/Publication 2014-04-08 15:25:29

NeedsCompilation no

R topics documented:

`as.Date1970` *Date from a number of days since the start of 1970.*

Description

`as.Date.numeric` requires orgin to be specified. The present function assumes that this origin is January 1, 1970.

Usage

```
as.Date1970(x, ...)
```

Arguments

x	a numeric vector of dates in days since the start of 1970.
...	optional arguments to pass to as.Date.

Value

Returns a vector of Dates

Author(s)

Spencer Graves

See Also

as.Date as.POSIXct1970

Examples

```
days <- c(0, 1, 365)
Dates <- as.Date1970(days)


all.equal(c('1970-01-01', '1970-01-02', '1971-01-01'),
          as.character(Dates))



all.equal(days, as.numeric(Dates))
```

---

asNumericDF *Coerce to numeric dropping commas and info after a blank*

Description

Delete commas (thousand separators) and drop information after a blank, then coerce to numeric and order the rows by the orderBy. Some Excel imports include commas as thousand separators; this replaces any commas with char(0), ". Also, some character data includes footnote references following the year. Table F-1 from the US Census Bureau needs all three of these features: It needs orderBy, because the most recent year appears first, just the opposite of most other data sets where the most recent year appears last. It has footnote references following a character string indicating the year. And it includes commas as thousand separators.

Usage

```
asNumericChar(x)
asNumericDF(x, keep=function(x)any(!is.na(x)), orderBy)
```

Arguments

x	For asNumericChar, this is a character vector to be converted to numeric after gsub(',', '', x). For asNumericDF, this is a data.frame with all character columns to be converted to numerics.
keep	something to indicate which columns to keep
orderBy	Which columns to order the rows of x[, keep] by

Details

1. Replace commas by nothing
2. strsplit on ' ' and take only the first part, thereby eliminating the footnote references.
3. Replace any blanks with NAs
4. as.numeric
5. lapply(x, 1-4)
6. order the rows; by default, ascending on the first column

Value

all numeric data.frame

Author(s)

Spencer Graves

See Also

scan gsub Quotes

Examples

```
fakeF1 <- data.frame(yr=c('1948', '1947 (1)'),
                     q1=c('1,234', ''), duh=rep(NA, 2) )
nF1 <- asNumericDF(fakeF1)

nF1. <- data.frame(yr=asNumericChar(fakeF1$yr),
                   q1=asNumericChar(fakeF1$q1))[2:1,]

# correct answer
row.names(nF1.) <- 2:1

nF1c <- data.frame(yr=1947:1948, q1=c(NA, 1234))
row.names(nF1c) <- 2:1


all.equal(nF1, nF1.)
```

```
all.equal(nF1, nF1c)
```

`camelParse` *Split a character string where a capital letter follows a lowercase letter*

Description

Split a character string where a capital letter follows a lowercase letter.

Usage

```
camelParse(x, except=c('De', 'Mc', 'Mac'))
```

Arguments

`x`	a character vector
`except`	character vector giving exceptions: If any of these are found, ignore and look for the next one

Details

Find all places where a lowercase letter is followed by a capital.

Split on those points

Value

list of character vectors

Author(s)

Spencer Graves

See Also

`strsplit`

Examples

```
tst <- c('Smith, JohnJohn Smith',
         'EducationNational DefenseOther Committee',
         'McCain, JohnJohn McCain')
tst. <- camelParse(tst)


all.equal(tst., list(c('Smith, John', 'John Smith'),
    c('Education', 'National Defense', 'Other Committee'),
    c('McCain, John', 'John McCain') ) )
```

financialCrisisFiles *Files containing financial crisis data*

Description

FinancialCrisisFiles in Ecdat is an object of class financialCrisisFiles created by the financialCrisisFiles function to describe files containing data on financial crises downloadable from http://www.reinhartandrogoff.com/data/browse-by-topic/topics/7/.

Usage

```
financialCrisisFiles(files=c("22_data.xls", "23_data.xls",
       "Varieties_Part_III.xls", "25_data.xls"), ...)
```

Arguments

files	character vector of file names
...	arguments to pass with file and sheet name to read.xls when reading a sheet of an MS Excel file. This is assumed to be the same for all sheets of all files. If this is not the case, the resulting financialCrisisFiles object will have to be edited manually before using it to read the data.

Details

Reinhart and Rogoff (http://www.reinhartandrogoff.com) provide numerous data sets analyzed in their book, "This Time Is Different: Eight Centuries of Financial Folly". Of interest here are data on financial crises of various types for 70 countries spanning the years 1800 - 2010, downloadable from http://www.reinhartandrogoff.com/data/browse-by-topic/topics/7/.

The function financialCrisisFiles produces a list of class financialCrisisFiles describing four different Excel files in very similar formats with one sheet per Country and a few extra descriptor sheets. The data object FinancialCrisisFiles is the default output of that function.

It does this in several steps:

1. Read the first sheet of each file
2. Extract the names of the Countries from that first sheet.
3. Elimiate any blank spaces in the names to convert, e.g., "Costa Rica" to "CostaRica".
4. Find the sheets corresponding to each of the compressed names.
5. Construct the output list.

Value

The function financialCrisisFiles returns a list of class financialCrisisFiles. This is a list with components carrying the names of files to be read. Each component is a list of optional arguments to pass to do.call(read.xls, ...) to read the sheet with name = name of that component.

The default value returned by `financialCrisisFiles` is the data object `FinancialCrisisFiles`. This corresponds to the files downloaded from http://www.reinhartandrogoff.com/data/browse-by-topic/topics/7/ in January 2013 (except for the fourth, which was not available there because of an error with the web site but instead was obtained directly from Prof. Reinhart).

Author(s)

Spencer Graves

Source

http://www.reinhartandrogoff.com

References

Carmen M. Reinhart and Kenneth S. Rogoff (2009) This Time Is Different: Eight Centuries of Financial Folly, Princeton U. Pr.

See Also

read.xls

Examples

```
Ecdat.demoFiles <- system.file('demoFiles', package='Ecdat')
Ecdat.xls <- dir(Ecdat.demoFiles, pattern='xls$',
                 full.names=TRUE)
tst <- financialCrisisFiles(Ecdat.xls)

## Not run:
# check
\dontshow{stopifnot(}
all.equal(tst, data(FinancialCrisisFiles))
\dontrun{)}

## End(Not run)
```

---

grepNonStandardCharacters

grep for nonstandard characters

Description

Return the indices of elements of `x` containing characters that are not in `standardCharacters`.

Usage

```
grepNonStandardCharacters(x, value=FALSE,
   standardCharacters=c(letters, LETTERS, ' ','.', ',', 0:9,
      '\"', "\'", '-', '_', '(', ')', '[', ']', '\n'),
   ... )
```

Arguments

x	character vector in which it is desired to identfy elements containing characters not in standardCharacters.
value	logical: TRUE to return the values found in x, FALSE to return their indices.
standardCharacters	Characters to overlook in x to identify anything not in standardCharacters.
...	optional arguments for regexpr

Details

1. x. <- strsplit(x, ''): convert the input character vector to a list of vectors of character vectors with nchar(x.[i]) == 1 for i in 1:length(x).

2. sapply(x., ...) to identify all elements for which any element of x[[i]] is not in standardCharacters.

Value

an integer vector identifying all elements of x containing a character not in standardCharacters.

Author(s)

Spencer Graves

See Also

grep, regexpr, subNonStandardCharacters, showNonASCII

Examples

```
Names <- c('Raul', 'Ra`l', 'Torres,Raul', 'Torres, Raul')
#  confusion in character sets can create
#  names like Names[2]

chk <- grepNonStandardCharacters(Names)


all.equal(chk, 2)


chkv <- grepNonStandardCharacters(Names, TRUE)


all.equal(chkv, 'Ra`l')
```

match.data.frame *Identify the row of* y *best matching each row of* x

Description

For each row of x[, by.x], find the best matching row of y[, by.y], with the best match defined by grep. and split.

grep. and split must either be missing or have the same length as by.x and by.y. If grep.[i] and split[i] are NA, do a complete match of x[, by.x[i]] and y[, by.y[i]]. Otherwise, for each row j, look for a match for strsplit(x[j, by.x[i]], split[i])[[1]][1] among strsplit(y[, by.y[i]], split[i]). See details.

Usage

```
match.data.frame(x, y, by, by.x=by, by.y=by, grep., split, sep=':')
```

Arguments

x, y	data.frames
by, by.x, by.y	names of columns of x and y to match.
grep.	a character vector of the type of match for each element of by.x and by.y. If NA, require a perfect match. Alternatives are grep and agrep to find a match for the first segment in strsplit(x, split=split[i]) among any of the segments of strsplit(y, split=split[i]). Use fixed=TRUE with the calls to these functions. NOTE: These alternatives are not examined if a unique match is found betweed x[, by.x[is.na(grep.) & is.na(split)]] and the corresponding columns of y.
split	A character vector of split characters to pass to strsplit; strsplit is not called if is.na(split).
sep	a sep argument to use with paste to produce a matching key for the columns of x and y for which perfect matches are required. If(missing(sep) && not(missing(grep.))) sep <- ' ' except where grep. = NA.

Details

1. Check by.x, by.y, grep. and split. If((missing(by.x) | missing(by.y)) && missing(by)) by <- names(x)

2. fullMatch <- (is.na(grep.) & is.na(split)). Create keyfx and keyfy by by pasting columns of x[, by.x[fullMatch]] and y[, by.y[fullMatch]]. Also create x. and y. = strsplit of x[, by.x[!fullMatch]].

3. Iterate over rows of x looking for the best match. This includes an inner loop over columns of x[, by.x[!fullMatch]], stopping on the first unique match. Return (-1) if no unique match is found.

Value

an integer vector of length nrow(x) containing the index of the best matching row of y or NA if no adequate match was found.

Author(s)

Spencer Graves

See Also

strsplit, is.na grep, agrep match, row.match, join, match_df classify

Examples

```
newdata <- data.frame(state=c("AL", "MI","NY"),
                      surname=c("Rogers", "Rogers", "Smith"),
                      givenName=c("Mike R.", "Mike K.", "Al"),
                      stringsAsFactors=FALSE)
reference <- data.frame(state=c("NY", "NY", "MI", "AL", "NY", "MI"),
                      surname=c("Smith", "Rogers", "Rogers (MI)",
                                "Rogers (AL)", "Smith", 'Jones'),
                      givenName=c("John", "Mike", "Mike", "Mike",
                                "T. Albert", 'Al Thomas'),
                      stringsAsFactors=FALSE)
newInRef <- match.data.frame(newdata, reference,
       grep.=c(NA, 'agrep', 'agrep'))


all.equal(newInRef, c(4, 3, 5))
```

---

mergeUShouse.senate *Expand a dataset on some members of the US Congress to the entire membership*

Description

Merge a data.frame regarding some members of the US Congress with a data.frame with general information on all members.

Usage

```
mergeUShouse.senate(x, UScongress=UShouse.senate(),
        newrows="amount0",
        default=list(member=FALSE, amount=0, vote="notEligible",
                     incumbent=TRUE) )
```

Arguments

x	a data.frame to be merged with UScongress
UScongress	a data.frame to be merged with x.
newrows	name of a logical column to add that is TRUE for rows added to x and FALSE otherwise.
default	default values for columns of x identified by regexpr(names(default)[i], tolower(names(x))).

Details

1. keyx <- with(x, paste(houseSenate, state, District, sep=":"))
2. keyy <- with(UScongress(houseSenate, state, District, sep=":"))
3. notx <- !is.elememt(keyy, keyx)
4. Y <- UScongress[notx,]
5. add default columns to Y
6. if(!newrows is not in names(x))x <- cbind(x, newrows=FALSE)
7. Y[, newrows] <- TRUE
8. xY <- rbind(x, Y[c(names(x)])
9. replace 'Democrat' with 'Democratic' in xY[['Party']]
10. Look for NAs in "incumbent" who are nevertheless in UScongress; fix. Thus, if x[['incumbent']] is TRUE or FALSE, this value is not checked in UScongress; it's checked only if NA. The check consists of comparing names for a given Office:state:district between strsplit(x[['surname']], ' ')[[1]][1] and strsplit(UScongress[['surname']], ' ')[[1]][1] and similarly for givenName. This allows 'Rogers' in x[['surname']] to match 'Rogers (AL)' in UScongress[['surname']], etc. The algorithm is not perfect, but errors should be rare – and could be fixed manually.

Value

a data.frame combining x and UScongress as desired

Author(s)

Spencer Graves

See Also

merge UShouse.senate

Examples

```
tst <- data.frame(Office=factor(rep(c('House', 'Senate'), c(4, 2))),
         State=factor(c('Missouri', 'Minnesota', 'Tennessee',
                        'New York', rep('South Carolina', 2))),
         state=factor(c('MO', 'MN', 'TN', 'NY', 'SC', 'SC')),
         district=as.character(c(4, 1, 8, 18, 2, 3)),
         surname=c('Hartzler', 'Walz', 'Fincher', 'Maloney',
                   'Graham', 'DeMint'),
```

```
          givenName=c('Vicky', 'Timothy J.', 'Stephen Lee',
                    'Sean Patrick', 'Lindsey', 'Jim'),
          Party=c('Republican', 'Democrat', 'Republican', 'Democrat',
                  'Republican', 'Democrat'),
          CommitteeMember=rep(c(TRUE, FALSE), c(4, 2)),
          amount=c(5000, 2000, 29500, 1000, 1000, 11500),
          xvote=c('Y', 'N', 'Y', 'Y', 'notEligible', 'notEligible'),
          incumbent=NA, stringsAsFactors=FALSE )
tst2 <- mergeUShouse.senate(tst)

# A couple of simple tests;  don't test too much,
# because the results of UShouse.senate change,
# and we don't want this test to fail
# due to changes that don't affect Ecdat code

tst3 <- tst2[!tst2$amount0, c(1, 4:6, 8:10)]
row.names(tst) <- row.names(tst3)

## Not run:
all.equal(tst[c(1, 4:6, 8:10)], tst3)

## End(Not run)
# tst3[2] = state = factor with 56 levels,
# and tst[2] only has 5;  compare without this
```

`mergeVote` *Merge Roll Call Vote*

Description

Merge roll call vote record with a `data.frame` containing other information. The `vote` records are typically incomplete, so match first on `houseSenate` and `surname`. If this match is incomplete, try using `givenName`. If that fails, try `state` and `district`, which may not always be present in `vote`.

Usage

```
mergeVote(x, vote, Office="House", vote.x, check.x=TRUE)
```

Arguments

`x`	a `data.frame` whose columns include `Office`, `surname`, and `givenName`.
`vote`	a `data.frame` with column names which when forced `tolower` would match `surname`, `givenname`, and `vote`. However, the `givenname` may not be complete, so use it only if the `surname` is not sufficient.
`Office`	Either "House" or "Senate"; ignored if `vote` includes a column `Office`.

vote.x — name of a column of x containing a vote to be updated with the vote column of the vote data.frame. If missing and x has a column with a name matching "vote", then vote.x is that column. If missing but x has no such column, then append a column to x with the name of the vote column of the vote data.frame.

check.x — logical: If TRUE, check for rows of x[, vote.x] that are NOT in vote and throw an error if found.

Details

1. Parse vote.x to get the name of the column of x into which to write the vote column of the vote data.frame.

2. If the vote data.frame contains a column Office, ignore the Office argument. Otherwise, add the argument houseSenate as a column of vote.

3. Create keyx <- with(x, paste(Office, surname, sep=":")), keyx2 <- paste(keyx, givenName, sep=":"), keyx. <- paste(houseSenate, state, district, sep=":"), and similarly keyv, leyv2, and keyv. from vote.

4. Look for keyv in keyx. When a unique match is found, transfer the vote the vote column of x. When no match is found, try for keyv2 in keyx2 or keyv. in keyx. If those fail, print an error message with the information from vote on all failures and ask the user to add state and district informuation.

5. if(check.x), check for rows in x[, vote.x] that are NOT "notEligible" but are also not in vote: Throw an error if any are found.

Value

a data.frame with the same columns as x with its vote column modified per the vote argument.

Author(s)

Spencer Graves

See Also

mergeUShouse.senate

Examples

```
##
## 1.  Test good cases
##
votetst <- data.frame(
  surName=c('Smith', 'Jones', 'Graves', 'Jsn', 'Jsn', 'Gay'),
  givenName=c("Sam", "", "", "John", "John", ''),
  votex=factor(c('Y', 'N', 'abstain', 'Y', 'Y', 'Y')),
  State=factor(rep(c("CA", "", "SC", "NY"), c(1, 2, 1, 2))),
  district=rep(c("13", "1", "2", "1"), c(1, 2, 2, 1)),
  stringsAsFactors=FALSE )
```

```
x1 <- data.frame(
  Office=factor(rep(c("House", "Senate"), e=8)),
  state=rep(c("NY", "SC", "SD", "CA", "AK", "AR", "NY", "NJ"), 2),
  District=rep(c("2", "2", "At Large", "13", "1", "9", "1", "3"), 2),
  surname=rep(c('Jsn', 'Jsn', 'Smith', 'Smith', 'Jones',
       'Graves', 'Rx', 'Agnew'), 2),
  givenName=rep(c("John D.", "John J.",
    "Samual", "Samual", "Mary", "Mary", "Susan", 'Spiro'), 2),
  don=1:16, stringsAsFactors=FALSE)

x1. <- mergeVote(x1, votetst)

x2 <- cbind(x1, votex=factor( rep(
   c('Y', 'notEligible', 'Y', 'N', 'abstain', 'Y', 'notEligible'),
   c(2,1,1,1,1,1,9) ) ) )


all.equal(x1., x2)


##
## 2.  Test a case with a vote error in x
##

x1a <- cbind(x1, voterr=rep(
     c('notEligible', 'Y', 'notEligible'), c(7, 1, 8)))

x1a. <- try(mergeVote(x1a, votetst))


class(x1a.)=='try-error'
```

---

parseCommas *Convert character string with Dollar signs and commas to numerics*

Description

as.numeric of character strings after suppressing commas and dollar signs. This is a generalization of parseDollars.

Usage

```
parseCommas(x, pattern='\$|,', replacement='',
        acceptableErrorRate=0, ...)
## Default S3 method:
parseCommas(x, pattern='\$|,', replacement='',
        acceptableErrorRate=0, ...)
## S3 method for class 'data.frame'
```

```
parseCommas(x, pattern='\$|,', replacement='',
        acceptableErrorRate=0, ...)
```

Arguments

x	vector of character strings to be converted to numerics
pattern	regular expression to be replaced by replacement
replacement	Character string to substitute for each occurrence of pattern
acceptableErrorRate	number indicating the proportion of new NAs to that can be introduced and still assume it's numeric
...	optional arguments to pass to gsub

Details

as.numeric(gsub(x, ...))

The data.frame method outputs another data.frame with character or factor columns connverted to numerics using parseDollars whenever that can be done without creating NAs.

Value

Numeric vector converted from the character strings in x or a data.frame with columns that are obviously numbers in character format converted to numerics.

Author(s)

Spencer Graves

See Also

gsub as.numeric parseDollars

Examples

```
##
## 1.  a character vector
##
X2 <- c('-$2,500', '$5,000.50')
x2 <- parseDollars(X2)


all.equal(x2, c(-2500, 5000.5))


##
## A data.frame
##
chDF <- data.frame(let=letters[1:2], Dol=X2, dol=x2)
numDF <- parseCommas(chDF)
```

```
chkDF <- chDF
chkDF$Dol <- x2

all.equal(numDF, chkDF)
```

---

parseDollars *Convert character string with Dollar signs and commas to numerics*

Description

as.numeric of character strings after suppressing commas and dollar signs. This is a special case of parseCommas.

Usage

```
parseDollars(x, pattern='\$|,', replacement='', ...)
```

Arguments

x	vector of character strings to be converted to numerics
pattern	regular expression to be replaced by replacement
replacement	Character string to substitute for each occurrence of pattern
...	optional arguments to pass to gsub

Details

as.numeric(gsub(x, ...)). See also parseCommas.

Value

Numeric vector converted from x.

Author(s)

Spencer Graves

See Also

gsub as.numeric parseCommas

Examples

```
##
## 1.  a character vector
##
X2 <- c('-$2,500', '$5,000.50')
x2 <- parseDollars(X2)


all.equal(x2, c(-2500, 5000.5))


##
## A data.frame
##
chDF <- data.frame(let=letters[1:2], Dol=X2, dol=x2)
numDF <- parseCommas(chDF)

chkDF <- chDF
chkDF$Dol <- x2

all.equal(numDF, chkDF)
```

`parseName` *Parse surname and given name*

Description

Identify the presumed surname in a character string assumed to represent a name and return the result in a character matrix with "surname" followed by "givenName".

Usage

```
parseName(x, surnameFirst=(median(regexpr(',', x))>0),
          suffix=c('Jr.', 'I', 'II', 'III', 'IV', 'Sr.'),
          fixNonStandard=subNonStandardNames, ...)
```

Arguments

`x`	a character vector
`surnameFirst`	logical: If TRUE, the surname comes first followed by a comma (","), then the given name. If FALSE, parse the surname from a standard Western "John Smith, Jr." format. If `missing(surnameFirst)`, use TRUE if half of the elements of `x` contain a comma.
`suffix`	character vector of strings that are NOT a surname but might appear at the end without a comma that would otherwise identify it as a suffix.

`fixNonStandard` function to look for and repair nonstandard names such as names containing characters with accent marks that are sometimes mangled by different software. Use `identity` if this is not desired.

`...` optional arguments passed to `fixNonStandard`

Details

If `surnameFirst` is `FALSE`:

1. If the last character is ")" and the matching "(" is 3 characters earlier, drop all that stuff. Thus, "John Smith (AL)" becomes "John Smith".

2. Look for commas to identify a suffix like Jr. or III; remove and call the rest x2.

3. split <- strsplit(x2, " ")

4. Take the last as the surname.

5. If the "surname" found per 3 is in `suffix`, save to append it to the `givenName` and recurse to get the actual surname.

NOTE: This gives the wrong answer with double surnames written without a hyphen in the Spanish tradition, in which, e.g., "Anistasio Somoza Debayle", "Somoza Debayle" give the (first) surnames of Anistasio's father and mother, respectively: The current algorithm would return "Debayle" as the surname, which is incorrect.

6. Recompose the rest with any suffix as the givenName.

Value

a character matrix with two columns: surname and givenName

Author(s)

Spencer Graves

See Also

`strsplit` `identity`

Examples

```
##
## 1.  Parse standard first-last name format
##
tst <- c('Joe Smith (AL)', 'Teresa Angelica Sanchez de Gomez',
         'John Brown, Jr.', 'John Brown Jr.',
         'John W. Brown III', 'John Q. Brown,I',
         'Linda Rosa Smith-Johnson', 'Anastasio Somoza Debayle',
         'Ra_l Vel_zquez')
library(Ecdat)
parsed <- parseName(tst)

tst2 <- matrix(c('Smith', 'Joe', 'Gomez', 'Teresa Angelica Sanchez de',
  'Brown', 'John, Jr.', 'Brown', 'John, Jr.',
```

```
  'Brown', 'John W., III', 'Brown', 'John Q., I',
  'Smith-Johnson', 'Linda Rosa', 'Debayle', 'Anastasio Somoza',
  'Velazquez', 'Raul'),
  ncol=2, byrow=TRUE)
# NOTE:  This second to last example is in the Spanish tradition
# and is handled incorrectly by the current algorithm.
# The correct answer should be "Somoza Debayle", "Anastasio".
# However, fixing that would complicate the algorithm excessively for now.
colnames(tst2) <- c("surname", 'givenName')


all.equal(parsed, tst2)


##
## 2.  Parse "surname, given name" format
##
tst3 <- c('Smith (AL),Joe', 'Sanchez de Gomez, Teresa Angelica',
     'Brown, John, Jr.', 'Brown, John W., III', 'Brown, John Q., I',
     'Smith-Johnson, Linda Rosa', 'Somoza Debayle, Anastasio',
     'Vel_zquez, Ra_l')
tst4 <- parseName(tst3)

tst5 <- matrix(c('Smith', 'Joe', 'Sanchez de Gomez', 'Teresa Angelica',
  'Brown', 'John, Jr.', 'Brown', 'John W., III', 'Brown', 'John Q., I',
  'Smith-Johnson', 'Linda Rosa', 'Somoza Debayle', 'Anastasio',
  'Velazquez', 'Raul'),
  ncol=2, byrow=TRUE)
colnames(tst5) <- c("surname", 'givenName')


all.equal(tst4, tst5)
```

`Ping` *ping a Uniform resource locator (URL)*

Description

***NOTE: THIS IS A PRELIMINARY VERSION OF THIS FUNCTION; ***NOTE: IT MAY BE CHANGED OR REMOVED IN A FUTURE RELEASE.

ping a Uniform resource locator (URL) or Internet Protocol (IP) address.

NOTE: Some Internet Service Providers (ISPs) play games with "ping". That makes the results of `Ping` unreliable.

Usage

```
Ping(url, pingArgs='', warn=NA,
     show.output.on.console=FALSE)
```

Arguments

url	a character string of a URL or IP address to ping. If url is a vector of length greater than 1, only the first component is used.
pingArgs	arguments to pass to the ping command of typical operating systems via pingResult <- system(paste('ping', pingArgs, url), intern=TRUE, ...)
warn	value for options('warn') during the call to system. NA to not change options('warn') during this call.
show.output.on.console	argument for system.

Details

1. urlSplit0 <- strsplit(url, '://')[[1]]
2. urlS0 <- urlSplit0[min(2, length(urlSplit0))]
3. host <- strsplit(urlS0, '/')[[1]][1]
4. pingCmd <- paste('ping', pingArgs, host)
5. system(pingCmd, intern=TRUE, ...)

Value

list with the following components:

rawResults	character vector of the raw results from the ping command
rawNumbers	numeric vector of the times measured
counts	numeric vector of numbers of packets sent, received, and lost
p.lost	proportion lost = lost / sent
stats	numeric vector of min, avg (mean), max, and mdev (standard deviation) of the measured round trip times

Author(s)

Spencer Graves

See Also

system, options

Examples

```
##
## Some ISPs play games with ping.
## Therefore, the results are not reliable.
##
## Not run:
##
## good
```

```
##
(google <- Ping('http://google.com/ping works on host not pages'))

\dontshow{stopifnot(}
with(google, (counts[1]>0) && (counts[3]<1))
\dontshow{)}

##
## ping oops <<-- at one time, this failed.
##      However, with some ISPs, it works, so don't test it.
##

##
(couldnotfindhost <- Ping('oops'))

\dontshow{stopifnot(}
with(couldnotfindhost,
     length(grep('could not find host', rawResults))>0)
\dontshow{)}

##
## impossible, but not so obvious
##
(requesttimedout <- Ping('requesttimedout.com'))

\dontshow{stopifnot(}
with(requesttimedout, (counts[1]>0) && (counts[2]<1) &&
         (counts[3]>0))
\dontshow{)}

## End(Not run)
```

---

qqnorm2 *Normal Probability Plot with Multiple Symbols*

Description

Create a normal probability plot with different symbols for the values of another variable. qqnorm2 produces an object of class qqnorm2, whose plot method produces the plot.

Usage

```
qqnorm2(y, z, plot.it=TRUE, datax=TRUE, pch=NULL, ...)
## S3 method for class 'qqnorm2'
plot(x, y, ...)
## S3 method for class 'qqnorm2'
lines(x, ...)
## S3 method for class 'qqnorm2'
points(x, ...)
```

Arguments

y	For qnorm2, y is a numeric vector for which a normal probability plot is desired. For plot.qqnorm2, y is ignored; it is included, because the generic plot function requires it.
z	A variable to indicate different plotting symbols.
plot.it	logical: Should the result be plotted?
datax	The datax argument of qqnorm: If TRUE, the data are displayed on the horizontal rather than the vertical axis. (The default value for datax is the opposite of that for qqnorm.)
x	an object of class qqnorm2.
pch	a named vector of the plotting symbols to be used with names corresponding to the levels of z. By default, if z takes levels FALSE and TRUE (or 0 and 1), pch=c(4, 1) to plot a "x" for FALSE and "o" for TRUE. If z assumes integer values between 0 and 255, by default, the symbols are chosen as described with points. Otherwise, by default, z is coerced to character, and the result is plotted. If pch is provided, it must eitehr have names corresponding to levels of z, or z must be integers between 1 and length(pch).
...	Optional arguments. For plot.qqnorm2, they are passed to plot. For qqnorm2, they are passed to qqnorm and to plot.qqnorm2.

Details

For qqnorm2:

1. q2 <- qqnorm(y, datax=datax, ...)
2. q2[["z"]] <- z
3. q2[["pch"]] gets whatever pch decodes to.
4. Silently return(list(x, y, z, pch)), where "x" and "y" are as returned by qqnorm in step 1 above.

For plot.qqnorm2, plot(x, y, pch=pch[z], ...). For lines.qqnorm2, lines(x, y, pch=pch[z], ...). For points.qqnorm2, points(x, y, pch=pch[z], ...).

Value

qqnorm2 returns a list with components, x, y, z, and pch.

Author(s)

Spencer Graves

See Also

qqnorm, qqnorm2s, plot points lines

Examples

```
##
## a simple test data.frame to illustrate the plot
## but too small to illustrate qqnorm concepts
##
tstDF <- data.frame(y=1:3, z1=1:3, z2=c(TRUE, TRUE, FALSE),
                    z3=c('tell', 'me', 'why'), z4=c(1, 2.4, 3.69) )
# plotting symbols circle, triangle, and "+"
qn1 <- with(tstDF, qqnorm2(y, z1))

# plotting symbols "x" and "o"
qn2 <- with(tstDF, qqnorm2(y, z2))

# plotting with "-" and "+"
qn. <- with(tstDF, qqnorm2(y, z2, pch=c('FALSE'='-', 'TRUE'='+')))

# plotting with "tell", "me", "why"
qn3 <- with(tstDF, qqnorm2(y, z3))

# plotting with the numeric values
qn4 <- with(tstDF, qqnorm2(y, z4))

##
## test plot, lines, points
##
plot(qn4, type='n') # establish the scales
lines(qn4)          # add a line
points(qn4)         # add points

##
## Check the objects created above
##
# check qn1
qn1. <- qqnorm(1:3, datax=TRUE, plot.it=FALSE)
qn1.$xlab <- 'y'
qn1.$ylab <- 'Normal scores'
qn1.$z <- tstDF$z1
qn1.$pch <- 1:3
names(qn1.$pch) <- 1:3
qn11 <- qn1.[c(3:4, 1:2, 5:6)]
class(qn11) <- 'qqnorm2'

all.equal(qn1, qn11)


# check qn2
qn2. <- qqnorm(1:3, datax=TRUE, plot.it=FALSE)
qn2.$xlab <- 'y'
qn2.$ylab <- 'Normal scores'
qn2.$z <- tstDF$z2
qn2.$pch <- c('FALSE'=4, 'TRUE'=1)
qn22 <- qn2.[c(3:4, 1:2, 5:6)]
```

```
class(qn22) <- 'qqnorm2'

all.equal(qn2, qn22)


# check qn.
qn.. <- qqnorm(1:3, datax=TRUE, plot.it=FALSE)
qn..$xlab <- 'y'
qn..$ylab <- 'Normal scores'
qn..$z <- tstDF$z2
qn..$pch <- c('FALSE'='-', 'TRUE'='+')
qn.2 <- qn..[c(3:4, 1:2, 5:6)]
class(qn.2) <- 'qqnorm2'

all.equal(qn., qn.2)


# check qn3
qn3. <- qqnorm(1:3, datax=TRUE, plot.it=FALSE)
qn3.$xlab <- 'y'
qn3.$ylab <- 'Normal scores'
qn3.$z <- as.character(tstDF$z3)
qn3.$pch <- as.character(tstDF$z3)
names(qn3.$pch) <- qn3.$pch
qn33 <- qn3.[c(3:4, 1:2, 5:6)]
class(qn33) <- 'qqnorm2'

all.equal(qn3, qn33)


# check qn4
qn4. <- qqnorm(1:3, datax=TRUE, plot.it=FALSE)
qn4.$xlab <- 'y'
qn4.$ylab <- 'Normal scores'
qn4.$z <- tstDF$z4
qn44 <- qn4.[c(3:4, 1:2, 5)]
qn44$pch <- NULL
class(qn44) <- 'qqnorm2'

all.equal(qn4, qn44)
```

---

qqnorm2s *Normal Probability Plot with Multiple Lines and Multiple Symbols*

Description

Create a normal probability plot with multiple lines for different variables and different symbols for the values of another variable. qqnorm2s produces an object of class qqnorm2s, whose plot method produces the plot.

Usage

```
qqnorm2s(y, z, data., plot.it=TRUE, datax=TRUE, outnames=y,
         pch=NULL, col=c(1:4, 6), legend.=NULL, ...)
## S3 method for class 'qqnorm2s'
plot(x, y, ...)
```

Arguments

y	For qnorm2s, y is a character vector of names of columns of data. for which normal probability plots are desired. data. is either a data.frame or a list of data.frames of the same length as y, with y[i] being the name of a column of the data.frame data.[[i]]. z is a similar character vector of names of columns of data., which identify symbols for plotting different points in a normal probability plot. The lengths of y, and z must match the number of data.frames in data.; if not, the lengths of the shorter are replicated to the length of the longest before computations begin. For plot.qqnorm2, y is ignored; it is included, because the generic plot function requires it.
z	A character vector giving the names of columns of data. to indicate different plotting symbols. z should be the same length as y and must equal the number of data.frames in the list data. of data.frames. If not, the shorter are replicated to the length of the longer.
data.	a data.frame or a list of data.frames with columns named in y and z.
plot.it	logical: Should the result be plotted?
datax	The datax argument of qqnorm: If TRUE, the data are displayed on the horizontal rather than the vertical axis. (The default value for datax is the opposite of that for qqnorm.)
outnames	Names for the components of the qqnorm2s object returned by the qqnorm2s function.
pch	a named vector of the plotting symbols to be used with names corresponding to the levels of z. By default, if z takes levels FALSE and TRUE (or 0 and 1), pch=c(4, 1) to plot a "x" for FALSE and "o" for TRUE. If z assumes integer values between 0 and 255, by default, the symbols are chosen as described with points. Otherwise, by default, z is coerced to character, and the result is plotted. If pch is provided, it must eitehr have names corresponding to levels of z, or z must be integers between 1 and length(pch).
col	A vector indicating the colors corresponding to each element of y. Defaults to rep(c(1:4, 6), length=length(y)), with 1:4 and 6 being black, red, green, blue, and pink.
x	an object of class qqnorm2.
legend.	A list with components pch and col providing information for legend to identify the plotting symbols (pch) and colors (col).

By default, pch = list(x='right', legend=names(qq2s[[1]][['pch']]), pch=qq2s[[1]][['pch']]), where qq2s is described below in details.

Similarly, by default, lines = list(x='bottomright', legend=y, lty=1, pch=NA, col=qq2s[[1]][['col']]).

... Optional arguments.

For `plot.qqnorm2s`, they are passed to `plot`.

For `qqnorm2s`, they are passed to `qqnorm2` and to `plot.qqnorm2s`.

Details

For `qqnorm2s`:

1. Create qq2s = a list of objects of class `qqnorm2`
2. Add `legend.` to `qq2s`.
3. class(qq2s) <- 'qqnorm2s'
4. if(plot.it)plot(qq2s, ...)
5. Silently return(qq2s).

For `plot.qqnorm2s`, create a plot with one line for each variable named in `y`.

Value

`qqnorm2s` returns a named list with components of class `qqnorm2` with names = `y` with each component having an additional component `col` plus one called "legend.".

Author(s)

Spencer Graves

See Also

`qqnorm2` `plot`

Examples

```
##
## One data.frame
##
tstDF2 <- data.frame(y=1:3, y2=3:5, z2=c(TRUE, TRUE, FALSE),
                     z3=c('tell', 'me', 'why'), z4=c(1, 2.4, 3.69) )
# produce the object and plot it
Qn2 <- qqnorm2s(c('y', 'y2'), 'z2', tstDF2)

# plot the object previously created
plot(Qn2)

# Check the object
qy <- with(tstDF2, qqnorm2(y, z2, type='b'))
qy$col <- 1
qy2 <- with(tstDF2, qqnorm2(y2, z2, type='b'))
```

```
qy2$col <- 2
legend. <- list(pch=list(x='right', legend=c('FALSE', 'TRUE'),
                pch=c('FALSE'=4, 'TRUE'= 1)),
                col=list(x='bottomright', legend=c('y', 'y2'),
                lty=1, col=1:2))
Qn2. <- list(y=qy, y2=qy2, legend.=legend.)
class(Qn2.) <- 'qqnorm2s'


all.equal(Qn2, Qn2.)

##
## Two data.frames
##
tstDF2b <- tstDF2
tstDF2b$y <- c(0.1, 0.1, 9)
Qn2b <- qqnorm2s('y', 'z2', list(tstDF2, tstDF2b),
           outnames=c('ok', 'oops'), log='x' )
```

`read.testURLs` *Read a file produced by testURLs*

Description

***NOTE: THIS IS A PRELIMINARY VERSION OF THIS FUNCTION; ***NOTE: IT MAY BE CHANGED OR REMOVED IN A FUTURE RELEASE.

`read.table(file.)` and return the result as an object of class c('testURLs', 'data.frame').

Usage

```
read.testURLs(file.='testURLresults.csv', ...)
```

Arguments

`file.`	Name of a CSV file to read
`...`	optional arguments for `read.csv`.

Details

dat <- read.csv(file., ...)

class(dat) <- c('testURLsFile', 'data.frame')

Value

a `data.frame` from the file written by `testURLs`, of the same format as the `testResults` attribute of the `testURLs` object returned by `testURLs`.

Author(s)

Spencer Graves

See Also

read.csv

Examples

```
# Test only 2 web sites, not the default 4,
# and test only twice, not the default 10 times:
tst <- testURLs(c(
 PVI="http://en.wikipedia.org/wiki/Cook_Partisan_Voting_Index",
 house="http://house.gov/representatives"),
    n=2, maxFail=2)

# The above should have created a file 'testURLresults.csv'
# in the working directory.  Read it.

dat <- read.testURLs()
```

---

read.transpose *Read a data table in transpose form*

Description

Read a text (e.g., csv) file, find rows with more than 3 sep characters. Parse the initial contiguous block of those into a matrix. Add attributes headers, footers, and a summary.

The initial application for this function is to read Table 6.16. Income and employment by industry in the National Income and Product Account tables published by the Bureau of Economic Analysis of the United States Department of Commerce.

Usage

```
read.transpose(file, header=TRUE, sep=',',
               na.strings='---', ...)
```

Arguments

file	the name of a file from which the data are to be read.
header	Logical: Is the second column of the identified data matrix to be interpreted as variable names?
sep	The field space separator charactor.
na.strings	character string(s) that translate into NA
...	optional arguments for strsplit

Details

1. txt <- readLines(file)
2. Split into fields.
3. Identifiy headers, Data, footers.
4. Recombine the second component of each Data row if necessary so all have the same number of fields.
5. Extract variable names
6. Numbers?
7. return the transpose

Value

A matrix of the transpose of the rows with the max number of fields with attributes 'headers', 'footers', 'other', and 'summary'. If this matrix can be coerced to numeric with no NAs, it will be. Otherwise, it will be left as character.

Author(s)

Spencer Graves

References

Table 6.16. Income and employment by industry in the National Income and Product Account tables published by the Bureau of Economic Analysis of the United States Department of Commerce. To get this table from www.bea.gov, under "U.S. Economic Accounts", first select "Corporate Profits" under "National". Then next to "Interactive Tables", select, "National Income and Product Accounts Tables". From there, select "Begin using the data...". Under "Section 6 - income and employment by industry", select each of the tables starting "Table 6.16". As of February 2013, there were 4 such tables available: Table 6.16A, 6.16B, 6.16C and 6.16D. Each of the last three are available in annual and quarterly summaries. The `USFinanceIndustry` data combined the first 4 rows of the 4 annual summary tables.

See Also

`read.table` `readLines` `strsplit`

Examples

```
#  Find demoFiles/*.csv
demoDir <- system.file('demoFiles', package='Ecdat')
(demoCsv <- dir(demoDir, pattern='csv$', full.names=TRUE))

# Use the fourth example
# to ensure the code will handle commas in a name
# and NAs
nipa6.16D <- read.transpose(demoCsv[4])
str(nipa6.16D)
```

readCookPVI *Read Cook Partisan Voting Index*

Description

Read tables of the Cook Partisan Voting Index and returns a list with components 'House' and 'Senate'. readCookPVI. returns the tables with the names of the current incumbents per readUShouse and readUSsenate; readCookPVI tables do not include the names of the incumbents.

Usage

```
readCookPVI(url.=
"http://en.wikipedia.org/wiki/Cook_Partisan_Voting_Index")
readCookPVI.(url.=
"http://en.wikipedia.org/wiki/Cook_Partisan_Voting_Index",
      UShouse=readUShouse(), USsenate=readUSsenate(), ...)
```

Arguments

url.	Universal resource locator to be read and processed to obtain the desired lists.
UShouse, USsenate	data.frames as returned by readUShouse and readUSsenate, respectively.
...	optional arguments passed to readUShouse and readUSsenate.

Details

The primary source for these data is the Cook Political Report web site. However, the current URL we have for these data on that web site includes "2012" in the title. If and when the numbers are updated, we would expect that file name to change.

To avoid that problem the code is currently set to read from the Wikipedia article on "Cook Partisan Voting Index".

The algorithm reads the web site into a list, finds the desired tables on the list, then parses and formats them as desired. Then it merges the results with UShouse and USsenate.

Value

A list with components "House" and "Senate". Each contains a data.frame. The "House" data.frame returned by readCookPVI includes the following columns:

State	name of the state
District	District, e.g, 1st, 2nd, At-Large
PVInum	PVI as a number ranging from roughly 50 to 150. 100 means that the vote split in that district was within 0.5 percent of the national average. 101 means that it tilts 1 percent (after rounding) to Republican. 98 means that it tilts 1 percent to Democratic; 99 is not used.

`PVIchar` PVI rating in character format. For example, 'D+1' means that the vote tilted 1 percent toward Democratic more than the national average. 'R+1' means that it tilted 1 percent toward Republican.

`PartyOfRepresentative`
Party of the incumbent, either 'Republican' or 'Democratic'

The 'Senate' data.frame includes the following columns:

`State` name of the state

`PVInum` PVI numeric, as for 'House'

`PVIchar` PVI rating in character format, as for 'House'

`PartyOfGovernor`
Party of the Governor of the state

`PartyInSenate` party of the incumbent senators, either 'Republican', Democratic', or 'Both'.

`houseBalanceNum`
House balance as a number with 0 = 100 percent Democratic, 99.9 = 100 percent Republican, and 500 for the same number of Republicans as Democrats.

`houseBalanceChar`
Count by pary in the house delegation for that state, e.g., '6R, 1D' for 6 Republicans and 1 Democrat.

`readCookPVI.` adds to the above the information returned by `readUShouse` and `readUSsenate`.

Author(s)

Spencer Graves

Source

Wikipedia, "Cook Partisan Voting Index" The Cook Political Report

See Also

`readUShouse`, `readUSsenate`

Examples

```
## Not run:
CookPVI <- readCookPVI()

## End(Not run)

CookPVI. <- readCookPVI.()
```

readFinancialCrisisFiles *banking crisis data and function to read financial crisis files*

Description

Read financial crisis data in files described by an object of class financialCrisisFiles. This is designed to read Excel files describing financial crises since 1800 downloaded from http://www.reinhartandrogoff.com/data/browse-by-topic/topics/7/.

bankingCrises is a data.frame created by readFinancialCrisisFiles() using 3 files downloaded from http://www.reinhartandrogoff.com and 1 obtained from Prof. Reinhart in January 2013.

Usage

```
readFinancialCrisisFiles(files, crisisType=7, ...)
```

Arguments

files	an object of class financialCrisisFiles.
crisisType	an integer (vector) between 1 and 8 indicating the type of data to be retrieved: 1=independence year (not a crisis but an indicator), 2=currency, 3=inflation, 4=stock market, 5=domestic soveregn debt crisis, 6=external sovereign debt crisis, 7=banking, 8=tally. ("Type" 1 = year.) These are all 0 or 1 indicating the presence of the event in the given year. Type 8 = sum of types 2 through 7.
...	arguments to pass with file and sheet name to read.xls when reading a sheet of an MS Excel file. This is assumed to be the same for all sheets of all files. If this is not the case, the resulting financialCrisisFiles object will have to be edited manually before using it to read the data.

Details

Reinhart and Rogoff (http://www.reinhartandrogoff.com) provide numerous data sets analyzed in their book, "This Time Is Different: Eight Centuries of Financial Folly". Of interest here are data on financial crises of various types for 70 countries spanning the years 1800 - 2010, downloadable from http://www.reinhartandrogoff.com/data/browse-by-topic/topics/7/.

The function financialCrisisFiles produces a list of class financialCrisisFiles describing different Excel files in very similar formats with one sheet per Country and a few extra descriptor sheets. The data object FinancialCrisisFiles is the default output of that function.

readFinancialCrisisFiles reads the sheets for the individual countries.

Value

If length(crisisType) == 1, a data.frame is returned with the first column being year, and with one other column containing the data for that crisisType for each country.

If length(crisisType) > 1, a list is returned containing a data.frame for each country.

Author(s)

Spencer Graves

Source

http://www.reinhartandrogoff.com

References

Carmen M. Reinhart and Kenneth S. Rogoff (2009) This Time Is Different: Eight Centuries of Financial Folly, Princeton U. Pr.

See Also

read.xls financialCrisisFiles

Examples

```
##
## Recreate / update the data object BankingCrises
##
library(Ecdat)

## Not run:
bankingCrises <- readFinancialCrisisFiles(FinancialCrisisFiles)

## End(Not run)

##
## Toy example using local data to check the code
## and illustrate returning all the data not just one crisisTpe
##
Ecdat.demoFiles <- system.file('demoFiles', package='Ecdat')
Ecdat.xls <- dir(Ecdat.demoFiles, pattern='xls$',
                 full.names=TRUE)
tst <- financialCrisisFiles(Ecdat.xls)

bankingCrises.tst <- readFinancialCrisisFiles(tst)
allCrises.tst <- readFinancialCrisisFiles(tst, 1:8)

# Manually construct tst from allCrises.tst
tst2 <- data.frame(year=1800:1999)
tst2$Algeria <- as.numeric(allCrises.tst$Algeria[-(1:12), 8])
tst2$CentralAfricanRep <- as.numeric(
                    allCrises.tst$CentralAfricanRep[-(1:12), 8])
tst2$Taiwan <- as.numeric(allCrises.tst$Taiwan[-(1:11), 8])
tst2$UK <- as.numeric(allCrises.tst$UK[-(1:11), 8])


all.equal(bankingCrises.tst, tst2)
```

```
# check
data(bankingCrises)

all.equal(bankingCrises.tst,
   bankingCrises[1:200, c('year', 'Algeria', 'CentralAfricanRep',
                          'Taiwan', 'UK')])
```

---

readNIPA *Read a National Income and Product Accounts data table*

Description

Read multiple files with data in rows using read.transpose and combine the initial columns.

Usage

```
readNIPA(files, sep.footnote='/', ...)
```

Arguments

files	A character vector of names of files from which the data are to be read using read.transpose.
sep.footnote	a single character to identify footnote references in the variable names in some but not all of files.
...	optional arguments for read.transpose

Details

This is written first and foremost to facilitate updating USFinanceIndustry from Table 6.16: Income and employment by industry in the National Income and Product Account tables published by the Bureau of Economic Analysis of the United States Department of Commerce. As of February 2013, this table can be obtained from http://www.bea.gov: Under "U.S. Economic Accounts", first select "Corporate Profits" under "National". Then next to "Interactive Tables", select, "National Income and Product Accounts Tables". From there, select "Begin using the data...". Under "Section 6 - income and employment by industry", select each of the tables starting "Table 6.16". As of February 2013, there were 4 such tables available: Table 6.16A, 6.16B, 6.16C and 6.16D. Each of the last three are available in annual and quarterly summaries. The USFinanceIndustry data combined the first 4 rows of the 4 annual summary tables.

This is available in 4 separate files, which must be downloaded and combined using readNIPA. The first three of these are historical data and are rarely revised. For convenience and for testing, they are provided in the demoFiles subdirectory of this Ecdat package.

It has not been tested on other data but should work for annual data with a sufficiently similar structure.

The algorithm proceeds as follows:

1. Data <- lapply(files, read.transpose)

2. Is Data a list of numeric matrices? If no, print an error.

3. cbind common initial variables, averaging overlapping years, reporting percent difference

4. attributes: stats from files and overlap. Stats include the first and last year and the last revision date for each file, plus the number of years overlap with the previous file and the relative change in the common files kept between those two files.

Value

a matrix of the common variables

Author(s)

Spencer Graves

References

United States Department of Commerce Bureau of Economic Analysis National Income and Product Account tables

See Also

read.table readLines strsplit

Examples

```
#  Find demoFiles/*.csv
demoDir <- system.file('demoFiles', package='Ecdat')
(demoCsv <- dir(demoDir, pattern='csv$', full.names=TRUE))

nipa6.16 <- readNIPA(demoCsv)
str(nipa6.16)
```

---

readUShouse *Read the list of representatives in the United States House of Representatives*

Description

Read the list of representatives in the United States House of Representatives.

Usage

```
readUShouse(url.="http://house.gov/representatives/",
   nonvoting=c('American Samoa', 'District of Columbia',
               'Guam', 'Northern Mariana Islands', 'Puerto Rico',
               'Virgin Islands'),
   fixNonStandard=subNonStandardNames, ...)
```

Arguments

url.	Universal resource locator to be read and processed to obtain the desired list
nonvoting	Character vector of the names of US territories that send a nonvoting delegate to the US House.
fixNonStandard	function to look for and repair nonstandard names such as names containing characters with accent marks that are sometimes mangled by different software. Use identity if this is not desired.
...	optional arguments passed to fixNonStandard

Details

1. House.gov <- readHTMLTable(url). As of April 2013, this is a list of 80 tables. The first 56 are for the 50 states and 6 territories. The remaining 24 are for the first letter of the last name of the representatives.

2. Use rbind to collapse these into 2 tables. The first has the district as a number without identifying the state (because that was with the names of the first 56 tables in House.gov). The second has the state names but with the district numbers in a form not easily parsed.

3. Obtain the state names from the second table to match the names of the representatives in the first.

4. Add a nonvoting column for those "States" in nonvoting.

5. Look for and fix surname and givenName with nonstandard characters using fixNonStandard.

Value

readUShouse returns a data.frame with the following columns:

State	A factor identifying the state or territory the person represents
state	2-letter US Postal Service abbreviation for the state or territory
district	the character vector identifying the district each person represents. This is either an integer in character format or 0 for "At Large".
Name	A character vector giving the name of each representative (in surname, given name format)
party	a factor identifying the party affiliation of each representative ("D" or "R").
Room	character vector identifying the room number of the office
Phone	character vector giving the phone number
Committees	a character vector giving the committee assignments of each representative
surname	character vector giving the surname of each representative
givenName	given name of each representative (possibly with middle name or initial, a nickname, and a suffix like "Jr.")

Author(s)

Spencer Graves

See Also

getURL readHTMLTable readUSsenate UShouse.senate parseName readUSstateAbbreviations subNonStandardNames readCookPVI

Examples

```
UShouse <- readUShouse()
```

---

readUSsenate *Read the list of elected officials in the United States Senate*

Description

Read the list of elected officias in the United States Senate.

Usage

```
readUSsenate(url.=
"http://en.wikipedia.org/wiki/List_of_current_United_States_Senators",
    stateAbbreviations=Ecdat::USstateAbbreviations,
   fixNonStandard=subNonStandardNames, ...)
```

Arguments

url.	Universal resource locator to be read and processed to obtain the desired list NOTE: On April 26, 2013 the obvious naive use of readHTMLTable worked with the Wikipedia article on the US Senate but did not work with "senate.gov".
stateAbbreviations	a data.frame giving names and alternative codes for US states and territories. This must have a column named "Name" giving the names of all 50 states as they appear on the url and another whose name includes "USPS" giving the correspoding 2-letter codes.
fixNonStandard	function to look for and repair nonstandard names such as names containing characters with accent marks that are sometimes mangled by different software.
...	optional arguments passed to fixNonStandard

Details

1. Senate <- readHTMLTable(url)
2. Use camelParse to remove duplication in Name.
3. Look for and fix surname and givenName with nonstandard characters using fixNonStandard.

Value

readUSsenate returns a data.frame with the following columns:

State	A factor identifying the state the person represents
state	A factor giving the 2-letter USPS code for the state represented
Class	"1", "2", or "3" for election in the 6-year cycle including 2008, 2010, or 2012, respectively.
Name	A character vector giving the name of each representative (in surname, given name format)
Party	a factor identifying the party affiliation of each representative ("Democrat", "Republican", or "Independent").
Experience	character vector highlighting prior experience.
assumedOffice	character vector giving the date assumed office
Born	a character vector giving the year of birth
endOffice	a character vector giving the last day in the present term.
surname	character vector giving the surname of each representative
givenName	given name of each representative (possibly with middle name or initial, a nickname, and a suffix like "Jr.")

Author(s)

Spencer Graves

See Also

getURL readHTMLTable camelParse to remove duplication in Name readUShouse UShouse.senate parseName subNonStandardNames

Examples

```
USsenate <- readUSsenate()
```

---

readUSstateAbbreviations

Read a list of abbreviations of states and territories of the United States

---

Description

Read the list of abbreviations of states and territories of the United states from the relevant Wikipedia article

Usage

```
readUSstateAbbreviations(url.=
"http://en.wikipedia.org/wiki/List_of_U.S._state_abbreviations",
      clean=TRUE, Names=c('Name', 'Status', 'ISO', 'ANSI.letters',
          'ANSI.digits', 'USPS', 'USCG', 'Old.GPO', 'AP', 'Other') )
```

Arguments

`url.`	Universal resource locator to be read and processed to obtain the desired list
`clean`	logical: If TRUE, clean the data using `subNonStandardCharacters` and `strsplit(x, "\[")`
`Names`	names for the columns of the data matrix read; ignored with a warning if the lengths do not match

Details

Wrapper for `readHTMLTable`.

NOTE: `readHTMLTable(url)` returns a list of length 7, only one of which is the table we want. Moreover, that table contains some duplicates, which are removed by `readUSstateAbbreviations`. For example, 'NB' is an "Obsolete postal code" for Nebraska. If you need this, please consult the Wikipedia article.

Value

`readUSstateAbbreviatons` returns a `data.frame` from the table in the Wikipedia article on "List of U.S. state abbreviations".

Author(s)

Spencer Graves

See Also

`getURL` `readHTMLTable` `make.names` `USstateAbbreviations`

Examples

```
abbreviations <- readUSstateAbbreviations()
```

recode2 *bivariate recode*

Description

Recode x1 and x2 per the lexical codes table.

Usage

```
recode2(x1, x2, codes)
```

Arguments

x1, x2	vectors of the same length assuming a discrete number of levels
codes	a 2-dimensional matrix indexed by the levels of x1 and x2. If dimnames(codes) are not provided, they are assumed to unique(x1) (or unique(x2)).

Details

1. If length(x1) != length(x2), complain.
2. if(is.logical(x1)) l1 <- c(FALSE, TRUE) else l1 <- unque(x1); ditto for x2.
3. If(missing(codes)) codes <- outer(unique(x1), unique(x2))
4. if(is.null(dim(codes))) dim(codes) <- c(length(unique(x1)), length(unique(x2)))
5. If is.null(rownames(codes)), set as follows: If nrow(codes) == length(unique(x1)), rownames(codes) <- unique(x1). Else, if nrow(codes) = max(x1), set rownames(codes) <- seq(1, max(x1). Else throw an error. Ditto for colnames, ncol, and x2.
6. codes[x1, x2]

Value

a vector of the same length as x1 and x2.

Author(s)

Spencer Graves

See Also

dim rownames link{colnames}

Examples

```
contrib <- c(-1, 0, 0, 1)
contrib0 <- c(FALSE, FALSE, TRUE, FALSE)

contribCodes <- recode2(contrib>0, contrib0,
   c('returned', 'received', '0', 'ERR') )

cC <- c('returned', 'returned', '0', 'received')

all.equal(contribCodes, cC)
```

---

subNonStandardCharacters

sub nonstandard characters with replacement

Description

Find the first and last character not in `standardCharacters` and replace all between them with `replacement`. For example, a string like "Ruben" where "e" carries and accent and is mangled by some software would become something like "Rub_n" using the default values for `standardCharacters` and `replacement`.

Usage

```
subNonStandardCharacters(x,
   standardCharacters=c(letters, LETTERS, ' ','.', ',', 0:9,
      '\"', "\'", '-', '_', '(', ')', '[', ']', '\n'),
   replacement='_',
   gsubList=list(list(pattern='\\\\\\\\\|\\\\',
      replacement='\"')),
   ... )
```

Arguments

`x`	character vector in which it is desired to find the first and last character not in `standardCharacters` and replace that substring by `replacement`.
`standardCharacters`	a character vector of acceptable characters to keep.
`replacement`	a character to replace the subtring starting and ending with characters not in `standardCharacters`.
`gsubList`	list of lists of `pattern` and `replacement` arguments to be called in succession before looking for nonStandardCharacters
`...`	optional arguments passed to `strsplit`

Details

1. for(il in 1:length(gsubList))x <- gsub(gsubList[[il]][["pattern"]], gsublist[[il]][['replacement']], x)

2. nx <- length(x)

3. x. <- strsplit(x, "", ...)

4. for(ix in 1:nx) find the first and last `standardCharacters` in x.[ix] and substitute `replacement` for everything in between.

Value

a character vector with everthing between the first and last character not in `standardCharacters` replaced by `replacement`.

Author(s)

Spencer Graves

See Also

sub, strsplit, grepNonStandardCharacters, subNonStandardNames encoded_text_to_latex subNonStandardNames

Examples

```
# Consider Names = Ruben, Avila and Jose, where "e" and "A" in
#    these examples carry an accent.  With the default values
#    for standardCharacters and replacement, these would become
#    Rub_en, _vila, and Jos_.
#    (The standard checks for R packages complains about
#    non-standard characters, so none are included here.)
#
Names <- c('Ra`l', 'Ra`', '`l', 'Torres, Raul',
           "Robert C. \\Bobby\\\\")
#  confusion in character sets can create
#  names like Names[2]
Name2 <- subNonStandardCharacters(Names)

Name2. <- c('Ra_l', 'Ra_', '_l', Names[4],
            'Robert C. "Bobby"')


all.equal(Name2, Name2.)
```

subNonStandardNames *sub for nonstandard names*

Description

sub(nonEnglishData["nonEnglish"], nonEnglishData["English"], x)

Usage

```
subNonStandardNames(x,
   standardCharacters=c(letters, LETTERS, ' ','.', ',', 0:9,
      '\"', "\'", '-', '_', '(', ')', '[', ']', '\n'),
   replacement='_',
   gsubList=list(list(pattern='\\\\\\\\\|\\\\',
      replacement='\"')),
   removeSecondLine=TRUE,
   nonStandardNames=Ecdat::nonEnglishNames, ...)
```

Arguments

x	character vector in which it is desired replace nonStandardNames[, 1] in subNonStandardCharacters(x, ...) with the corresponding element of nonStandardNames[, 2].
standardCharacters, replacement, gsubList, ...	arguments passed to subNonStandardCharacters
removeSecondLine	logical: If TRUE, delete anything following "\n" and return it as an attribute "secondLine"
nonStandardNames	data.frame or character matrix with two columns: Replace any substring of x matching nonStandardNames[, 1] with the corresponding elemeng of nonStandardNames[, 2]

Details

1. removeSecondLine

2. x. <- subNonStandardCharacters(x, standardCharacters, replacement, ...)

3. Loop over all rows of nonStandardNames substituting anything matching nonEnglishData[i, 1] with nonEnglishData[i, 2].

4. Eliminate leading and trailing blanks.

Value

a character vector with all nonStandardCharacters replaced first by replacement and then by the second column of nonStandardNames for any that match the first column.

Author(s)

Spencer Graves

See Also

sub nonEnglishNames subNonStandardCharacters

Examples

```
Names <- c('Raul', 'Ra`l', 'Torres,Raul', 'Torres, Raul',
           "Robert C. \\Bobby\\\\", 'Ed  \n --Vacancy')
#  confusion in character sets can create
#  names like Names[2]

library(Ecdat)

Name2 <- subNonStandardNames(Names)
Name2

Name2. <- c('Raul', 'Raul', Names[3:4],
            'Robert C. "Bobby"', 'Ed')
attr(Name2., 'secondLine') <- c(rep(NA, 5), ' --Vacancy')
Name2.


all.equal(Name2, Name2.)
```

`testURLs` *Test URLs for intermittent download problems*

Description

***NOTE: THIS IS A PRELIMINARY VERSION OF THIS FUNCTION; ***NOTE: IT MAY BE CHANGED OR REMOVED IN A FUTURE RELEASE.

`try(getURL(...))` to read each element of `urls`. After each try, write a row to `file`. indicating which of `urls` was tested, the test time in seconds, and any error message. Repeat any failures up to `maxFail` times. After testing each element of `urls` once, repeat `n` times.

If(ping), preceed each test with "ping url[i]". NOTE: Some Internet Service Providers seem to block some attepts to use "ping" or return fraudulet replies to "ping". It is included in the code, because it seemed like an obvious test. However, it is not executed by default because the results do not necessarily reflect what people might expect from "ping".

Return a list of the last successful version read if any from each element of `urls` with two attributes: (1) "urls" containing the `urls` argument. (2) "testResults" being an object of class `c('testURLs', 'data.frame')` of the test results written to `file`..

This function was written to diagnose a download problem with a particular Internet Service Provider (ISP). For other tools for testing an ISP, see measurementlab.net or the "Test your ISP" software discussion by the Electronic Frontier Foundation at the URL mentioned in references below.

Usage

```
testURLs(urls=c(
 wiki="http://en.wikipedia.org",
 wiki.PVI="http://en.wikipedia.org/wiki/Cook_Partisan_Voting_Index",
 house="http://house.gov",
 house.reps="http://house.gov/representatives"),
         file.='testURLresults.csv',
         n=10, maxFail=10, warn=-1, tzone='GMT', ping=FALSE, ...)
```

Arguments

urls	a character vector assumed to be universal resource locators to pass to getURL for testing. The default was selected to provide a 2 x 2 experiment with two different web sites (en.wikipedia.org and house.gov) vs. the landing page and a subordinate page for each site.
file.	Name of a CSV file to which to write the results. If the file already exists, new results are appended to it.
n	number of times to repeat the cycle testing each member of urls.
maxFail	max tests for a continually failing URL. This is designed to make it relatively easy to determine determine dependencies between failures. If the failure rate is constant, the number of consecutive failures will follow a Poisson distribution. Otherwise, it may be possible to evaluate various effects using, e.g., state space techniques for non-normal time series. This could include daily and weekly cycles possibly with holiday effects and trends as well as drifts suggesting abnormal drifts in web traffic congestion.
warn	warn argument to pass to Ping.
tzone	Time zone for Time. Defaults to GMT (UTC). tzone=NULL will use the current locale.
ping	logical: TRUE to include Ping, FALSE otherwise.
...	optional arguments for Ping.

Details

for(i in 1:n):

1. pingi <- Ping(urls[i], ...)

2. The time for each call to getURL is computed by computing `start.time <- proc.time()` before calling `try(getURL(.))`, then computing the following after:

`elapsed.time <- max(proc.time() - start.time, na.rm=TRUE)`

After each of the urls is tested, a summary of the results is appended to file.. This includes the pingi[['stats']], elapsed.time and the error message if the download failed.

The Electronic Frontier Foundation provides a table of existing software to "Test your ISP"; see the references below. This table includes a column noting whether the software is "active" (sending test traffic) or "passive" (observing the way the network treats natural traffic). The current testURLs function is "active", because it asks for a copy of the code at the indicated URL.

Value

an object of class testURLs, which in this case is a list of the last successful result returned by getURL for each element of urls with the following attributes:

urls	the urls argument used for this call
testURLresults	an object of class c('testURLs', 'data.frame') of the data written to file.. This has the following columns:

- Time date() for the time a particular test started
- URL the name in urls of the URL tested
- ping statistics several columns with the count and stats returned by Ping.
- readTime time in seconds for the attempt to read the URL (getURL(urls[j])) to complete.
- error character: " if the read attempt was successful; the error message if not.

Author(s)

Spencer Graves

References

measurementlab.net "Test your ISP" software discussion by the Electronic Frontier Foundation "active" (sending test traffic) or "passive" (observing the way the network treats natural traffic).

See Also

try getURL Ping

Examples

```
# Test only 2 web sites, not the default 4,
# and test only twice, not the default 10 times:
tst <- testURLs(c(
 PVI="http://en.wikipedia.org/wiki/Cook_Partisan_Voting_Index",
 house="http://house.gov/representatives"),
    n=2, maxFail=2)


(class(tst) == 'testURLs') &&
all(names(tst) == c('PVI', 'house')) &&
all(names(attributes(tst)) ==
    c('names', 'urls', 'testURLresults', 'class'))
```

UShouse.senate *Create a list of members of the US House and Senate*

Description

Combine the output of readUShouse and readUSsenate.

Usage

```
UShouse.senate(house=readUShouse(), senate=readUSsenate())
```

Arguments

house, senate	data.frames as returned by the functions readUShouse and readUSsenate, respectively.

Details

Convert the two into a common format and rbind.

Value

a data.frame with the following columns:

Office	A factor identifying "House" vs. "Senate", indicating whether the person is in the US House or Senate
state	A factor identifying the state using the USPS 2-letter state code (all caps)
district	"0" or "At-Large" for members of the US House representing an entire state or integers in character format indicating the district. For the Senate, this contains the "class", which codes the year of the next election for that seat is an integer multiple of 6 years after 2012, 2008, or 2010 for class "1", "2", or "3", respectively.
Party	a factor identifying the party affiliation of each representative, e.g., 'Democratic', 'Republican', 'Democratic-Farmer-Labor', 'Independent'.
surname	family name
givenname	first name with possibly a middle name, nickname, and suffix (e.g., Jr., III).

Author(s)

Spencer Graves

See Also

readUShouse readUSsenate

Examples

```
house <- readUShouse()

USreps <- UShouse.senate(house)
```

`USsenateClass` *Election Class given state and surname of a US Senator*

Description

For all indiviuals in `x` with `houseSenate` == "Senate", look up their `state` and `surname` in the reference table `senate` and return their `Class`. For indivuals not found in `senate`, return `x[[district]]`.

Senate classes 1, 2 and 3 have their normal elections in 6-year cycles including 2000, 2002, and 2004 (or 2012, 2008, and 2010), respectively. When vacancies occur out of cycle, the vacancy is first filled with appointment by the governor of the state, and an election to fill that seat occurs in the next even-numbered year; the class of that seat does not change.

For example, South Carolina Senator Jim DeMint resigned effective January 1, 2013. South Carolina Governor Nikki Haley appointed Tim Scott to serve until a special election in 2014. This is a Class 3 seat, which means that another election for that seat will occur in 2016.

Usage

```
USsenateClass(x, senate=readUSsenate(),
   Office='Office', state='state',
   surname='surname', district='district', senatePattern='^Senate')
```

Arguments

`x`	`data.frame` with character or factor columns `Office`, `state`, `surname`, and `district`.
`senate`	`data.frame` as returned by `readUSsenate`.
`Office`	name of a character or factor variable `x` in which the members of the US Senate can be identifed by `grep(senatePattern, x[, Office])`.
`state`	Standard 2-letter abbreviation for the state of the US
`surname`	the name of a column of `x` containing the surname
`district`	name of a column of `x` containing the number of the district in the US House. For states with only one representataive, this may be 0.
`senatePattern`	a regular expression for identifying the senators from `x[, Office]`.

Details

The current algorithm may fail if both senators in a `state` have the same `surname`.

Value

a data.frame with one row for each row of x and the following columns:

incumbent — logical vector: NA if Office == 'house'. If Office == 'senate', then TRUE if state:surname found in senate and FALSE otherwise.

District — a character vector containing the desired Class for all US Senators found in senate or a guess at the Class for non-incumbents. For members of the House, this returned the previous content of x[[District]].

NOTES:

1. Incumbents can be missed if the spelling of the surname is different between x and senate. This can occur with, for example, Spanish surnames containing an accent.

2. If one but not two incumbents is found, others are currently assigned to the class of an incumbent not found. This could be a mistake, because the person could be a previous incumbent or could have lost to the incumbent in the last election.

Author(s)

Spencer Graves

See Also

readUSsenate

Examples

```
tst <- data.frame(Office=factor(c("House", "Senate", "Senate", 'Senate')),
                  state=factor(c('SC', 'SC', 'SC', 'NY')),
                  surname=c("Jones", "DeMint", "Graham", 'Smith'),
                  district=c("9", NA, NA, NA),
                  stringsAsFactors=FALSE)
tst. <- USsenateClass(tst)

chk <- data.frame(incumbent=c(NA, FALSE, TRUE, FALSE),
                  district=c("9", "3", "2", "1 or 3"),
                  stringsAsFactors=FALSE)

all.equal(tst., chk)


##
## test with names different from the default
##
tst2 <- tst
names(tst2) <- letters[1:4]
tst2. <- USsenateClass(tst2, Office='a',
           state='b', surname='c', district='d')

all.equal(tst., tst2.)
```

Index