Package 'DataCombine'

February 19, 2015

Title Tools for Easily Combining and Cleaning Data Sets

Description Tools for combining and cleaning data sets, particularly with grouped and time series data.

Version 0.2.9

Date 2015-02-11

License GPL (>= 3)

URL http://christophergandrud.github.io/DataCombine/

BugReports https://github.com/christophergandrud/DataCombine/issues

Depends R (>= 3.0.2)

Imports data.table, dplyr (>= 0.3)

Suggests devtools, testthat

Author Christopher Gandrud [aut, cre]

Maintainer Christopher Gandrud <christopher.gandrud@gmail.com>

NeedsCompilation no

Repository CRAN

Date/Publication 2015-02-11 10:39:41

R topics documented:

`CountSpell` *Count spells, including for grouped data*

Description

`CountSpell` is a function that returns a variable counting the spell number for an observation. Works with grouped data.

Usage

```
CountSpell(data, TimeVar, SpellVar, GroupVar, NewVar, SpellValue)
```

Arguments

`data`	a data frame object.
`TimeVar`	a character string naming the time variable.
`SpellVar`	a character string naming the variable with information on when each spell starts.
`GroupVar`	a character string naming the variable grouping the units experiencing the spells.
`NewVar`	NewVar a character string naming the new variable to place the spell counts in.
`SpellValue`	a value indicating when a unit is in a spell. If `SpellValue` is missing then any change in `Var`'s value will be treated as the start/end of a spell.

Examples

```
# Create fake data
ID <- sort(rep(seq(1:4), 5))
Time <- rep(1:20)
Dummy <-  c(1, sample(c(0, 1), size = 19, replace = TRUE))
Data <- data.frame(ID, Time, Dummy)

# Find spell for whole data frame
DataSpell1 <- CountSpell(Data, TimeVar = 'Time', SpellVar = 'Dummy',
                         SpellValue = 1)
```

```
head(DataSpell1)

# Find spell for each ID group
DataSpell2 <- CountSpell(Data, TimeVar = 'Time', SpellVar = 'Dummy',
                         GroupVar = 'ID', SpellValue = 1)

head(DataSpell2)
```

---

dMerge *Merges 2 data frames and report/drop/keeps only duplicates.*

Description

dMerge merges 2 data frames and reports/drops/keeps only duplicates.

Usage

```
dMerge(data1, data2, Var, dropDups = TRUE, dupsOut = FALSE,
  fromLast = FALSE, all = FALSE, all.x = all, all.y = all,
  sort = TRUE, suffixes = c(".x", ".y"), incomparables = NULL)
```

Arguments

data1	a data frame. The first data frame to merge.
data2	a data frame. The second data frame to merge.
Var	character vector containing the names of the variables merge by. See merge.
dropDups	logical. Whether or not to drop duplicated rows based on Var. If dropDups = FALSE then it gives a count of the duplicated rows.
dupsOut	logical. If TRUE then a data frame only containing duplicated values is returned and dropDups is ignored.
fromLast	logical indicating if duplication should be considered from the reverse side. Only relevant if dropDups = TRUE.
all	logical; all = L is shorthand for all.x = L and all.y = L, where L is either TRUE or FALSE.
all.x	logical; if TRUE, then extra rows will be added to the output, one for each row in x that has no matching row in y. These rows will have NAs in those columns that are usually filled with values from y. The default is #' FALSE, so that only rows with data from both x and y are included in the output.
all.y	logical; analogous to all.x.
sort	logical. Should the result be sorted on the by columns?
suffixes	a character vector of length 2 specifying the suffixes to be used for making unique the names of columns in the result which not used for merging (appearing in by etc).
incomparables	values which cannot be matched. See match.

See Also

duplicated, merge

DropNA *Drop rows from a data frame with missing values on a given variable(s).*

Description

DropNA drops rows from a data frame when they have missing (NA) values on a given variable(s).

Usage

```
DropNA(data, Var, message = TRUE)
```

Arguments

data	a data frame object.
Var	a character vector naming the variables you would like to have only non-missing (NA) values.
message	logical. Whether or not to give you a message about the number of rows that are dropped.

Source

Partially based on Stack Overflow answer written by donshikin: http://stackoverflow.com/questions/4862178/remove-rows-with-nas-in-data-frame

Examples

```
# Create data frame
a <- c(1, 2, 3, 4, NA)
b <- c( 1, NA, 3, 4, 5)
ABData <- data.frame(a, b)

# Remove missing values from column a
ASubData <- DropNA(ABData, Var = "a", message = FALSE)

# Remove missing values in columns a and b
ABSubData <- DropNA(ABData, Var = c("a", "b"))
```

FillDown *Fills in missing (NA) values with the previous non-missing value*

Description

Fills in missing (NA) values with the previous non-missing value

Usage

```
FillDown(data, Var)
```

Arguments

data	a data frame. Optional as you can simply specify a vector with Var,
Var	the variable in data or a vector you would like to fill down missing (NA) values.

Value

data frame

Examples

```
# Create fake data
id <- c('Algeria', NA, NA, NA, 'Mexico', NA, NA)
score <- rnorm(7)
Data <- data.frame(id, score)

# FillDown id
DataOut <- FillDown(Data, 'id')

## Not run:
# Use group_by and mutate from dplyr to FillDown grouped data, e.g.:
Example <- Example %>% group_by(grouping) %>%
             mutate(NewFilled = FillDown(Var = VarToFill))

## End(Not run)
```

FillIn *A function for filling in missing values of a variable from one data frame with the values from another variable.*

Description

FillIn uses values of a variable from one data set to fill in missing values in another.

Usage

```
FillIn(D1, D2, Var1, Var2, KeyVar = c("iso2c", "year"),
  allow.cartesian = FALSE, KeepD2Vars = FALSE)
```

Arguments

D1	the data frame with the variable you would like to fill in.
D2	the data frame with the variable you would like to use to fill in D1.
Var1	a character string of the name of the variable in D1 you want to fill in.
Var2	an optional character string of variable name in D2 that you would like to use to fill in. Note: must be of the same class as Var1.
KeyVar	a character vector of variable names that are shared by D1 and D2 that can be used to join the data frames.
allow.cartesian	logical. See the data.table documentation for more details.
KeepD2Vars	logical, indicating whether or not to keep the variables from D2 in the output data frame. The default is KeepD2Vars = FALSE. Hint: avoid having variables in your D2 data frame that share names with variables in D1 other than the KeyVars

Examples

```
# Create data set with missing values
naDF <- data.frame(a = sample(c(1,2), 100, rep = TRUE),
                   b = sample(c(3,4), 100, rep = TRUE),
                   fNA = sample(c(100, 200, 300, 400, NA), 100, rep = TRUE))

# Created full data set
fillDF <- data.frame(a = c(1, 2, 1, 2),
                     b = c(3, 3, 4, 4),
                     j = c(5, 5, 5, 5),
                     fFull = c(100, 200, 300, 400))

# Fill in missing f's from naDF with values from fillDF
FilledInData <- FillIn(naDF, fillDF, Var1 = "fNA",
                       Var2 = "fFull", KeyVar = c("a", "b"))
```

FindReplace *Replace multiple patterns found in a character string column of a data frame*

Description

FindReplace allows you to find and replace multiple character string patterns in a data frame's column.

Usage

```
FindReplace(data, Var, replaceData, from = "from", to = "to",
  exact = TRUE, vector = FALSE)
```

Arguments

data	data frame with the column you would like to replace string patterns.
Var	character string naming the column you would like to replace string patterns. The column must be of class character or factor.
replaceData	a data frame with at least two columns. One contains the patterns to replace and the other contains their replacement. Note: the pattern and its replacement must be in the same row.
from	character string naming the column with the patterns you would like to replace.
to	character string naming the column with the the pattern replacements.
exact	logical. Indicates whether to only replace exact pattern matches (TRUE) or not (FALSE).
vector	logical. If TRUE then the replacement is returned as a single vector. If FALSE then the whole data frame is returned.

Examples

```
# Create original data
ABData <- data.frame(a = c("London, UK", "Oxford, UK", "Berlin, DE",
                     "Hamburg, DE", "Oslo, NO"),
                     b = c(8, 0.1, 3, 2, 1))

# Create replacements data frame
Replaces <- data.frame(from = c("UK", "DE"), to = c("England", "Germany"))

# Replace patterns and return full data frame
ABNewDF <- FindReplace(data = ABData, Var = "a", replaceData = Replaces,
                     from = "from", to = "to", exact = FALSE)

# Replace patterns and return the Var as a vector
ABNewVector <- FindReplace(data = ABData, Var = "a", replaceData = Replaces,
                     from = "from", to = "to", vector = TRUE)
```

---

grepl.sub *Subset a data frame if a specified pattern is found in a character string*

Description

Subset a data frame if a specified pattern is found in a character string

Usage

```
grepl.sub(data, pattern, Var, keep.found = TRUE, useBytes = TRUE)
```

Arguments

data	data frame.
pattern	character vector containing a regular expression to be matched in the given character vector.
Var	character vector of the variables that the pattern should be found in.
keep.found	logical. whether or not to keep observations where the pattern is found (TRUE) or not found (FALSE).
useBytes	logical. If TRUE the matching is done byte-by-byte rather than character-by-character. See grep.

Examples

```
# Create data frame
ABData <- data.frame(a = c("London, UK", "Oxford, UK", "Berlin, DE",
                    "Hamburg, DE", "Oslo, NO"),
                    b = c(8, 0.1, 3, 2, 1))

# Keep only data from Germany (DE)
ABGermany <- grepl.sub(data = ABData, pattern = "DE", Var = "a")
```

InsertRow *Inserts a new row into a data frame*

Description

Inserts a new row into a data frame

Usage

```
InsertRow(data, NewRow, RowNum = NULL)
```

Arguments

data	a data frame to insert the new row into.
NewRow	a vector whose length is the same as the number of columns in data.
RowNum	numeric indicating which row to insert the new row as. If not specified then the new row is added to the end using a vanilla rbind call.

Source

The function largely implements: http://stackoverflow.com/a/11562428

Examples

```
# Create dummy data
A <- B <- C <- D <- sample(1:20, size = 20, replace = TRUE)
Data <- data.frame(A, B, C, D)

# Create new row
New <- rep(1000, 4)

# Insert into 4th row
Data <- InsertRow(Data, NewRow = New, RowNum = 4)
```

MoveFront *Move variables to the front of a data frame.*

Description

MoveFront moves variables to the front of a data frame.

Usage

```
MoveFront(data, Var, exact = TRUE, ignore.case = NULL, fixed = NULL)
```

Arguments

data	a data frame object containing the variable you want to move.
Var	a character vector naming the variables you would like to move to the front of the data frame. The order of the variables should match the order you want them to have in the data frame, i.e. the first variable in the vector will be the first variable in the data frame.
exact	logical. If TRUE (the default), only exact variable names are matched.
ignore.case	logical. If FALSE, the variable name matching is case sensitive and if TRUE, case is ignored during matching. Only available when exact = FALSE.
fixed	logical. If TRUE, pattern is a string to be matched as is. Overrides all conflicting arguments. Only available when exact = FALSE.

Source

Based primarily on a Stack Overflow answer written by rcs: http://stackoverflow.com/questions/3369959/moving-columns-within-a-data-frame-without-retyping.

Examples

```
# Create fake data
A <- B <- C <- 1:50
OldOrder <- data.frame(A, B, C)

# Move C to front
```

```
NewOrder1 <- MoveFront(OldOrder, "C")
names(NewOrder1)

# Move B and A to the front
NewOrder2 <- MoveFront(OldOrder, c("B", "A"))
names(NewOrder2)

## Non-exact matching (example from Felix Hass)
 # Create fake data
 df <- data.frame(dummy = c(1,0), Name = c("Angola", "Chad"),
                  DyadName = c("Government of Angola - UNITA",
                  "Government of Chad - FNT"),
                  Year = c("2002", "1992"))

 df <- MoveFront(df, c("Name", "Year"), exact = FALSE)

 names(df)

 df <- MoveFront(df, c("Name", "Year"), exact = TRUE)

 names(df)
```

NaVar *Create new variable(s) indicating if there are missing values in other variable(s)*

Description

Create new variable(s) indicating if there are missing values in other variable(s)

Usage

```
NaVar(data, Var, Stub = "Miss_", reverse = FALSE, message = TRUE)
```

Arguments

data	a data frame object.
Var	a character vector naming the variable(s) within which you would like to identify missing values.
Stub	a character string indicating the stub you would like to append to the new variables' name(s).
reverse	logical. If `reverse = FALSE` then missing values are coded as `1` and non-missing values are coded as `0`. If `reverse = TRUE` then missing values are coded as `0` and non-missing values are coded as `1`.
message	logical. Whether or not to give you a message about the names of the new variables that are created.

Examples

```
# Create data frame
a <- c(1, 2, 3, 4, NA)
b <- c( 1, NA, 3, 4, 5)
ABData <- data.frame(a, b)

# Create varibles indicating missing values in columns a and b
ABData1 <- NaVar(ABData, Var = c('a', 'b'))

# Create varible indicating missing values in columns a with reversed dummy
ABData2 <- NaVar(ABData, Var = 'a', reverse = TRUE, message = FALSE)
```

`PercChange` *Calculate the percentage change from a specified lag, including withing group*

Description

Calculate the percentage change from a specified lag, including withing group

Usage

```
PercChange(data, Var, GroupVar, NewVar, slideBy = -1, type = "percent", ...)
```

Arguments

`data`	a data frame object.
`Var`	a character string naming the variable you would like to find the percentage change for.
`GroupVar`	a character string naming the variable grouping the units within which the percentage change will be found for (i.e. countries in a time series). If `GroupVar` is missing then the entire data frame is treated as one unit.
`NewVar`	a character string specifying the name for the new variable to place the percentage change in.
`slideBy`	numeric value specifying how many rows (time units) to make the percentage change comparison for. Positive values shift the data up–lead the data.
`type`	character string set at either `percent` for percentages or `proportion` to find proportions.
`...`	arguments passed to `slide`.

Details

Finds the percentage or proportion change for over a given time period either within groups of data or the whole data frame. Important: the data must be in time order and, if groups are used, group-time order.

Value

a data frame

Examples

```
# Create fake data frame
A <- c(1, 1, 1, 1, 1, 2, 2, 2, 2, 2)
B <- c(1:10)
Data <- data.frame(A, B)

# Find percentage change from two periods before
Out <- PercChange(Data, Var = 'B',
         type = 'proportion',
         NewVar = 'PercentChange',
         slideBy = -2)

Out
```

---

rmExcept *Remove all objects from a workspace except those specified by the user.*

Description

rmExcept removes all objects from a workspace except those specified by the user.

Usage

```
rmExcept(keepers, envir = globalenv(), message = TRUE)
```

Arguments

keepers	a character vector of the names of object you would like to keep in your workspace.
envir	the environment to remove objects from. The default is the global environment (i.e. globalenv).
message	logical, whether or not to return a message informing the user of which objects were removed.

Examples

```
# Create objects
A <- 1; B <- 2; C <- 3

# Remove all objects except for A
rmExcept("A")

# Show workspace
ls()
```

shift *A function for creating lag and lead variables.*

Description

The function shifts a vector up or down to create lag or lead variables. Note: your data needs to be sorted by date. The date should be ascending (i.e. increasing as it moves down the rows).

Usage

```
shift(VarVect, shiftBy, reminder = TRUE)
```

Arguments

VarVect	a vector you would like to shift (create lag or lead).
shiftBy	numeric value specifying how many rows (time units) to shift the data by. Negative values shift the data down–lag the data. Positive values shift the data up–lead the data.
reminder	logical. Whether or not to remind you to order your data by the GroupVar and time variable before running shift.

Details

shift a function for creating lag and lead variables, including for time-series cross-sectional data.

Value

a vector

Source

Largely based on TszKin Julian's shift function: http://ctszkin.com/2012/03/11/generating-a-laglead-variable

See Also

slide

shiftMA *Internal function for slideMA*

Description

Internal function for slideMA

Usage

```
shiftMA(x, shiftBy, Abs, reminder)
```

Arguments

x	vector
shiftBy	numeric
Abs	numeric
reminder	logical

slide *A function for creating lag and lead variables, including for time-series cross-sectional data.*

Description

The function slides a column up or down to create lag or lead variables. If GroupVar is specified it will slide Var for each group. This is important for time-series cross-section data. The slid data is placed in a new variable in the original data frame. Note: your data needs to be sorted by date. The date should be ascending (i.e. increasing as it moves down the rows). Also, the time difference between rows should be constant, e.g. days, months, years.

Usage

```
slide(data, Var, TimeVar, GroupVar, NewVar, slideBy = -1,
  keepInvalid = FALSE, reminder = TRUE)
```

Arguments

data	a data frame object.
Var	a character string naming the variable you would like to slide (create lag or lead).
TimeVar	optional character string naming the time variable. If specified then the data is ordered by Var-TimeVar before sliding.
GroupVar	a character string naming the variable grouping the units within which Var will be slid. If GroupVar is missing then the whole variable is slid up or down. This is similar to shift, though shift returns the slid data to a new vector rather than the original data frame.

NewVar	a character string specifying the name for the new variable to place the slid data in.
slideBy	numeric value specifying how many rows (time units) to shift the data by. Negative values slide the data down–lag the data. Positive values shift the data up–lead the data.
keepInvalid	logical. Whether or not to keep observations for groups for which no valid lag/lead can be created due to an insufficient number of time period observations. If TRUE then these groups are returned to the bottom of the data frame and NA is given for their new lag/lead variable value.
reminder	logical. Whether or not to remind you to order your data by the GroupVar and time variable before running slide, plus other messages.

Details

slide a function for creating lag and lead variables, including for time-series cross-sectional data.

Value

a data frame

Source

Partially based on TszKin Julian's shift function: http://ctszkin.com/2012/03/11/generating-a-laglead-variable

See Also

shift, dplyr

Examples

```
# Create dummy data
A <- B <- C <- sample(1:20, size = 20, replace = TRUE)
ID <- sort(rep(seq(1:4), 5))
Data <- data.frame(ID, A, B, C)

# Lead the variable by two time units
DataSlid1 <- slide(Data, Var = "A", NewVar = "ALead", slideBy = 2)

# Lag the variable one time unit by ID group
DataSlid2 <- slide(data = Data, Var = "B", GroupVar = "ID",
                NewVar = "BLag", slideBy = -1)

# Lag the variable one time unit by ID group, with invalid lags
Data <- Data[1:16, ]

DataSlid3 <- slide(data = Data, Var = "B", GroupVar = "ID",
                 NewVar = "BLag", slideBy = -2, keepInvalid = TRUE)
```

slideMA *Create a moving average for a period before or after each time point for a given variable*

Description

Create a moving average for a period before or after each time point for a given variable

Usage

```
slideMA(data, Var, GroupVar, periodBound = -3, offset = 1, NewVar,
  reminder = TRUE)
```

Arguments

data	a data frame object.
Var	a character string naming the variable you would like to create the lag/lead moving averages from.
GroupVar	a character string naming the variable grouping the units within which Var will be turned into slid moving averages. If GroupVar is missing then the whole variable is slid up or down and moving averages will be created. This is similar to shift, though shift returns the slid data to a new vector rather than the original data frame.
periodBound	integer. The time point for the outer bound of the time period over which to create the moving averages. The default is -3, i.e. the moving average period begins three time points before a given time point. Can also be positive for leading moving averages.
offset	integer. How many time increments away from a given time point to begin the moving average time period. The default is 1. Effectively controls how wide the moving average window is in the other direction of periodBound. Note: must be positive.
NewVar	a character string specifying the name for the new variable to place the slid data in.
reminder	logical. Whether or not to remind you to order your data by the GroupVar and time variable before running slideMA.

Details

slideMA is designed to give you more control over the window for creating the moving average. Think of the periodBound and offset arguments working together. If for example, periodBound = -3 and offset = 1 then the variable of interest will be lagged by 2 then a moving average window of three time increments around the lagged variable is found.

Value

a data frame

See Also

shift, slide, dplyr

Examples

```
# Create dummy data
 A <- B <- C <- sample(1:20, size = 20, replace = TRUE)
 ID <- sort(rep(seq(1:4), 5))
 Data <- data.frame(ID, A, B, C)

 # Lead the variable by two time units
 DataSlidMA1 <- slideMA(Data, Var = "A", NewVar = "ALead_MA",
                 periodBound = 3)

 # Lag the variable one time unit by ID group
 DataSlidMA2 <- slideMA(data = Data, Var = "B", GroupVar = "ID",
                NewVar = "BLag_MA", periodBound = -3, offset = 2)
```

SpreadDummy *Spread a dummy variable (1's and 0') over a specified time period and for specified groups*

Description

Spread a dummy variable (1's and 0') over a specified time period and for specified groups

Usage

```
SpreadDummy(data, Var, GroupVar, NewVar, spreadBy = -2, reminder = TRUE)
```

Arguments

data	a data frame object.
Var	a character string naming the numeric dummy variable with values 0 and 1 that you would like to spread. Can be either spread as a lag or lead.
GroupVar	a character string naming the variable grouping the units within which Var will be spread If GroupVar is missing then the whole variable is spread up or down. This is similar to shift, though shift slides the data and returns it to a new vector rather than the original data frame.
NewVar	a character string specifying the name for the new variable to place the spread dummy data in.
spreadBy	numeric value specifying how many rows (time units) to spread the data over. Negative values spread the data down–lag the data. Positive values spread the data up–lead the data.
reminder	logical. Whether or not to remind you to order your data by the GroupVar and time variable before running SpreadDummy.

See Also

slide

Examples

```
# Create dummy data
ID <- sort(rep(seq(1:4), 5))
NotVar <- rep(1:5, 4)
Dummy <-  sample(c(0, 1), size = 20, replace = TRUE)
Data <- data.frame(ID, NotVar, Dummy)

# Spread
DataSpread1 <- SpreadDummy(data = Data, Var = 'Dummy',
                           spreadBy = 2, reminder = FALSE)

DataSpread2 <- SpreadDummy(data = Data, Var = 'Dummy', GroupVar = 'ID',
                           spreadBy = -2)
```

StartEnd *Find the starting and ending time points of a spell*

Description

StartEnd finds the starting and ending time points of a spell, including for time-series cross-sectional data. Note: your data needs to be sorted by date. The date should be ascending (i.e. increasing as it moves down the rows).

Usage

```
StartEnd(data, SpellVar, GroupVar, SpellValue, OnlyStart = FALSE)
```

Arguments

data	a data frame object.
SpellVar	a character string naming the variable with information on when each spell starts.
GroupVar	a character string naming the variable grouping the units experiencing the spells. If GroupVar is missing then .
SpellValue	a value indicating when a unit is in a spell. If SpellValue is missing then any change in Var's value will be treated as the start/end of a spell. Must specify if OnlyStart = TRUE.
OnlyStart	logical for whether or not to only add a new Spell_Start variable. Please see the details.

Value

a data frame. If `OnlyStart = FALSE` then two new variables are returned:

- Spell_Start: The time period year of a given spell.
- Spell_End: The end time period of a given spell.

If `OnlyStart = TRUE` then only `Spell_Start` is added. This variable includes both 1's for the start of a new spell and for the start of a 'gap spell', i.e. a spell after `Spell_End`.

See Also

`slide`

Examples

```
# Create fake data
ID <- sort(rep(seq(1:4), 5))
Time <- rep(1:5, 4)
Dummy <-  c(1, sample(c(0, 1), size = 19, replace = TRUE))
Data <- data.frame(ID, Time, Dummy)

# Find start/end of spells denoted by Dummy = 1
DataSpell <- StartEnd(Data, SpellVar = 'Dummy', GroupVar = 'ID',
                      SpellValue = 1)

head(DataSpell)
```

TimeExpand *Expands a data set so that it includes an observation for each time point in a sequence. Works with grouped data.*

Description

Expands a data set so that it includes an observation for each time point in a sequence. Works with grouped data.

Usage

```
TimeExpand(data, GroupVar, TimeVar, begin, end, by = 1)
```

Arguments

`data`	a data frame.
`GroupVar`	the variable in `data` that signifies the group variable.
`TimeVar`	the variable in `data` that signifies the time variable. The sequence will be expanded between its minimum and maximum value if `begin` and `end` are not specified.

begin	numeric of length 1. Specifies beginning time point. Only relevant if end is specified.
end	numeric of length 1. Specifies ending time point. Only relevant if begin is specified.
by	numeric or character string specifying the steps in the TimeVar sequence. Can use "month", "year" etc for POSIXt data.

Examples

```
Data <- data.frame(country = c("Cambodia", "Camnodia", "Japan", "Japan"),
                   year = c(1990, 2001, 1994, 2012))

ExpandedData <- TimeExpand(Data, GroupVar = 'country', TimeVar = 'year')
```

TimeFill *Creates a continuous Unit-Time-Dummy data frame from a data frame with Unit-Start-End times*

Description

Creates a continuous Unit-Time-Dummy data frame from a data frame with Unit-Start-End times

Usage

```
TimeFill(data, GroupVar, StartVar, EndVar, NewVar = "TimeFilled",
  NewTimeVar = "Time", KeepStartStop = FALSE)
```

Arguments

data	a data frame with a Group, Start, and End variables.
GroupVar	a character string naming the variable grouping the units within which the new dummy variable will be found.
StartVar	a character string indicating the variable with the starting times of some series.
EndVar	a character string indicating the variable with the ending times of some series.
NewVar	a character string specifying the name of the new dummy variable for the series. The default is TimeFilled.
NewTimeVar	a character string specifying the name of the new time variable. The default is Time.
KeepStartStop	logical indicating whether or not to keep the StartVar and EndVar variables in the output data frame.

Value

Returns a data frame with at least three columns, with the GroupVar, NewTimeVar, and a new dummy variable with the name specified by NewVar. This variable is 1 for every time increment between and including StartVar and EndVar. It is 0 otherwise.

Examples

```
# Create fake data

Country = c('Panama', 'Korea', 'Korea', 'Germany', 'Finland')
Start = c(1995, 1980, 2004, 2000, 2012)
End = c(1995, 2001, 2010, 2002, 2014)

Data <- data.frame(Country, Start, End)

# TimeFill
FilledData <- TimeFill(Data, GroupVar = 'Country',
                 StartVar = 'Start', EndVar = 'End')

# Show selection from TimeFill-ed data
FilledData[90:100, ]
```

VarDrop *Drop one or more variables from a data frame.*

Description

VarDrop drops one or more variables from a data frame.

Usage

```
VarDrop(data, Var)
```

Arguments

data	a data frame.
Var	character vector containing the names of the variables to drop.

Examples

```
# Create dummy data
a <- c(1, 2, 3, 4, NA)
b <- c( 1, NA, 3, 4, 5)
c <- c(1:5)
d <- c(1:5)
ABCData <- data.frame(a, b, c, d)

# Drop a and b
DroppedData <- VarDrop(ABCData, c('b', 'c'))
```

Index