Package 'm61r'

May 6, 2022

Type Package

Version 0.0.3

Title Package About Data Manipulation in Pure Base R

Description Data manipulation in one package and in base R.
Minimal. No dependencies.
'dplyr' and 'tidyr'-like in one place.
Nothing else than base R to build the package.

Depends R (>= 3.4.4)

License MIT + file LICENSE

URL https://github.com/pv71u98h1/m61r/

BugReports https://github.com/pv71u98h1/m61r/issues/

Encoding UTF-8

NeedsCompilation no

Author Jean-Marie Lepioufle [aut, cre]

Maintainer Jean-Marie Lepioufle <pv71u98h1@gmail.com>

Repository CRAN

Date/Publication 2022-05-06 15:50:02 UTC

R topics documented:

`arrange` *Arrange your data.frames*

Description

Re-arrange your data.frame in ascending or descending order and given one or several columns.

Usage

```
arrange_(df, ...)

desange_(df, ...)
```

Arguments

`df`	data.frame
`...`	formula used for arranging the data.frame

Value

The functions return an object of the same type as `df`. The output has the following properties: Properties:

- Columns are not modified.
- Output get rows in the order specified by
- ....
- Data frame attributes are preserved.

Examples

```
tmp <- arrange_(CO2,~c(conc))
head(tmp)

tmp <- arrange_(CO2,~c(Treatment,conc,uptake))
head(tmp)

tmp <- desange_(CO2,~c(Treatment,conc,uptake))
head(tmp)
```

expression *Formula to be run on a data.frame given a group*

Description

Evaluate a formula on the data.frame.

Usage

```
expression_(df, group=NULL, fun_expr)
```

Arguments

df	data.frame
group	formula that describes the group
fun_expr	formula that describes the expression to be run on the data.frame

Value

The function returns a list. Each element of the list get the result of processed expressions determined in ... on the whole data frame df if group is kept NULL, or for each group determined in group otherwise. The class of each element is intrinsic to the output of the expression determined in argument ....

Examples

```
expression_(CO2,fun_expr=~mean(conc))

expression_(CO2,fun_expr=~conc/uptake)

# with group
expression_(CO2,group=~Type,fun_expr=~mean(uptake))

expression_(CO2,group=~Type,fun_expr=~lm(uptake~conc))
```

filter *filter a data.frame*

Description

Filter rows of a data.frame with conditions.

Usage

```
filter_(df, subset = NULL)
```

Arguments

df	data.frame
subset	formula that describes the conditions

Value

The function returns an object of the same type as df. Properties:

- Columns are not modified.
- Only rows following the condtion determined by
- subset appear.
- Data frame attributes are preserved.

Examples

```
tmp <- filter_(CO2,~Plant=="Qn1")
head(tmp)

tmp <- filter_(CO2,~Type=="Quebec")
head(tmp)
```

---

group_by *group_by a data.frame by chosen columns*

Description

Group a data.frame by chosen columns

Usage

```
group_by_(df, group = NULL)
```

Arguments

df	data.frame
group	formula that describes the group

Value

The function returns a list. Each element of the list is a subset of data frame df. Subset is determined by variables given in group. Each data frame get the following properties:

- Columns are not modified.
- Only rows corresponding to the subset.
- Data frame attributes are preserved.

Examples

```
tmp <- group_by_(CO2,~c(Type,Treatment))

tmp[[1]]
```

`join` *Join two data.frames*

Description

Join two data.frames.

Usage

```
left_join_(df, df2, by = NULL, by.x = NULL, by.y = NULL)

anti_join_(df, df2, by = NULL, by.x = NULL, by.y = NULL)

full_join_(df, df2, by = NULL, by.x = NULL, by.y = NULL)

inner_join_(df, df2, by = NULL, by.x = NULL, by.y = NULL)

right_join_(df, df2, by = NULL, by.x = NULL, by.y = NULL)

semi_join_(df, df2, by = NULL, by.x = NULL, by.y = NULL)
```

Arguments

`df`	data.frame
`df2`	data.frame
`by`	column names of the pivot of both data.frame 1 and data.frame 2 if they are identical. Otherwise, better to use by.x and by.y
`by.x`	column names of the pivot of data.frame 1
`by.y`	column names of the pivot of data.frame 2

Value

The functions return a data frame. The output has the following properties:

- For functions `left_join()`, `inner_join()`, `full_join()`, and `right_join()`, output includes all `df1` columns and all `df2` columns. For columns with identical names in `df1` and `df2`, a suffix '.x' and '.y' is added. For `left_join()`, all `df1` rows with matching rows of `df2` For `inner_join()`, a subset of `df1` rows matching rows of `df2`. For `full_join()`, all `df1` rows, with all `df2` rows. For `right_join()`, all `df2` rows with matching rows of `df1`.
- For functions `semi_join()` and `anti_join()`, output include columns of `df1` only. For `semi_join()`, all `df1` rows with a match in `df2`. For `anti_join()`, a subset of `df1` rows not matching rows of `df2`.

Examples

```
books <- data.frame(
             name = I(c("Tukey", "Venables", "Tierney","Ripley",
                   "Ripley", "McNeil", "R Core")),
             title = c("Exploratory Data Analysis",
                   "Modern Applied Statistics ...",
                   "LISP-STAT",
                   "Spatial Statistics", "Stochastic Simulation",
                   "Interactive Data Analysis",
                   "An Introduction to R"),
              other.author = c(NA, "Ripley", NA, NA, NA, NA,"Venables & Smith"))

authors <- data.frame(
             surname = I(c("Tukey", "Venables", "Tierney", "Ripley", "McNeil","Asimov")),
               nationality = c("US", "Australia", "US", "UK", "Australia","US"),
               deceased = c("yes", rep("no", 4),"yes"))

tmp <- left_join_(books,authors, by.x = "name", by.y = "surname")
head(tmp)

tmp <- inner_join_(books,authors, by.x = "name", by.y = "surname")
head(tmp)

tmp <- full_join_(books,authors, by.x = "name", by.y = "surname")
head(tmp)

tmp <- right_join_(books,authors, by.x = "name", by.y = "surname")
head(tmp)

tmp <- semi_join_(books,authors, by.x = "name", by.y = "surname")
head(tmp)

tmp <- anti_join_(books,authors, by.x = "name", by.y = "surname")
head(tmp)
```

`m61r` *Create m61r object*

Description

Create a m61r object that enables to run a sequence of operations on a data.frame.

Usage

```
m61r(df = NULL)

## S3 method for class 'm61r'
x[i, j, ...]
```

```
## S3 replacement method for class 'm61r'
x[i, j] <- value

## S3 method for class 'm61r'
print(x, ...)

## S3 method for class 'm61r'
names(x, ...)

## S3 method for class 'm61r'
dim(x, ...)

## S3 method for class 'm61r'
as.data.frame(x, ...)

## S3 method for class 'm61r'
rbind(x, ...)

## S3 method for class 'm61r'
cbind(x, ...)
```

Arguments

df	data.frame
x	object of class m61r
i	row
j	column
...	further arguments passed to or from other methods
value	value to be assigned

Value

The function m61r returns an object of type m61r.

Argument df get stored internally to the object m61r. One manipulates the internal data.frame by using internal functions similar to the ones implemented in package m61r for data.frames as arrange, desange, filter, join and its relatives, mutate and transmutate, gather and spread, select, groupe_by, summarise, values and modify. The result of the last action is stored internally to the object m61r until the internal function values get called. It is thus possible to create a readable sequence of actions on a data.frame.

In addition,

- [.m61r returns a subset of the internal data.frame embedded to the object m61r.
- [<-.m61r assigns value to the internal data.frame embedded to the object m61r.
- print.m61r prints the internal data.frame embedded to the object m61r.

- names.m61r provides the names of the column of the internal data.frame embedded to the object m61r.
- dim.m61r provides the dimensions of the internal data.frame embedded to the object m61r.
- as.data.frame.m61r extracts the internal data.frame embedded to the object m61r.
- cbind.m61r combines by _c_olumns two objects m61r.
- rbind.m61r combines by _r_ows two objects m61r.
- left_join, anti_join, full_join, inner_join, right_join, semi_join join two objects m61r.

Finally, it is possible to clone a m61r object into a new one by using the internal function clone.

Examples

```
# init
co2 <- m61r(df=CO2)

# filter
co2$filter(~Plant=="Qn1")
co2

co2$filter(~Type=="Quebec")
co2

# select
co2$select(~Type)
co2

co2$select(~c(Plant,Type))
co2

co2$select(~-Type)
co2

co2$select(variable=~-(Plant:Treatment))
co2

# mutate/transmutate
co2$mutate(z=~conc/uptake)
co2

co2$mutate(mean=~mean(uptake))
co2

co2$mutate(z1=~uptake/conc,y=~conc/100)
co2

co2$transmutate(z2=~uptake/conc,y2=~conc/100)
co2

# summarise
```

```
co2$summarise(mean=~mean(uptake),sd=~sd(uptake))
co2

co2$group_by(~c(Type,Treatment))
co2$summarise(mean=~mean(uptake),sd=~sd(uptake))
co2

# arrange/dessange
co2$arrange(~c(conc))
co2

co2$arrange(~c(Treatment,conc,uptake))
co2

co2$desange(~c(Treatment,conc,uptake))
co2

# join
authors <- data.frame(
             surname = I(c("Tukey", "Venables", "Tierney", "Ripley", "McNeil")),
             nationality = c("US", "Australia", "US", "UK", "Australia"),
             deceased = c("yes", rep("no", 4)))

books <- data.frame(
           name = I(c("Tukey", "Venables", "Tierney","Ripley",
                 "Ripley", "McNeil", "R Core")),
          title = c("Exploratory Data Analysis",
                 "Modern Applied Statistics ...",
                 "LISP-STAT",
                 "Spatial Statistics", "Stochastic Simulation",
                 "Interactive Data Analysis",
                 "An Introduction to R"),
       other.author = c(NA, "Ripley", NA, NA, NA, NA,"Venables & Smith"))

## inner join
tmp <- m61r(df=authors)

tmp$inner_join(books, by.x = "surname", by.y = "name")
tmp

## left join
tmp$left_join(books, by.x = "surname", by.y = "name")
tmp

## right join
tmp$right_join(books, by.x = "surname", by.y = "name")
tmp

## full join
tmp$full_join(books, by.x = "surname", by.y = "name")
tmp

## semi join
```

```
tmp$semi_join(books, by.x = "surname", by.y = "name")
tmp

## anti join #1
tmp$anti_join(books, by.x = "surname", by.y = "name")
tmp

## anti join #2
tmp2 <- m61r(df=books)
tmp2$anti_join(authors, by.x = "name", by.y = "surname")
tmp2

## with two m61r objects
tmp1 <- m61r(books)
tmp2 <- m61r(authors)
tmp3 <- anti_join(tmp1,tmp2, by.x = "name", by.y = "surname")
tmp3

# Reshape

## gather
df3 <- data.frame(id = 1:4,
                  age = c(40,50,60,50),
                  dose.a1 = c(1,2,1,2),
                  dose.a2 = c(2,1,2,1),
                  dose.a14 = c(3,3,3,3))

df4 <- m61r::m61r(df3)
df4$gather(pivot = c("id","age"))
df4

## spread
df3 <- data.frame(id = 1:4,
                  age = c(40,50,60,50),
                  dose.a1 = c(1,2,1,2),
                  dose.a2 = c(2,1,2,1),
                  dose.a14 = c(3,3,3,3))

df4 <- m61r::gather_(df3,pivot = c("id","age"))
df4 <- rbind(df4,
  data.frame(id=5, age=20,parameters="dose.a14",values=8),
  data.frame(id=6, age=10,parameters="dose.a1",values=5))

tmp <- m61r::m61r(df4)
tmp$spread(col_name="parameters",col_values="values",pivot=c("id","age"))
tmp


# equivalence
co2           # is not equivalent to co2[]
co2[]         # is equivalent to co2$values()
co2[1,]       # is equivalent to co2$values(1,)
co2[,2:3]     # is equivalent to co2$values(,2:3)
```

```
co2[1:10,1:3] # is equivalent to co2$values(1:10,2:3)
co2[1,"Plant"]# is equivalent to co2$values(1,"Plant")

# modification on m61r object only stay for one step
co2[1,"conc"] <- 100
co2[1,] # temporary result
co2[1,] # back to normal

# WARNING:
# Keep the brackets to manipulate the intern data.frame
co2[] <- co2[-1,]
co2[1:3,] # temporary result
co2[1:3,] # back to normal

# ... OR you will destroy co2, and only keep the data.frame
# co2 <- co2[-1,]
# class(co2) # data.frame

# descriptive manipulation
names(co2)
dim(co2)
str(co2)

## cloning
# The following will only create a second variable that point on
# the same object (!= cloning)
foo <- co2
str(co2)
str(foo)

# Instead, cloning into a new environemnt
foo <- co2$clone()
str(co2)
str(foo)
```

`mutate` *Mutate and transmutate a data.frame*

Description

Mutate and transmutate a data.frame.

Usage

```
mutate_(df, ...)

transmutate_(df, ...)
```

Arguments

df	data.frame
...	formula used for mutating/transmutating the data.frame

Value

The functions return a data frame. The output has the following properties:

- For function mutate_(), output includes all df columns. In addition, new columns are created according to argument ... and placed after the others.
- For function transmutate_(), output includes only columns created according to argument ... and placed after the others.

Examples

```
tmp <- mutate_(CO2,z=~conc/uptake)
head(tmp)

# Return an warning: expression mean(uptake) get a result with 'nrow' different from 'df'
# tmp <- mutate_(CO2,mean=~mean(uptake))

tmp <- mutate_(CO2,z1=~uptake/conc,y=~conc/100)
head(tmp)

tmp <- transmutate_(CO2,z2=~uptake/conc,y2=~conc/100)
head(tmp)
```

---

reshape *Reshape a data.frame*

Description

Reshape a data.frame.

Usage

```
gather_(df, new_col_name = "parameters", new_col_values = "values",
  pivot)

spread_(df, col_name, col_values, pivot)
```

Arguments

df	data.frame
new_col_name	name of the new column 'parameters'
new_col_values	name of the new columns 'values'
col_name	name of the column 'parameters'
col_values	name of the new columns 'values'
pivot	name of the columns used as pivot

Details

A data frame is said 'wide' if several of its columns describe connected information of the same record. A data frame is said 'long' if two of its columns provide information about records, with one describing their name and the second their value. Functions `gather_()` and `spread_()` enable to reshape a data frames from a 'wide' format to a 'long' format, and vice-versa.

Value

The functions return a data frame.

- Output from function `gather_()` get 'pivot' columns determined by argument `pivot`, and 'long' columns named according to arguments `new_col_name` and `new_col_values`.
- Output from function `spread_()` get 'pivot' columns determined by argument `pivot`, and 'wide' columns named according to values in column determined by argument `col_name`. For 'wide' columns, each row corresponds to values present in column determined by argument `col_values`.

Examples

```
df3 <- data.frame(id = 1:4,
                    age = c(40,50,60,50),
                    dose.a1 = c(1,2,1,2),
                    dose.a2 = c(2,1,2,1),
                    dose.a14 = c(3,3,3,3))

gather_(df3,pivot = c("id","age"))

df4 <- gather_(df3,pivot = c("id","age"))
df5 <- rbind(df4,
    data.frame(id=5, age=20,parameters="dose.a14",values=8),
    data.frame(id=6, age=10,parameters="dose.a1",values=5))

spread_(df5,col_name="parameters",col_values="values",pivot=c("id","age"))
```

`select` *select columns of a data.frame*

Description

Select columns of a data.frame.

Usage

```
select_(df, variable = NULL)
```

Arguments

`df`	data.frame
`variable`	formula that describes the selection

Value

`select_()` returns a data frame. Properties:

- Only columns following the condtion determined by
- variable appear.
- Rows are not modified.

Examples

```
tmp <- select_(CO2,~Type)
head(tmp)

tmp <- select_(CO2,~c(Plant,Type))
head(tmp)

tmp <- select_(CO2,~-Type)
head(tmp)

tmp <- select_(CO2,variable=~-(Plant:Treatment))
head(tmp)
```

summarise *Summarise formula on groups*

Description

Summarise of formulas on a data.frame.

Usage

```
summarise_(df, group = NULL, ...)
```

Arguments

df	data.frame
group	formula that describes the group
...	formulas to be generated

Value

summarise_() returns a data frame. If argument group is not NULL, output get its first columns called according to the names present in argument group. The following columns are called according to the name of each argument present in .... Each row corresponds to processed expressions determined in ... for each group determined in group, or over the whole data frame if group is NULL.

Examples

```
summarise_(CO2,a=~mean(uptake),b=~sd(uptake))

summarise_(CO2, group=~c(Type,Treatment),a=~mean(uptake),b=~sd(uptake))
```

value *get or assign a value to a data.frame*

Description

Get or assign a value to a data.frame

Usage

```
value_(df, i, j)

'modify_<-'(df,i,j,value)
```

Arguments

df	data.frame
i	row
j	column
value	value to be assigned

Value

The functions value_ and 'modify_<-' return a data frame. Properties:

- Only rows determined by
- i appear. If
- i is missing, no row is filtered.
- Only columns determined by
- j appear. If
- j is missing, no column is filtered.

Besides,

- For function value_: If argument i is non-missing and argument j is missing, the function returns an object of the same type as df. If both arguments i and j are missing, the function returns an object of the same type as df.
- For function 'modify_<-': The function returns an object of the same type as df.

Examples

```
tmp <- value_(CO2,1,2)
attributes(tmp) # data frame

tmp <- value_(CO2,1:2,2)
attributes(tmp) # data frame

tmp <- value_(CO2,1:2,2:4)
attributes(tmp) # data frame

tmp <- value_(CO2,,2)
attributes(tmp) # data frame

tmp <- value_(CO2,2)
attributes(tmp) # same as CO2

tmp <- value_(CO2)
attributes(tmp) # same as CO2

df3 <- data.frame(id = 1:4,
                      age = c(40,50,60,50),
                      dose.a1 = c(1,2,1,2),
                      dose.a2 = c(2,1,2,1),
```

```
                        dose.a14 = c(3,3,3,3))

  'modify_<-'(df3,1,2,6)

  'modify_<-'(df3,1:3,2:4,data.frame(c(20,10,90),c(9,3,4),c(0,0,0)))
```

Index