Package: dplyrExtras (via r-universe)

August 30, 2024

Type Package

Title extra functionality for dplyr like mutate_rows for mutation of a subset of rows

Version 0.1.4

Date 2020-05-05

Author Sebastian Kranz

Maintainer Sebastian Kranz <sebastian.kranz@uni-ulm.de>

Description Some extra functionality that is not (yet) in dplyr, e.g. mutate_rows (mutation of subset of rows) xsummarise_each (summarise_each with more flexible alignment of results), or s_filter, s_arrange ,... that allow string arguments.

License GPL>=2.0

Depends data.table,dplyr, dtplyr

RoxygenNote 5.0.0

Repository https://skranz.r-universe.dev

RemoteUrl https://github.com/skranz/dplyrExtras

RemoteRef master

RemoteSha 8b05cf2a3957167484ec31add14fdb3b534b6995

Contents

`modify` *In place modification of data tables*

Description

If dt is a data table, then modify is essentially just a wrapper for data.table syntax that allows modification or creation of new columns. If dt is not a data table, it will by default be converted to a data table and then transformed and returned as the original data frame. Unlike mutate from dplyr, one can use the .SD argument in function calls, which can quite useful sometimes.

Usage

```
modify(.dt, .if, .by = NULL, ..., .envir = parent.frame(),
  .inplace = is.data.table(.dt), .as.data.table = is.data.table(.dt))
```

Arguments

`.dt`	a data.table
`.if`	optional a boolean conditions that specifies the rows that shall be modifed
`.by`	optional a vector of column names used for computations that are splitted by groups
`...`	formulas for columns that are modified or newly created
`.envir`	optional an environment in which the expressions shall be evaluated if variables are not found in .dt
`.inplace`	allows .dt inplace modification (TRUE if .dt is a data table)
`.as.data.table`	shall result be a data.table (only true if .dt is a data.table)

Examples

```
## Not run:
  library(microbenchmark)

  K = 3
  n = 10
  dt = data.table(a= sample(1:3,n,replace=TRUE),
                  b= sample(1:100,n,replace=TRUE),
                  x=rnorm(n))
  df = as.data.frame(dt)
  # Set x to 100 where a==2
  modify(dt,a==2, y=x+100, z=y+K)
```

```
modify(df,a==2,y=200,z=y*5+K)

dt[,y:=x+2]

# Set x to the mean value of b*100 in each group of a
modify(dt,.by=c("a"), x=mean(b)*100)
dt
# Call with strings
com = "x=200"
s_modify(dt,"a==2", com)
dt



## End(Not run)
```

`mutate_if` *mutate selected rows*

Description

Old pseudonym for mutate_rows. Originally the function mutate_rows was called mutate_if, however, dplyr 5.0 introduced also a function called mutate_if that does something different however. I keep the synonym mutate_if in order to reduce the probability that old code breaks, but it is recommended to use mutate_rows instead.#'

Usage

```
mutate_if(.data, .if, ...)
```

Arguments

`.data`	the data
`.if`	a logical condition that selects rows, e.g. a=="B"
`...`	the command to mutate existing columns

`mutate_rows` *mutate selected rows*

Description

change values of columns only in rows that satisfy the .if condition Note: you cannot create new columns with mutate_rows but only change values in selected rows of existing columns

Usage

```
mutate_rows(.data, .if, ...)
```

Arguments

.data	the data
.if	a logical condition that selects rows, e.g. a=="B"
...	the command to mutate existing columns

Examples

```
## Not run:
  library(microbenchmark)
  library(dplyr)
  library(data.table)
  library(dplyrExtras)

  # create a data
  set.seed(123456)
  n = 10
  df = data.frame(a= sample(1:3,n,replace=TRUE),
  b= sample(1:100,n,replace=TRUE),
  x=rnorm(n))
  dt = as.data.table(df)

  # different calls to mutate_rows
  mutate_rows(df,a==3,y=100)
  mutate_rows(tbl_df(df),a==1,x=200)
  mutate_rows(as.tbl(df),a==1,x=300,b=400)
  mutate_rows(dt,a==1 | a==2,x=400)
  mutate_rows(group_by(dt,a),a==1 | a==2,x=mean(b))
  mutate_rows(group_by(df,a),a==1 | a==2,x=mean(b))

  # if you create a new column rows that don't
  # match the if condition have an NA
  mutate_rows(df,a==3,z=100)

  # You can only have one if condition in a mutate_rows call
  # So multiple changes require nesting or piping
  library(magrittr)
  df
         mutate_rows(a==2,z=200)

  # For historical reasons there is also still the synonym
  # mutate_if. But not that dplyr 5.0 has introduced its own
  # mutate_if function with quite different functionality

  mutate_if(df,a==3,y=99)
```

```
  # Small benchmark: compare with mutate + ifelse
  n = 1e6
  df = data.frame(a= sample(1:3,n,replace=TRUE),
  b= sample(1:100,n,replace=TRUE),
  x=rnorm(n))
  microbenchmark(times = 5L,
    mutate(df, x=ifelse(a==2,x+100,x)),
    mutate(df, x=if_else(a==2,x+100,x)),
    mutate_rows(df, a==2, x=x+100)
  )
#             mean
# ifelse      540.7145
# if_else     360.4928
# mutate_rows 114.3891

## End(Not run)
```

`remove_cols` *Inplace remove columns from a data.table*

Description

Inplace remove columns from a data.table

Usage

```
remove_cols(dt, cols)
```

Arguments

`dt`	a data.table
`cols`	a string with column names

Examples

```
## Not run:
  dt = data.table(a=1:3, b=1:3, c=4:6)
  dt
  remove_cols(dt,c("a" ,"b"))
  dt

## End(Not run)
```

s_arrange *Modified version of dplyr's arrange that uses string arguments*

Description

Modified version of dplyr's arrange that uses string arguments

Usage

```
s_arrange(.data, ...)
```

Arguments

.data	the data frame (or similar object)
...	string version of arguments to original dplyr function

Examples

```
## Not run:
  library(dplyrExtras)

  d = mtcars
  cols = c("mpg","cyl","hp:vs")
  s_select(d,cols)
  # also works and yields identical result...
  cols = c("mpg","cyl, hp:vs")
  s_select(d,cols)

  s_filter(d,"gear == 3","cyl == 8")
  s_arrange(d, "-mpg, gear, carb")
  gd = s_group_by(d,"cyl")
  s_summarise(gd, "mean(disp), max(disp)")
  s_mutate_if(d, "cyl==6, new.col=100")


## End(Not run)
```

s_filter *Modified version of dplyr's filter that uses string arguments*

Description

Modified version of dplyr's filter that uses string arguments

Usage

```
s_filter(.data, ...)
```

Arguments

.data	the data frame (or similar object)
...	string version of arguments to original dplyr function

Examples

```
## Not run:
  library(dplyrExtras)

  d = mtcars
  cols = c("mpg","cyl","hp:vs")
  s_select(d,cols)
  # also works and yields identical result...
  cols = c("mpg","cyl, hp:vs")
  s_select(d,cols)

  s_filter(d,"gear == 3","cyl == 8")
  s_arrange(d, "-mpg, gear, carb")
  gd = s_group_by(d,"cyl")
  s_summarise(gd, "mean(disp), max(disp)")
  s_mutate_if(d, "cyl==6, new.col=100")


## End(Not run)
```

---

s_group_by *Modified version of dplyr's group_by that uses string arguments*

Description

Modified version of dplyr's group_by that uses string arguments

Usage

```
s_group_by(.data, ...)
```

Arguments

.data	the data frame (or similar object)
...	string version of arguments to original dplyr function

s_modify *Modified version of modify that uses string arguments*

Description

Modified version of modify that uses string arguments

Usage

```
s_modify(.dt, ..., .envir = parent.frame())
```

Arguments

.dt	the data.table / similar object that shall be modified
...	string version of arguments of modify (see example)
.envir	environment in which arguments will be evaluated (relevant if variables outside .dt are used)

s_mutate *Modified version of dplyr's mutate that uses string arguments*

Description

Modified version of dplyr's mutate that uses string arguments

Usage

```
s_mutate(.data, ...)
```

Arguments

.data	the data frame (or similar object)
...	string version of arguments to original dplyr function

Examples

```
## Not run:
  library(dplyrExtras)

  d = mtcars
  cols = c("mpg","cyl","hp:vs")
  s_select(d,cols)
  # also works and yields identical result...
  cols = c("mpg","cyl, hp:vs")
  s_select(d,cols)
```

```
  s_filter(d,"gear == 3","cyl == 8")
  s_arrange(d, "-mpg, gear, carb")
  gd = s_group_by(d,"cyl")
  s_summarise(gd, "mean(disp), max(disp)")
  s_mutate_if(d, "cyl==6, new.col=100")


## End(Not run)
```

`s_mutate_if` *Modified version of mutate_if that uses string arguments*

Description

Modified version of mutate_if that uses string arguments

Usage

```
s_mutate_if(.data, ...)
```

Arguments

`.data`	the data frame (or similar object)
`...`	string version of arguments to mutate_if

Examples

```
## Not run:
  library(dplyrExtras)

  d = mtcars
  cols = c("mpg","cyl","hp:vs")
  s_select(d,cols)
  # also works and yields identical result...
  cols = c("mpg","cyl, hp:vs")
  s_select(d,cols)

  s_filter(d,"gear == 3","cyl == 8")
  s_arrange(d, "-mpg, gear, carb")
  gd = s_group_by(d,"cyl")
  s_summarise(gd, "mean(disp), max(disp)")
  s_mutate_if(d, "cyl==6, new.col=100")


## End(Not run)
```

s_select — *Modified version of dplyr's select that uses string arguments*

Description

Modified version of dplyr's select that uses string arguments

Usage

```
s_select(.data, ...)
```

Arguments

.data	the data frame (or similar object)
...	string version of arguments to original dplyr function

Examples

```
## Not run:
  library(dplyrExtras)

  d = mtcars
  cols = c("mpg","cyl","hp:vs")
  s_select(d,cols)
  # also works and yields identical result...
  cols = c("mpg","cyl, hp:vs")
  s_select(d,cols)

  s_filter(d,"gear == 3","cyl == 8")
  s_arrange(d, "-mpg, gear, carb")
  gd = s_group_by(d,"cyl")
  s_summarise(gd, "mean(disp), max(disp)")
  s_mutate_if(d, "cyl==6, new.col=100")


## End(Not run)
```

s_summarise — *Modified version of summarise that uses string arguments*

Description

Modified version of summarise that uses string arguments

Usage

```
s_summarise(.data, ...)
```

Arguments

.data	the data frame (or similar object)
...	string version of arguments to original dplyr function

Examples

```
## Not run:
  library(dplyrExtras)

  d = mtcars
  cols = c("mpg","cyl","hp:vs")
  s_select(d,cols)
  # also works and yields identical result...
  cols = c("mpg","cyl, hp:vs")
  s_select(d,cols)

  s_filter(d,"gear == 3","cyl == 8")
  s_arrange(d, "-mpg, gear, carb")
  gd = s_group_by(d,"cyl")
  s_summarise(gd, "mean(disp), max(disp)")
  s_mutate_if(d, "cyl==6, new.col=100")


## End(Not run)
```

---

xsummarise_each *extended version of summarise_each that allows alternative arrangement of outputs*

Description

extended version of summarise_each that allows alternative arrangement of outputs

Usage

```
xsummarise_each(tbl, funs, ..., .long = "funs", .wide = NA,
  .fun.var = ".fun", .var.var = ".var")
```

Arguments

tbl	the date frame
funs	a list of functions, see the help of summarize_each, best use the helper function funs() to create them
...	columns in tbl on which the functions should be applied
.long	either "funs" or "vars". Shall variables or functions be put in different rows?
.wide	currently ignored

Examples

```
## Not run:
  library(dplyr)
  library(dplyrExtras)
  by_species <- group_by(iris,Species)
  summarise_each(by_species,funs(min=min, max=max, mean=mean), Petal.Width, Sepal.Width)
 xsummarise_each(by_species,funs(min=min, max=max(.,na.rm=TRUE)), Petal.Width, Sepal.Width, .long="vars", .var.v
 xsummarise_each(by_species,funs(min=min, max=max), Petal.Width, Sepal.Width, .long="funs")

  xsummarise_each_(by_species,funs(min, max), vars=c("Petal.Width", "Sepal.Width"))
 xsummarise_each_(by_species,funs(min, max), vars=c("Petal.Width", "Sepal.Width"), .long="vars", .var.var = "Var

## End(Not run)
```

---

xsummarise_each_ *extended version of summarise_each_q that allows alternative arrangement of outputs*

Description

extended version of summarise_each_q that allows alternative arrangement of outputs

Usage

```
xsummarise_each_(tbl, funs, vars, .long = "funs", .wide = NA,
  fun.var = ".fun", var.var = ".var")
```

Arguments

tbl	the date frame
funs	a list of functions, see the help of summarize_each, best use the helper function funs() to create them
vars	columns in tbl on which the functions should be applied
.long	either "funs" or "vars". Shall variables or functions be put in different rows?
.wide	currently ignored

Index