Package 'edm1.df'

July 26, 2024

Title Set of functions for dataframe manipulation

Version 2.0.0.0

Description Provides multiple functions to manipulate data in dataframe according to different algorythms for different goals.

License GPL (==3)

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.1

Imports stringr,
stringi,
dplyr,
openxlsx

Contents

colins_datf *colins_datf*

Description

Allow to insert vectors into a dataframe.

Usage

```
colins_datf(inpt_datf, target_col = list(), target_pos = list())
```

Arguments

inpt_datf	is the dataframe where vectors will be inserted
target_col	is a list containing all the vectors to be inserted
target_pos	is a list containing the vectors made of the columns names or numbers where the associated vectors from target_col will be inserted after

Examples

```
datf1 <- data.frame("frst_col"=c(1:5), "scd_col"=c(5:1))

print(colins_datf(inpt_datf=datf1, target_col=list(c("oui", "oui", "oui", "non", "non"),
             c("u", "z", "z", "z", "u")),
                target_pos=list(c("frst_col", "scd_col"), c("scd_col"))))

#  frst_col cur_col scd_col cur_col.1 cur_col
#1        1     oui       5       oui       u
#2        2     oui       4       oui       z
#3        3     oui       3       oui       z
#4        4     non       2       non       z
#5        5     non       1       non       u

print(colins_datf(inpt_datf=datf1, target_col=list(c("oui", "oui", "oui", "non", "non"),
             c("u", "z", "z", "z", "u")),
                target_pos=list(c(1, 2), c("frst_col"))))

#  frst_col cur_col scd_col cur_col cur_col
#1        1     oui       5       u     oui
#2        2     oui       4       z     oui
#3        3     oui       3       z     oui
#4        4     non       2       z     non
#5        5     non       1       u     non
```

cumulated_rows *cumulated_rows*

Description

Output a vector of size that equals to the rows number of the input dataframe, with TRUE value at the indices corresponding to the row where at least a cell of any column is equal to one of the values inputed in `values_v`

Usage

```
cumulated_rows(inpt_datf, values_v = c())
```

Arguments

`inpt_datf`	is the input data.frame
`values_v`	is a vector containing all the values that a cell has to equal to return a TRUE value in the output vector at the index corresponding to the row of the cell

Examples

```
datf_teste <- data.frame(c(1:10), c(10:1))

print(datf_teste)
```

```
   c.1.10. c.10.1.
1         1      10
2         2       9
3         3       8
4         4       7
5         5       6
6         6       5
7         7       4
8         8       3
9         9       2
10       10       1
```

```
print(cumulated_rows(inpt_datf = datf_teste, values_v = c(2, 3)))
```

```
[1]  FALSE TRUE TRUE  FALSE  FALSE  FALSE  FALSE TRUE TRUE  FALSE
```

cumulated_rows_na *cumulated_rows_na*

Description

Output a vector of size that equals to the rows number of the input dataframe, with TRUE value at the indices corresponding to the row where at least a cell of any column is equal to NA.

Usage

```
cumulated_rows_na(inpt_datf)
```

Arguments

inpt_datf	is the input data.frame

Examples

```
datf_teste <- data.frame(c(1, 2, 3, 4, 5, NA, 7), c(10, 9, 8, NA, 7, 6, NA))

print(datf_teste)
```

```
  c.1..2..3..4..5..NA..7. c.10..9..8..NA..7..6..NA.
1                       1                        10
2                       2                         9
3                       3                         8
4                       4                        NA
5                       5                         7
6                      NA                         6
7                       7                        NA
```

```
print(cumulated_rows_na(inpt_datf = datf_teste))
```

```
[1] FALSE FALSE FALSE  TRUE FALSE  TRUE  TRUE
```

---

cut_v *cut_v*

Description

Allow to convert a vector to a dataframe according to a separator.

Usage

```
cut_v(inpt_v, sep_ = "")
```

Arguments

inpt_v	is the input vector
sep_	is the separator of the elements in inpt_v, defaults to ""

Examples

```
print(cut_v(inpt_v=c("oui", "non", "oui", "non")))
```

```
#    X.o. X.u. X.i.
#oui "o"  "u"  "i"
#non "n"  "o"  "n"
#oui "o"  "u"  "i"
#non "n"  "o"  "n"
```

```
print(cut_v(inpt_v=c("ou-i", "n-on", "ou-i", "n-on"), sep_="-"))

#     X.ou. X.i.
#ou-i "ou"  "i"
#n-on "n"   "on"
#ou-i "ou"  "i"
#n-on "n"   "on"
```

`diff_datf` *diff_datf*

Description

Returns a vector with the coordinates of the cell that are not equal between 2 dataframes (row, column).

Usage

```
diff_datf(datf1, datf2)
```

Arguments

`datf1`	is an an input dataframe
`datf2`	is an an input dataframe

Examples

```
datf1 <- data.frame(c(1:6), c("oui", "oui", "oui", "oui", "oui", "oui"), c(6:1))

datf2 <- data.frame(c(1:7), c("oui", "oui", "oui", "oui", "non", "oui", "zz"))

print(diff_datf(datf1=datf1, datf2=datf2))

#[1] 5 1 5 2
```

`groupr_datf` *groupr_datf*

Description

Allow to create groups from a dataframe. Indeed, you can create conditions that lead to a flag value for each cell of the input dataframeaccording to the cell value. This function is based on see_datf and nestr_datf2 functions.

Usage

```
groupr_datf(
  inpt_datf,
  condition_lst,
  val_lst,
  conjunction_lst,
  rtn_val_pos = c()
)
```

Arguments

inpt_datf	is the input dataframe
condition_lst	is a list containing all the condition as a vector for each group
val_lst	is a list containing all the values associated with condition_lst as a vector for each group
conjunction_lst	is a list containing all the conjunctions associated with condition_lst and val_lst as a vector for each group
rtn_val_pos	is a vector containing all the group flag value like this ex: c("flag1", "flag2", "flag3")

Examples

```
interactive()

datf1 <- data.frame(c(1, 2, 1), c(45, 22, 88), c(44, 88, 33))

val_lst <- list(list(c(1), c(1)), list(c(2)), list(c(44, 88)))

condition_lst <- list(c(">", "<"), c("%%"), c("==", "=="))

conjunction_lst <- list(c("|"), c(), c("|"))

rtn_val_pos <- c("+", "++", "+++")

print(groupr_datf(inpt_datf=datf1, val_lst=val_lst, condition_lst=condition_lst,
conjunction_lst=conjunction_lst, rtn_val_pos=rtn_val_pos))

#    X1  X2  X3
#1 <NA>   + +++
#2   ++  ++ +++
#3 <NA> +++   +
```

---

id_keepr *id_keepr_datf*

Description

Allow to get the original indexes after multiple equality comparaison according to the original number of row

Usage

```
id_keepr(inpt_datf, col_v = c(), el_v = c(), rstr_l = NA)
```

Arguments

inpt_datf	is the input dataframe
col_v	is the vector containing the column numbers or names to be compared to their respective elements in "el_v"
el_v	is a vector containing the elements that may be contained in their respective column described in "col_v"
rstr_l	is a list containing the vector composed of the indexes of the elements chosen for each comparison. If the length of the list is inferior to the lenght of comparisons, so the last vector of rstr_l will be the same as the last one to fill make rstr_l equal in term of length to col_v and el_v

Examples

```
datf1 <- data.frame(c("oui", "oui", "oui", "non", "oui"),
     c("opui", "op", "op", "zez", "zez"), c(5:1), c(1:5))

print(id_keepr(inpt_datf=datf1, col_v=c(1, 2), el_v=c("oui", "op")))

#[1] 2 3

print(id_keepr(inpt_datf=datf1, col_v=c(1, 2), el_v=c("oui", "op"),
     rstr_l=list(c(1:5), c(3, 2, 2, 2, 3))))

#[1] 2 3

print(id_keepr(inpt_datf=datf1, col_v=c(1, 2), el_v=c("oui", "op"),
     rstr_l=list(c(1:5), c(3))))

#[1] 3

print(id_keepr(inpt_datf=datf1, col_v=c(1, 2), el_v=c("oui", "op"), rstr_l=list(c(1:5))))

#[1] 2 3
```

---

insert_datf *edm1 insert_datf*

Description

Allow to insert dataframe into another dataframe according to coordinates (row, column) from the dataframe that will be inserted

Usage

```
insert_datf(datf_in, datf_ins, ins_loc)
```

Arguments

datf_in	is the dataframe that will be inserted
datf_ins	is the dataset to be inserted
ins_loc	is a vector containg two parameters (row, column) of the begining for the insertion

Examples

```
datf1 <- data.frame(c(1, 4), c(5, 3))

datf2 <- data.frame(c(1, 3, 5, 6), c(1:4), c(5, 4, 5, "ereer"))

print(insert_datf(datf_in=datf2, datf_ins=datf1, ins_loc=c(4, 2)))

#   c.1..3..5..6. c.1.4. c.5..4..5...ereer..
# 1             1      1                   5
# 2             3      2                   4
# 3             5      3                   5
# 4             6      1                   5

print(insert_datf(datf_in=datf2, datf_ins=datf1, ins_loc=c(3, 2)))

#   c.1..3..5..6. c.1.4. c.5..4..5...ereer..
# 1             1      1                   5
# 2             3      2                   4
# 3             5      1                   5
# 4             6      4                   3

print(insert_datf(datf_in=datf2, datf_ins=datf1, ins_loc=c(2, 2)))

#   c.1..3..5..6. c.1.4. c.5..4..5...ereer..
# 1             1      1                   5
# 2             3      1                   5
# 3             5      4                   3
# 4             6      4               ereer
```

---

intersect_mod *intersect_mod*

Description

Returns the mods that have elements in common

Usage

```
intersect_mod(datf, inter_col, mod_col, n_min, descendly_ordered = NA)
```

Arguments

datf	is the input dataframe
inter_col	is the column name or the column number of the values that may be commun betwee the different mods

`mod_col` is the column name or the column number of the mods in the dataframe

`n_min` is the minimum elements in common a mod should have to be taken in count

`ordered_descendly`
in case that the elements in commun are numeric, this option can be enabled by giving a value of TRUE or FALSE see examples

Examples

```
datf <- data.frame("col1"=c("oui", "oui", "oui", "oui", "oui", "oui",
                    "non", "non", "non", "non", "ee", "ee", "ee"), "col2"=c(1:6, 2:5, 1:

print(intersect_mod(datf=datf, inter_col=2, mod_col=1, n_min=2))

   col1 col2
2   oui    2
3   oui    3
7   non    2
8   non    3
12   ee    2
13   ee    3

print(intersect_mod(datf=datf, inter_col=2, mod_col=1, n_min=3))

   col1 col2
2   oui    2
3   oui    3
4   oui    4
5   oui    5
7   non    2
8   non    3
9   non    4
10  non    5

print(intersect_mod(datf=datf, inter_col=2, mod_col=1, n_min=5))

  col1 col2
1  oui    1
2  oui    2
3  oui    3
4  oui    4
5  oui    5
6  oui    6

datf <- data.frame("col1"=c("non", "non", "oui", "oui", "oui", "oui",
                     "non", "non", "non", "non", "ee", "ee", "ee"), "col2"=c(1:6, 2:5, 1

print(intersect_mod(datf=datf, inter_col=2, mod_col=1, n_min=3))

   col1 col2
8   non    3
9   non    4
10  non    5
3   oui    3
4   oui    4
5   oui    5
```

nestr_datf1 *nestr_datf1*

Description

Allow to write a value (1a) to a dataframe (1b) to its cells that have the same coordinates (row and column) than the cells whose value is equal to a another special value (2a), from another another dataframe (2b). The value (1a) depends of the cell value coordinates of the third dataframe (3b). If a cell coordinates (1c) of the first dataframe (1b) does not correspond to the coordinates of a good returning cell value (2a) from the dataframe (2b), so this cell (1c) can have its value changed to the same cell coordinates value (3a) of a third dataframe (4b), if (4b) is not set to NA.

Usage

```
nestr_datf1(
  inptf_datf,
  inptt_pos_datf,
  nestr_datf,
  yes_val = TRUE,
  inptt_neg_datf = NA
)
```

Arguments

inptf_datf	is the input dataframe (1b)
inptt_pos_datf	is the dataframe (2b) that corresponds to the (1a) values
nestr_datf	is the dataframe (2b) that has the special value (2a)
yes_val	is the special value (2a)
inptt_neg_datf	is the dataframe (4b) that has the (3a) values, defaults to NA

Examples

```
print(nestr_datf1(inptf_datf=data.frame(c(1, 2, 1), c(1, 5, 7)),
inptt_pos_datf=data.frame(c(4, 4, 3), c(2, 1, 2)),
inptt_neg_datf=data.frame(c(44, 44, 33), c(12, 12, 12)),
nestr_datf=data.frame(c(TRUE, FALSE, TRUE), c(FALSE, FALSE, TRUE)), yes_val=TRUE))

#  c.1..2..1. c.1..5..7.
#1          4         12
#2         44         12
#3          3          2

print(nestr_datf1(inptf_datf=data.frame(c(1, 2, 1), c(1, 5, 7)),
inptt_pos_datf=data.frame(c(4, 4, 3), c(2, 1, 2)),
inptt_neg_datf=NA,
nestr_datf=data.frame(c(TRUE, FALSE, TRUE), c(FALSE, FALSE, TRUE)), yes_val=TRUE))

#   c.1..2..1. c.1..5..7.
#1          4          1
#2          2          5
```

```
#3          3          2
```

`nestr_datf2` *nestr_datf2*

Description

Allow to write a special value (1a) in the cells of a dataframe (1b) that correspond (row and column) to whose of another dataframe (2b) that return another special value (2a). The cells whose coordinates do not match the coordinates of the dataframe (2b), another special value can be written (3a) if not set to NA.

Usage

```
nestr_datf2(inptf_datf, rtn_pos, rtn_neg = NA, nestr_datf, yes_val = T)
```

Arguments

`inptf_datf`	is the input dataframe (1b)
`rtn_pos`	is the special value (1a)
`rtn_neg`	is the special value (3a)
`nestr_datf`	is the dataframe (2b)
`yes_val`	is the special value (2a)

Examples

```
print(nestr_datf2(inptf_datf=data.frame(c(1, 2, 1), c(1, 5, 7)), rtn_pos="yes",
rtn_neg="no", nestr_datf=data.frame(c(TRUE, FALSE, TRUE), c(FALSE, FALSE, TRUE)), yes_val

#  c.1..2..1. c.1..5..7.
#1        yes         no
#2         no         no
#3        yes        yes
```

`paste_datf` *paste_datf*

Description

Return a vector composed of pasted elements from the input dataframe at the same index.

Usage

```
paste_datf(inpt_datf, sep = "")
```

Arguments

inpt_datf	is the input dataframe
sep	is the separator between pasted elements, defaults to ""

Examples

```
print(paste_datf(inpt_datf=data.frame(c(1, 2, 1), c(33, 22, 55))))

#[1] "133" "222" "155"
```

---

see_datf *see_datf*

Description

Allow to return a dataframe with special value cells (ex: TRUE) where the condition entered are respected and another special value cell (ex: FALSE) where these are not

Usage

```
see_datf(
  datf,
  condition_l,
  val_l,
  conjunction_l = c(),
  rt_val = TRUE,
  f_val = FALSE
)
```

Arguments

datf	is the input dataframe
condition_l	is the vector of the possible conditions ("==", ">", "<", "!=", "%%", "reg", "not_reg", "sup_nchar", "inf_nchar", "nchar") (equal to some elements in a vector, greater than, lower than, not equal to, is divisible by, the regex condition returns TRUE, the regex condition returns FALSE, the length of the elements is strictly superior to X, the length of the element is strictly inferior to X, the length of the element is equal to one element in a vector), you can put the same condition n times.
val_l	is the list of vectors containing the values or vector of values related to condition_l (so the vector of values has to be placed in the same order)
conjunction_l	contains the and or conjunctions, so if the length of condition_l is equal to 3, there will be 2 conjunctions. If the length of conjunction_l is inferior to the length of condition_l minus 1, conjunction_l will match its goal length value with its last argument as the last arguments. For example, c("&", "|", "&") with a goal length value of 5 –> c("&", "|", "&", "&", "&")
rt_val	is a special value cell returned when the conditions are respected
f_val	is a special value cell returned when the conditions are not respected

Details

This function will return an error if number only comparative conditions are given in addition to having character values in the input dataframe.

Examples

```
datf1 <- data.frame(c(1, 2, 4), c("a", "a", "zu"))

print(see_datf(datf=datf1, condition_l=c("nchar"), val_l=list(c(1))))

#    X1    X2
#1 TRUE  TRUE
#2 TRUE  TRUE
#3 TRUE FALSE

print(see_datf(datf=datf1, condition_l=c("=="), val_l=list(c("a", 1))))

#    X1    X2
#1  TRUE  TRUE
#2 FALSE  TRUE
#3 FALSE FALSE


print(see_datf(datf=datf1, condition_l=c("nchar"), val_l=list(c(1, 2))))

#    X1   X2
#1 TRUE TRUE
#2 TRUE TRUE
#3 TRUE TRUE

print(see_datf(datf=datf1, condition_l=c("not_reg"), val_l=list("[a-z]")))

#    X1    X2
#1 TRUE FALSE
#2 TRUE FALSE
#3 TRUE FALSE
```

`swipr` *swipr*

Description

Returns an ordered dataframes according to the elements order given. The input datafram has two columns, one with the ids which can be bonded to multiple elements in the other column.

Usage

```
swipr(inpt_datf, how_to = c(), id_w = 2, id_ids = 1)
```

Arguments

inpt_datf	is the input dataframe
how_to	is a vector containing the elements in the order wanted
id_w	is the column number or the column name of the elements
id_ids	is the column number or the column name of the ids

Examples

```
datf <- data.frame("col1"=c("Af", "Al", "Al", "Al", "Arg", "Arg", "Arg", "Arm", "Arm", "A

        "col2"=c("B", "B", "G", "S", "B", "S", "G", "B", "G", "B"))

print(swipr(inpt_datf=datf, how_to=c("G", "S", "B")))

   col1 col2
1    Af    B
2    Al    G
3    Al    S
4    Al    B
5   Arg    G
6   Arg    S
7   Arg    B
8   Arm    G
9   Arm    B
10   Al    B
```

---

unique_datf *unique_datf*

Description

Returns the input dataframe with the unique columns or rows.

Usage

```
unique_datf(inpt_datf, col = FALSE)
```

Arguments

inpt_datf	is the input dataframe
col	is a parameter that specifies if the dataframe returned should have unique columns or rows, defaults to F, so the dataframe returned by default has unique rows

Examples

```
datf1 <- data.frame(c(1, 2, 1, 3), c("a", "z", "a", "p"))

print(datf1)

  c.1..2..1..3. c..a....z....a....p.. c.1..2..1..3..1
1             1                     a               1
```

```
2             2                     z              2
3             1                     a              1
4             3                     p              3

print(unique_datf(inpt_datf=datf1))

#   c.1..2..1..3. c..a....z....a....p..
#1             1                     a
#2             2                     z
#4             3                     p

datf1 <- data.frame(c(1, 2, 1, 3), c("a", "z", "a", "p"), c(1, 2, 1, 3))

print(datf1)

  c.1..2..1..3. c..a....z....a....p..
1             1                     a
2             2                     z
3             1                     a
4             3                     p

print(unique_datf(inpt_datf=datf1, col=TRUE))

#  cur_v cur_v
#1     1     a
#2     2     z
#3     1     a
#4     3     p
```

`val_replacer` *val_replacer*

Description

Allow to replace value from dataframe to another one.

Usage

```
val_replacer(datf, val_replaced, val_replacor = TRUE)
```

Arguments

`datf`	is the input dataframe
`val_replaced`	is a vector of the value(s) to be replaced
`val_replacor`	is the value that will replace val_replaced

Examples

```
print(val_replacer(datf=data.frame(c(1, "oo4", TRUE, FALSE), c(TRUE, FALSE, TRUE, TRUE)),
     val_replaced=c(TRUE), val_replacor="NA"))

#  c.1...oo4...T..F. c.T..F..T..T.
```

```
#1                1            NA
#2              oo4         FALSE
#3               NA            NA
#4            FALSE            NA
```

`vec_in_datf` *vec_in_datf*

Description

Allow to get if a vector is in a dataframe. Returns the row and column of the vector in the dataframe if the vector is contained in the dataframe.

Usage

```
vec_in_datf(
  inpt_datf,
  inpt_vec = c(),
  coeff = 0,
  stop_untl = 1,
  conventional = FALSE
)
```

Arguments

`inpt_datf`	is the input dataframe
`inpt_vec`	is the vector that may be in the input dataframe
`coeff`	is the "slope coefficient" of inpt_vec
`stop_untl`	is the maximum number of the input vector the function returns, if in the dataframe
`conventional`	is if a positive slope coefficient means that the vector goes upward or downward

Examples

```
datf1 <- data.frame(c(1:5), c(5:1), c("a", "z", "z", "z", "a"))

print(datf1)

#  c.1.5. c.5.1. c..a....z....z....z....a..
#1      1      5                          a
#2      2      4                          z
#3      3      3                          z
#4      4      2                          z
#5      5      1                          a

print(vec_in_datf(inpt_datf=datf1, inpt_vec=c(5, 4, "z"), coeff=1))

#NULL

print(vec_in_datf(inpt_datf=datf1, inpt_vec=c(5, 2, "z"), coeff=1))

#[1] 5 1
```

```
print(vec_in_datf(inpt_datf=datf1, inpt_vec=c(3, "z"), coeff=1))

#[1] 3 2

print(vec_in_datf(inpt_datf=datf1, inpt_vec=c(4, "z"), coeff=-1))

#[1] 2 2

print(vec_in_datf(inpt_datf=datf1, inpt_vec=c(2, 3, "z"), coeff=-1))

#[1] 2 1

print(vec_in_datf(inpt_datf=datf1, inpt_vec=c(5, 2, "z"), coeff=-1, conventional=TRUE))

#[1] 5 1

datf1[4, 2] <- 1

print(vec_in_datf(inpt_datf=datf1, inpt_vec=c(1, "z"), coeff=-1, conventional=TRUE, stop_

#[1] 4 2 5 2
```

`vlookup_datf` *vlookup_datf*

Description

Alow to perform a vlookup on a dataframe

Usage

```
vlookup_datf(datf, v_id, col_id = 1, included_col_id = "yes")
```

Arguments

`datf`	is the input dataframe
`v_id`	is a vector containing the ids
`col_id`	is the column that contains the ids (default is equal to 1)
`included_col_id`	is if the result should return the col_id (default set to yes)

Examples

```
datf1 <- data.frame(c("az1", "az3", "az4", "az2"), c(1:4), c(4:1))

print(vlookup_datf(datf=datf1, v_id=c("az1", "az2", "az3", "az4")))

#   c..az1....az3....az4....az2.. c.1.4. c.4.1.
#2                             az1      1      4
#4                             az2      4      1
#21                            az3      2      3
#3                             az4      3      2
```

wider_datf *wider_datf*

Description

Takes a dataframe as an input and the column to split according to a seprator.

Usage

```
wider_datf(inpt_datf, col_to_splt = c(), sep_ = "-")
```

Arguments

inpt_datf	is the input dataframe
col_to_splt	is a vector containing the number or the colnames of the columns to split according to a separator
sep_	is the separator of the elements to split to new columns in the input dataframe

Examples

```
datf1 <- data.frame(c(1:5), c("o-y", "hj-yy", "er-y", "k-ll", "ooo-mm"), c(5:1))

datf2 <- data.frame("col1"=c(1:5), "col2"=c("o-y", "hj-yy", "er-y", "k-ll", "ooo-mm"))

print(wider_datf(inpt_datf=datf1, col_to_splt=c(2), sep_="-"))

#       pre_datf X.o.  X.y.
#o-y    1        "o"   "y"  5
#hj-yy  2        "hj"  "yy" 4
#er-y   3        "er"  "y"  3
#k-ll   4        "k"   "ll" 2
#ooo-mm 5        "ooo" "mm" 1

print(wider_datf(inpt_datf=datf2, col_to_splt=c("col2"), sep_="-"))

#       pre_datf X.o.  X.y.
#o-y    1        "o"   "y"
#hj-yy  2        "hj"  "yy"
#er-y   3        "er"  "y"
#k-ll   4        "k"   "ll"
#ooo-mm 5        "ooo" "mm"
```

Index