

Package ‘edm1’

February 19, 2024

Title edm

Version 1.0

Author person('Julien', 'Larget-Piet', role = c('aut', 'cre'))

Description What the package does (one paragraph).

License GPL-2

description Set of tools to manage mostly dataframe and character.

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.2.3

Imports openxlsx, stringr, stringi,

NeedsCompilation no

Maintainer Julien Larget-Piet <julien.larget-piet@edu.univ-eiffel.fr>

R topics documented:

all_stat	3
any_join_df	3
append_row	5
appndr	6
calc_occu_v	6
can_be_num	7
change_date	7
chr_removr	8
closer_ptrn	8
closer_ptrn_adv	11
closest_date	12
clusterizer_v	12
cost_and_taxes	14
data_gen	15
data_meshup	16
date_sort	16
days_from_month	17
df_tuned	17
diff_xlsx	18
equalizer_v	19
extrm_dates	19

extrt_only_v	20
file_rec	20
file_rec2	21
fillr	21
fittr_v	22
fixer_nest_v	22
format_date	23
geo_min	23
get_rec	24
globe	24
groupr_df	25
incr_fillr	26
insert_df	26
inter_max	27
inter_min	27
letter_to_nb	28
list_files	28
lst_flatnr	29
match_n	29
match_n2	30
multitud	30
nb_to_letter	31
nestr_df1	31
nestr_df2	32
nest_v	32
occu	33
paste_df	33
pattern_generator	34
pattern_gettr	34
pattern_tuning	35
ptrn_switchr	36
ptrn_twkr	36
rearangr_v	37
see_df	38
see_file	38
see_idx	39
see_inside	39
unique_pos	40
until_stnl	40
val_replacer	41
vec_in_df	41
vlookup_df	42
v_to_df	42

all_stat

*all_stat***Description**

Allow to see all the main statistics indicators (mean, median, variance, standard deviation, sum, max, min, quantile) of variables in a dataframe by the modality of a variable in a column of the input datarame. In addition to that, you can get the occurence of other qualitative variables by your chosen qualitative variable, you have just to precise it in the vector "stat_var" where all the statistics indicators are given with "occu-var_you_want/".

Usage

```
all_stat(inpt_v, var_add = c(), stat_var = c(), inpt_df)
```

Arguments

inpt_v	is the modalities of the variables
var_add	is the variables you want to get the stats from
stat_var	is the stats indicators you want
inpt_df	is the input dataframe

Examples

```
df <- data.frame("mod"=c("first", "seco", "seco", "first", "first", "third", "first"),
                 "var1"=c(11, 22, 21, 22, 22, 11, 9),
                 "var2"=c("d", "d", "z", "z", "z", "d", "z"),
                 "var3"=c(45, 44, 43, 46, 45, 45, 42),
                 "var4"=c("A", "A", "A", "A", "B", "C", "C"))

all_stat(inpt_v=c("first", "seco"), var_add = c("var1", "var2", "var3", "var4"),
        stat_var=c("sum", "mean", "median", "sd", "occu-var2/", "occu-var4/", "variance", "quantile"),
        inpt_df=df)
```

any_join_df

*any_join_df***Description**

Allow to perform SQL joints with more features

Usage

```
any_join_df(
  inpt_df_l,
  join_type = "inner",
  join_spe = NA,
  id_v = c(),
  excl_col = c(),
  rtn_col = c(),
  d_val = NA
)
```

Arguments

inpt_df_l	is a list containing all the dataframe
join_type	is the joint type. Defaults to inner but can be changed to a vector containing all the dataframes you want to take their ids to don external joints.
join_spe	can be equal to a vector to do an external joints on all the dataframes. In this case, join_type should not be equal to "inner"
id_v	is a vector containing all the ids name of the dataframes. The ids names can be changed to number of their columns taking in count their position in inpt_df_l. It means that if my id is in the third column of the second dataframe and the first dataframe have 5 columns, the column number of the ids is $5 + 3 = 8$
excl_col	is a vector containing the column names to exclude, if this vector is filled so "rtn_col" should not be filled. You can also put the column number in the manner indicated for "id_v". Defaults to c()
rtn_col	is a vector containing the column names to retain, if this vector is filled so "excl_col" should not be filled. You can also put the column number in the manner indicated for "id_v". Defaults to c()
d_val	is the default val when here is no match

Examples

```
df1 <- data.frame("val"=c(1, 1, 2, 4), "ids"=c("e", "a", "z", "a"),
  "last"=c("oui", "oui", "non", "oui"),
  "second_ids"=c(13, 11, 12, 8))

df2 <- data.frame("val"=c(3, 7, 2, 4, 1, 2), "ids"=c("a", "z", "z", "a", "a", "a"),
  "bool"=c(T, F, F, T, T, T),
  "second_ids"=c(13, 12, 8, 34, 22, 12))

df3 <- data.frame("val"=c(1, 9, 2, 4), "ids"=c("a", "a", "z", "a"),
  "last"=c("oui", "oui", "non", "oui"),
  "second_ids"=c(13, 11, 12, 8))

print(any_join_df(inpt_df_l=list(df1, df2, df3), join_type="inner",
  id_v=c("ids", "second_ids"),
  excl_col=c(), rtn_col=c()))
  ids val ids last second_ids val ids bool second_ids val ids last second_ids
3 z12  2  z non          12  7  z FALSE          12  2  z non          12

print(any_join_df(inpt_df_l=list(df1, df2, df3), join_type="inner", id_v=c("ids"),
  excl_col=c(), rtn_col=c()))

  ids val ids last second_ids val ids bool second_ids val ids last second_ids
2  a  1  a oui          11  3  a TRUE          13  1  a oui          13
3  z  2  z non          12  7  z FALSE          12  2  z non          12
4  a  4  a oui           8  4  a FALSE          34  9  a oui          11

print(any_join_df(inpt_df_l=list(df1, df2, df3), join_type=c(1), id_v=c("ids"),
  excl_col=c(), rtn_col=c()))

  ids val ids last second_ids val ids bool second_ids val ids last
1  e  1  e oui          13 <NA> <NA> <NA>          <NA> <NA> <NA> <NA>
2  a  1  a oui          11  3  a TRUE          13  1  a oui
3  z  2  z non          12  7  z FALSE          12  2  z non
```

```

4   a   4   a   oui           8   4   a FALSE           34   9   a   oui
  second_ids
1      <NA>
2         13
3         12
4         11

```

```

print(any_join_df(inpt_df_l=list(df2, df1, df3), join_type=c(1, 3), id_v=c("ids", "second_ids"),
                  excl_col=c(), rtn_col=c()))

```

```

  ids val  ids bool second_ids val  ids last second_ids val  ids last
1 a13   3   a  TRUE         13 <NA> <NA> <NA>         <NA>   1   a   oui
2 z12   7   z FALSE         12   2   z   non         12   2   z   non
3 z8    2   z FALSE          8 <NA> <NA> <NA>         <NA> <NA> <NA> <NA>
4 a34   4   a FALSE         34 <NA> <NA> <NA>         <NA> <NA> <NA> <NA>
5 a22   1   a  TRUE         22 <NA> <NA> <NA>         <NA> <NA> <NA> <NA>
6 a12   2   a  TRUE         12 <NA> <NA> <NA>         <NA> <NA> <NA> <NA>
7 a13 <NA> <NA> <NA>         <NA> <NA> <NA> <NA>         <NA> <NA> <NA> <NA>
8 a11 <NA> <NA> <NA>         <NA>   1   a   oui         11   9   a   oui
9 z12 <NA> <NA> <NA>         <NA> <NA> <NA> <NA>         <NA> <NA> <NA> <NA>
10 a8 <NA> <NA> <NA>         <NA>   4   a   oui          8   4   a   oui
  second_ids
1         13
2         12
3      <NA>
4      <NA>
5      <NA>
6      <NA>
7      <NA>
8         11
9      <NA>
10        8

```

```

print(any_join_df(inpt_df_l=list(df1, df2, df3), join_type=c(1), id_v=c("ids"),
                  excl_col=c(), rtn_col=c()))

```

```

  ids val  ids last second_ids val  ids bool second_ids val  ids last
1   e   1   e   oui         13 <NA> <NA> <NA>         <NA> <NA> <NA> <NA>
2   a   1   a   oui         11   3   a  TRUE         13   1   a   oui
3   z   2   z   non         12   7   z FALSE         12   2   z   non
4   a   4   a   oui          8   4   a FALSE         34   9   a   oui
  second_ids
1      <NA>
2         13
3         12
4         11

```

append_row

append_row

Description

Append the last row from dataframe to the another or same dataframe

Usage

```
append_row(df_in, df, hmn = 1, na_col = c(), unique_do_not_know = NA)
```

Arguments

df_in	is the dataframe from which the row will append to another or the same dataframe
df	is the dataframe to which the row will append
hmn	is how many time the last row will be appended
na_col	is a vector containing the columns that won't append and will be replaced by another value (unique_do_not_know)
unique_do_not_know	is the value of the non appending column in the appending row

appndr

appndr

Description

Append to a vector "inpt_v" a special value "val" n times "mmn". The appending begins at "strt" index.

Usage

```
appndr(inpt_v, val = NA, hmn, strt = "max")
```

Arguments

inpt_v	is the input vector
val	is the special value
hmn	is the number of special value element added
strt	is the index from which appending begins, defaults to max which means the end of "inpt_v"

calc_occu_v

calc_occu_v

Description

Rearranges the index of a vector "w_v" to match the occurrences of the common elements in another vector "f_v"

Usage

```
calc_occu_v(f_v, w_v, nvr_here = NA)
```

Examples

```
print(calc_occu_v(f_v=c("e", "a", "z", NA, "a"), w_v=c("a", "a", "z")))
```

```
[1] 1 3 2
```

can_be_num	<i>can_be_num</i>
------------	-------------------

Description

Return TRUE if a variable can be converted to a number and FALSE if not (supports float)

Usage

```
can_be_num(x)
```

Arguments

x	is the input value
---	--------------------

change_date	<i>change_date</i>
-------------	--------------------

Description

Allow to add to a date second-minute-hour-day-month-year

Usage

```
change_date (
  date_,
  sep_,
  day_ = NA,
  month_ = NA,
  year_ = NA,
  hour_ = NA,
  min_ = NA,
  second_ = NA,
  frmt = "snhdmy"
)
```

Arguments

date_	is the input date
sep_	is the date separator
day_	is the day to add (can be negative)
month_	is the month to add (can be negative)
year_	is the year to add (can be negative)
hour_	is the hour to add (can be negative)
min_	is the minute to add (can be negative)
second_	is the second to add (can be negative)
frmt	is the format of the input date, (default set to "snhdmy" (second, minute, hour, day, month, year), so all variable are taken in count), if you only want to work with standard date for example change this variable to "dmy"

chr_removr	<i>chr_removr</i>
------------	-------------------

Description

Allow to remove certain characters contained in a vector "ptrn_v" from elements in a another vector "inpt_v".

Usage

```
chr_removr(inpt_v, ptrn_v)
```

Arguments

inpt_v	is the input vector containing all the elements that may have the characters to be removed
ptrn_v	is the vector containing all the characters that will be removed

Examples

```
print(chr_removr(inpt_v=c("oui?", "!oui??", "non", "!non"), ptrn_v=c("?")))  
[1] "oui" "oui" "non" "non"  
  
print(chr_removr(inpt_v=c("oui?", "!oui??", "non", "!non"), ptrn_v=c("?", "!")))  
[1] "oui" "oui" "non" "non"
```

closer_ptrn	<i>closer_ptrn</i>
-------------	--------------------

Description

Take a vector of patterns as input and output each chosen word with their closest patterns from chosen patterns.

Usage

```
closer_ptrn(  
  inpt_v,  
  default_val = "?",  
  base_v = c(default_val, letters),  
  excl_v = c(),  
  rtn_v = c(),  
  sub_excl_v = c(),  
  sub_rtn_v = c()  
)
```


Arguments

<code>inpt_v</code>	is the input vector containing all the patterns
<code>excl_v</code>	is the vector containing all the patterns from <code>inpt_v</code> to exclude for comparing them to others patterns. If this parameter is filled, so <code>"rtn_v"</code> must be empty.
<code>rtn_v</code>	is the vector containing all the patterns from <code>inpt_v</code> to keep for comparing them to others patterns. If this parameter is filled, so <code>"rtn_v"</code> must be empty.
<code>sub_excl_v</code>	is the vector containing all the patterns from <code>inpt_v</code> to exclude for using them to compare to another pattern. If this parameter is filled, so <code>"sub_rtn_v"</code> must be empty.
<code>sub_rtn_v</code>	is the vector containing all the patterns from <code>inpt_v</code> to retain for using them to compare to another pattern. If this parameter is filled, so <code>"sub_excl_v"</code> must be empty.

Examples

```
print(closer_ptrn(inpt_v=c("bonjour", "lpoerc", "nonnour", "bonnour", "nonjour", "aurevoir")

[[1]]
[1] "bonjour"

[[2]]
[1] "lpoerc" "nonnour" "bonnour" "nonjour" "aurevoir"

[[3]]
[1] 1 1 2 7 8

[[4]]
[1] "lpoerc"

[[5]]
[1] "bonjour" "nonnour" "bonnour" "nonjour" "aurevoir"

[[6]]
[1] 7 7 7 7 7

[[7]]
[1] "nonnour"

[[8]]
[1] "bonjour" "lpoerc" "bonnour" "nonjour" "aurevoir"

[[9]]
[1] 1 1 2 7 8

[[10]]
[1] "bonnour"

[[11]]
[1] "bonjour" "lpoerc" "nonnour" "nonjour" "aurevoir"

[[12]]
[1] 1 1 2 7 8

[[13]]
```

```

[1] "nonjour"

[[14]]
[1] "bonjour" "lpoerc" "nonnour" "bonnour" "aurevoir"

[[15]]
[1] 1 1 2 7 8

[[16]]
[1] "aurevoir"

[[17]]
[1] "bonjour" "lpoerc" "nonnour" "bonnour" "nonjour"

[[18]]
[1] 7 8 8 8 8
print(closer_ptrn(inpt_v=c("bonjour", "lpoerc", "nonnour", "bonnour", "nonjour", "aurevoir",
                           sub_excl_v=c("nonnour"))))

[1] 3 5
[[1]]
[1] "bonjour"

[[2]]
[1] "lpoerc" "bonnour" "nonjour" "aurevoir"

[[3]]
[1] 1 1 7 8

[[4]]
[1] "lpoerc"

[[5]]
[1] "bonjour" "bonnour" "nonjour" "aurevoir"

[[6]]
[1] 7 7 7 7

[[7]]
[1] "bonnour"

[[8]]
[1] "bonjour" "lpoerc" "bonnour" "nonjour" "aurevoir"

[[9]]
[1] 0 1 2 7 8

[[10]]
[1] "aurevoir"

[[11]]
[1] "bonjour" "lpoerc" "nonjour" "aurevoir"

[[12]]
[1] 0 7 8 8

```

closer_ptrn_adv	<i>closer_ptrn_adv</i>
-----------------	------------------------

Description

Allow to find how patterns are far or near between each other relatively to a vector containing characters at each index ("base_v"). The function gets the sum of the indexes of each pattern letter relatively to the characters in base_v. So each pattern can be compared.

Usage

```
closer_ptrn_adv(
  inpt_v,
  res = "raw_stat",
  default_val = "?",
  base_v = c(default_val, letters),
  c_word = NA
)
```

Arguments

inpt_v	is the input vector containing all the patterns to be analyzed
res	is a parameter controlling the result. If set to "raw_stat", each word in inpt_v will come with its score (indexes of its letters relatively to base_v). If set to something else, so "c_word" parameter must be filled.
default_val	is the value that will be added to all patterns that do not equal the length of the longest pattern in inpt_v. Those get this value added to make all patterns equal in length so they can be compared, defaults to "?"
base_v	is the vector from which all pattern get its result (letters indexes for each pattern relatively to base_v), defaults to c("default_val", letters). "default_val" is another parameter and letters is all the western alphabetic letters in a vector
c_word	is a pattern from which the nearest to the farthest pattern in inpt_v will be compared

Examples

```
print(closer_ptrn_adv(inpt_v=c("aurevoir", "bonnour", "nonnour", "fin", "mois", "bonjour")

[[1]]
[1] 1 5 15 17 38 65

[[2]]
[1] "bonjour" "bonnour" "aurevoir" "nonnour" "mois" "fin"

print(closer_ptrn_adv(inpt_v=c("aurevoir", "bonnour", "nonnour", "fin", "mois")))

[[1]]
[1] 117 107 119 37 64

[[2]]
[1] "aurevoir" "bonnour" "nonnour" "fin" "mois"
```

closest_date	<i>closest_date</i>
--------------	---------------------

Description

return the closest dates from a vector compared to the input date

Usage

```
closest_date (
  vec,
  date_,
  frmt,
  sep_ = "/",
  sep_vec = "/",
  only_ = "both",
  head = NA
)
```

Arguments

vec	is a vector containing the dates to be compared to the input date
date_	is the input date
frmt	is the format of the input date, (default set to "snhdmy" (second, minute, hour, day, month, year), so all variable are taken in count), if you only want to work with standard date for example change this variable to "dmy"
sep_	is the separator for the input date
sep_vec	is the separator for the dates contained in vec
only_	is can be changed to "+" or "-" to repectively only return the higher dates and the lower dates (default set to "both")
head	is the number of dates that will be returned (default set to NA so all dates in vec will be returned)

clusterizer_v	<i>clusterizer_v</i>
---------------	----------------------

Description

Allow to output clusters of elements. Takes as input a vector "inpt_v" containing a sequence of number. Can also take another vector "w_v" that has the same size of inpt_v because its elements are related to it. The way the clusters are made is related to an accuracy value which is "c_val". It means that if the difference between the values associated to 2 elements is superior to c_val, these two elements are in distinct clusters.

Usage

```
clusterizer_v(inpt_v, w_v = NA, c_val)
```

Arguments

`inpt_v` is the vector containing the sequence of number
`w_v` is the vector containing the elements related to `inpt_v`, defaults to NA
`c_val` is the accuracy of the clusterization

Examples

```
print(clusterizer_v(inpt_v=sample.int(20, 26, replace=T), w_v=NA, c_val=0.9))

[[1]]
[[1]][[1]]
[1] "j" "v"

[[1]][[2]]
[1] "x"

[[1]][[3]]
[1] "e" "m" "p" "s" "t" "b" "q" "z" "f"

[[1]][[4]]
[1] "a" "i"

[[1]][[5]]
[1] "c" "n" "o" "g" "u" "y" "h" "l"

[[1]][[6]]
[1] "d" "r" "w" "k"

[[2]]
[1] "1" "2" "-" "4" "4" "-" "6" "10" "-" "12" "12" "-" "14" "16" "-"
[16] "18" "19"

print(clusterizer_v(inpt_v=sample.int(40, 26, replace=T), w_v=letters, c_val=0.29))

[[1]]
[[1]][[1]]
[1] "a" "b" "c" "d" "e" "f" "g" "h"

[[1]][[2]]
[1] "i" "j" "k" "l"

[[1]][[3]]
[1] "m" "n"

[[1]][[4]]
[1] "o" "p" "q" "r" "s" "t" "u" "v" "w" "x" "y" "z"

[[2]]
[1] "1" "5" "-" "8" "10" "-" "12" "13" "-" "15" "20"
```

cost_and_taxes	<i>cost_and_taxes</i>
----------------	-----------------------

Description

Allow to calculate basic variables related to cost and taxes from a bunch of products (elements) So put every variable you know in the following order:

Usage

```
cost_and_taxes (
  qte = NA,
  pu = NA,
  prix_ht = NA,
  tva = NA,
  prix_ttc = NA,
  prix_tva = NA,
  pu_ttc = NA,
  adjust = NA,
  prix_d_ht = NA,
  prix_d_ttc = NA,
  pu_d = NA,
  pu_d_ttc = NA
)
```

Arguments

qte	is the quantity of elements
pu	is the price of a single elements without taxes
prix_ht	is the duty-free price of the whole set of elements
tva	is the percentage of all taxes
prix_ttc	is the price of all the elements with taxes
prix_tva	is the cost of all the taxes
pu_ttc	is the price of a single element taxes included
adjust	is the discount percentage
prix_d_ht	is the free-duty price of an element after discount
prix_d_ttc	is the price with taxes of an element after discount
pu_d	is the price of a single element after discount and without taxes
pu_d_ttc	is the free-duty price of a single element after discount the function return a vector with the previous variables in the same order those that could not be calculated will be represented with NA value

data_gen

*data_gen***Description**

Allo to generate in a csv all kind of data you can imagine according to what you provide

Usage

```
data_gen(
  type_ = c("number", "mixed", "string"),
  strt_l = c(0, 0, 10),
  nb_r = c(50, 10, 40),
  output = "gened.csv",
  properties = c("1-5", "1-5", "1-5"),
  type_distri = c("random", "random", "random"),
  str_source = c("a", "b", "c", "d", "e", "f", "g", "h", "i", "j", "k", "l", "m",
    "o", "p", "q", "r", "s", "t", "u", "w", "x", "y", "z"),
  round_l = c(0, 0, 0),
  sep_ = ",",
)
```

Arguments

type_	is a vector for wich argument is a column, a column can be made of numbers ("number"), string ("string") or both ("mixed")
strt_l	is a vector containing for each column the row from which the data will begin to be generated
nb_r	is a vector containing for each column, the number of row full from generated data
output	is the name of the output csv file
properties	is linked to type_distri because it is the parameters ("min_val-max_val") for "random type", ("u-x") for the poisson distribution, ("u-d") for gaussian distribution
type_distri	is a vector which, for each column, associate a type of distribution ("random", "poisson", "gaussian"), it meas that non only the number but also the length of the string will be randomly generated according to these distribution laws
str_source	is the source (vector) from which the character creating random string are (default set to the occidental alphabet)
round_l	is a vector which, for each column containing number, associate a round value
sep_	is the separator used to write data in the csv

Value

new generated data in addition to saving it in the output

data_meshup	<i>data_meshup</i>
-------------	--------------------

Description

Allow to automatically arrange 1 dimensional data according to vector and parameters

Usage

```
data_meshup (
  data,
  cols = NA,
  file_ = NA,
  sep_ = ";",
  organisation = c(2, 1, 0),
  unic_sep1 = "_",
  unic_sep2 = "-"
)
```

Arguments

data	is the data provided (vector) each column is separated by a unic separator and each dataset from the same column is separated by another unic separator (ex: <code>c("", c("d", "-", "e", "-", "f"), "", c("a", "a1", "-", "b", "-", "c", "c1"))</code>)
cols	is the colnames of the data generated in a csv
file_	is the file to which the data will be outputed
sep_	is the separator of the csv outputed
organisation	is the way variables include themselves, for instance ,resuming precedent example, if organisation=c(1, 0) so the data output will be: d, a d, a1 e, c f, c f, c1
unic_sep1	is the unic separator between variables (default is "_")
unic_sep2	is the unic separator between datasets (default is "-")

date_sort	<i>date_sort</i>
-----------	------------------

Description

Allow to ascendely or descendely sort dates in a vector.

Usage

```
date_sort(vec, asc = F, sep = "-")
```

Arguments

vec	is the vector containing the dates.
asc	is a boolean variable, that if set to TRUE will sort the dates ascendely and descendely if set to FALSE
sep	is the separator of the date strings ex: "11-12-1998" the separator is "-"

days_from_month	<i>days_from_month</i>
-----------------	------------------------

Description

Allow to find the number of days month from a month date, take in count leap year

Usage

```
days_from_month(date_, sep_)
```

Arguments

date_	is the input date
sep_	is the separator of the input date

df_tuned	<i>df_tuned</i>
----------	-----------------

Description

Allow to return a list from a dataframe following these rules: First situation, I want the vectors from the returned list be composed of values that are separated by special values contained in a vector ex: data.frame(c(1, 1, 2, 1), c(1, 1, 2, 1), c(1, 1, 1, 2)) will return list(c(1, 1), c(1, 1, 1), c(1, 1, 1, 1)) or list(c(1, 1, 2), c(1, 1, 1, 2), c(1, 1, 1, 1, 2)) if i have chosen to take in count the 2. As you noticed here the value to stop is 2 but it can be several contained in a vector Second situation: I want to return a list for every jump of 3. If i take this dataframe data.frame(c(1, 1, 2, 1, 4, 4), c(1, 1, 2, 1, 3, 3), c(1, 1, 1, 2, 3, 3)) it will return list(c(1, 1, 2), c(1, 4, 4), c(1, 1, 2), c(1, 3, 3), c(1, 1, 1), c(2, 3, 3))

Usage

```
df_tuned(df, val_to_stop, index_rc = NA, included = "yes")
```

Arguments

df	is the input data.frame
val_to_stop	is the vector containing the values to stop
index_rc	is the value for the jump (default set to NA so default will be first case)
included	is if the values to stop has to be also returned in the vectors (defaultn set to "yes")

diff_xlsx

*diff_xlsx***Description**

Allow to see the difference between two datasets and output it into an xlsx file. If the dimensions of the new datasets are bigger than the old one, only the matching cells will be compared, if the dimensions of the new one are lower than the old one, there will be an error.

Usage

```
diff_xlsx(
  file_,
  sht,
  v_old_begin,
  v_old_end,
  v_new_begin,
  v_new_end,
  df2 = NA,
  overwrite = T,
  color_ = "red",
  pattern = "",
  output = "out.xlsx",
  new_val = T,
  pattern_only = T
)
```

Arguments

<code>file_</code>	is the file where the data is
<code>sht</code>	is the sheet where the data is
<code>v_old_begin</code>	is a vector containing the coordinates (row, column) where the data to be compared starts
<code>v_old_end</code>	is the same but for its end
<code>v_new_begin</code>	is the coordinates where the comparator data starts
<code>v_new_end</code>	is the same but for its end If the dimensions of the new datasets are bigger than the old one, only the matching cells will be compared, if the dimensions of the new one are lower than the old one, there will be an error.
<code>df2</code>	is optional, if the comparator dataset is directly a dataframe
<code>overwrite</code>	allow to overwrite differences is (set to T by default)
<code>color_</code>	is the color the differences will be outputed
<code>pattern</code>	is the pattern that will be added to the differences if overwritten is set to TRUE
<code>output</code>	is the name of the outputed xlsx (can be set to NA if no output)
<code>new_val</code>	if overwrite is TRUE, then the differences will be overwritten by the comparator data
<code>pattern_only</code>	will cover differences by pattern if overwritten is set to TRUE

equalizer_v	<i>equalizer_v</i>
-------------	--------------------

Description

Takes a vector of character as an input and returns a vector with the elements at the same size. The size can be chosen via depth parameter.

Usage

```
equalizer_v(inpt_v, depth = "max", default_val = "?")
```

Arguments

inpt_v	is the input vector containing all the characters
depth	is the depth parameter, defaults to "max" which means that it is equal to the character number of the element(s) in inpt_v that has the most
default_val	is the default value that will be added to the output characters if those has an inferior length (characters) than the value of depth

Examples

```
print(equalizer_v(inpt_v=c("aa", "zzz", "q"), depth=2))
[1] "aa" "zz" "q?"

print(equalizer_v(inpt_v=c("aa", "zzz", "q"), depth=12))
[1] "aa???????????" "zzz???????????" "q?????????????"
```

extrm_dates	<i>extrm_dates</i>
-------------	--------------------

Description

Allow to find the minimum or the maximum of a date in a vector. The format of dates is Year/Month/Day.

Usage

```
extrm_dates(inpt_l, extrm = "min", sep = "-")
```

Arguments

inpt_l	is the input vector
extrm	is either "min" or "max", defaults to "min"
sep	is the separator of the dates, defaults to "-"

extrt_only_v	<i>extrt_only_v</i>
--------------	---------------------

Description

return the elements from a vector "inpt_v" that are in another vector "pttrn_v"

Usage

```
extrt_only_v(inpt_v, pttrn_v)
```

Arguments

inpt_v	is the input vector
pttrn_v	is the vector contining all the elements that can be in inpt_v

Examples

```
print(extrt_only_v(inpt_v=c("oui", "non", "peut", "oo", "ll", "oui", "non", "oui", "oui"))  
[1] "oui" "oo"  "oui" "oui" "oui"
```

file_rec	<i>file_rec</i>
----------	-----------------

Description

Allow to get all the files recursively from a path according to an end and start depth value. If you want to have an other version of this function that uses a more sophisticated algorythm (which can be faster), check file_rec2. Depth example: if i have dir/dir2/dir3, dir/dir2b/dir3b, i have a depth equal to 3

Usage

```
file_rec(xmax, xmin = 1, pathc = ".")
```

Arguments

xmax	is the end depth value
xmin	is the start depth value
pathc	is the reference path

file_rec2

file_rec2

Description

Allow to find the directories and the subdirectories with a specified end and start depth value from a path. This function might be more powerfull than file_rec because it uses a custom algorythm that does not nee to perform a full recursive search before tuning it to only find the directories with a good value of depth. Depth example: if i have dir/dir2/dir3, dir/dir2b/dir3b, i have a depth equal to 3

Usage

```
file_rec2(xmax, xmin = 1, pathc = ".")
```

Arguments

xmax	is the depth value
xmin	is the minimum value of depth
pathc	is the reference path, from which depth value is equal to 1

fillr

fillr

Description

Allow to fill a vector by the last element n times

Usage

```
fillr(inpt_v, ptrn_fill = "...\\d")
```

Arguments

inpt_v	is the input vector
ptrn_fill	is the pattern used to detect where the function has to fill the vector by the last element n times. It defaults to "...\\d" where "\\d" is the regex for an int value. So this paramater has to have "\\d" which designates n.

Examples

```
fillr(c("a", "b", "...3", "c"))
```

fittr_v

fittr_v

Description

Return the indexes of elements contained in "w_v" according to "f_v"

Usage

```
fittr_v(f_v, w_v, nvr_here = NA)
```

Arguments

f_v is the input vector

w_v is the vector containing the elements that can be in f_v

Examples

```
print(fittr_v(f_v=c("non", "non", "non", "oui"), w_v=c("oui", "non", "non")))

[1] 4 1 2
```

fixer_nest_v

fixer_nest_v

Description

Retur the elements of a vector "wrk_v" (1) that corresponds to the pattern of elements in another vector "cur_v" (2) according to another vector "pttrn_v" (3) that contains the patterof eleemnts.

Usage

```
fixer_nest_v(cur_v, pttrn_v, wrk_v)
```

Examples

```
print(fixer_nest_v(cur_v=c("oui", "non", "peut-etre", "oui", "non", "peut-etre"), pttrn_v=c(1, 2, 3, 4, 5, 6), wrk_v=c(1, 2, 3, 4, 5, 6)))

[1] 1 2 3 4 5 6

print(fixer_nest_v(cur_v=c("oui", "non", "peut-etre", "oui", "non", "peut-etre"), pttrn_v=c(1, 2, 3, 4, 5, 6), wrk_v=c(1, 2, 3, 4, 5, 6)))

[1] 1 2 NA 4 5 NA
```

format_date	<i>format_date</i>
-------------	--------------------

Description

Allow to convert xx-month-xxxx date type to xx-xx-xxxx

Usage

```
format_date(f_dialect, sentc, sep_in = "-", sep_out = "-")
```

Arguments

f_dialect	are the months from the language of which the month come
sentc	is the date to convert
sep_in	is the separator of the dat input (default is "-")
sep_out	is the separator of the converted date (default is "-")

geo_min	<i>geo_min</i>
---------	----------------

Description

Return a dataframe containing the nearest geographical points (row) according to established geographical points (column).

Usage

```
geo_min(inpt_df, established_df)
```

Arguments

inpt_df	is the input dataframe of the set of geographical points to be classified, its firts column is for latitude, the second for the longitude and the third, if exists, is for the altitude. Each point is one row.
established_df	is the dataframe containing the coordiantes of the established geographical points

Examples

```
in_ <- data.frame(c(11, 33, 55), c(113, -143, 167))
in2_ <- data.frame(c(12, 55), c(115, 165))
print(geo_min(inpt_df=in_, established_df=in2_))

in_ <- data.frame(c(51, 23, 55), c(113, -143, 167), c(6, 5, 1))
in2_ <- data.frame(c(12, 55), c(115, 165), c(2, 5))
geo_min(inpt_df=in_, established_df=in2_)
```

get_rec

get_rec

Description

Allow to get the value of directorie depth from a path.

Usage

```
get_rec(pathc = ".")
```

Arguments

pathc is the reference path example: if i have dir/dir2/dir3, dir/dir2b/dir3b, i have a depth equal to 3

globe

globe

Description

Allow to calculate the distances between a set of geographical points and another established geographical point. If the altitude is not filled, so the result returned won't take in count the altitude.

Usage

```
globe(lat_f, long_f, alt_f = NA, lat_n, long_n, alt_n = NA)
```

Arguments

lat_f is the latitude of the established geographical point
long_f is the longitude of the established geographical point
alt_f is the altitude of the established geographical point, defaults to NA
lat_n is a vector containing the latitude of the set of points
long_n is a vector containing the longitude of the set of points
alt_n is a vector containing the altitude of the set of points, defaults to NA

Examples

```
globe(lat_f=23, long_f=112, alt_f=NA, lat_n=c(2, 82), long_n=c(165, -55), alt_n=NA)
```

grouppr_df	<i>grouppr_df</i>
------------	-------------------

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 dataframe according to the cell value. This function is based on `see_df` and `nestr_df2` functions.

Usage

```
grouppr_df(inpt_df, condition_lst, val_lst, conjunction_lst, rtn_val_pos = c())
```

Arguments

`inpt_df` 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()
df1 <- 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)))

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

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

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

grouppr_df(inpt_df=df1, val_lst=val_lst, condition_lst=condition_lst,
conjunction_lst=conjunction_lst, rtn_val_pos=rtn_val_pos)
```

incr_fillr	<i>incr_fillr</i>
------------	-------------------

Description

Take a vector uniquely composed by double and sorted ascendingly, a step, another vector of elements whose length is equal to the length of the first vector, and a default value. If an element of the vector is not equal to its predecessor minus a user defined step, so these can be the output according to the parameters (see example):

Usage

```
incr_fillr(inpt_v, wrk_v = NA, default_val = NA, step = 1)
```

Arguments

inpt_v	is the asending double only composed vector
wrk_v	is the other vector (size equal to inpt_v), defaults to NA
default_val	is the default value put when the difference between two following elements of inpt_v is greater than step, defaults to NA
step	is the allowed difference between two elements of inpt_v

Examples

```
print(incr_fillr(inpt_v=c(1, 2, 4, 5, 9, 10),
                 wrk_v=NA,
                 default_val="increasing"))

[1] 1 2 3 4 5 6 7 8 9 10

print(incr_fillr(inpt_v=c(1, 1, 2, 4, 5, 9),
                 wrk_v=c("ok", "ok", "ok", "ok", "ok"),
                 default_val=NA))

[1] "ok" "ok" "ok" NA "ok" "ok" NA NA NA

print(incr_fillr(inpt_v=c(1, 2, 4, 5, 9, 10),
                 wrk_v=NA,
                 default_val="NAN"))

[1] "1" "2" "NAN" "4" "5" "NAN" "NAN" "NAN" "9" "10"
```

insert_df	<i>insert_df</i>
-----------	------------------

Description

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

Usage

```
insert_df(df_in, df_ins, ins_loc)
```

Arguments

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

inter_max	<i>inter_max</i>
-----------	------------------

Description

Takes as input a list of vectors composed of ints or floats ascendly ordered (intervals) that can have a different step to one of another element ex: list(c(0, 2, 4), c(0, 4), c(1, 2, 2.3)) The function will return the list of lists altered according to the maximum step found in the input list.

Usage

```
inter_max(inpt_l, max_ = -1000, get_lst = T)
```

Arguments

inpt_l	is the input list
max_	is a value you are sure is the minimum step value of all the sub-lists
get_lst	is the parameter that, if set to True, will keep the last values of vectors in the return value if the last step exceeds the end value of the vector.

inter_min	<i>inter_min</i>
-----------	------------------

Description

Takes as input a list of vectors composed of ints or floats ascendly ordered (intervals) that can have a different step to one of another element ex: list(c(0, 2, 4), c(0, 4), c(1, 2, 2.3)) This function will return the list of vectors with the same steps preserving the begin and end value of each interval. The way the algorythmn searches the common step of all the sub-lists is also given by the user as a parameter, see how_to paramaters.

Usage

```
inter_min(
  inpt_l,
  min_ = 1000,
  sensi = 3,
  sensi2 = 3,
  how_to_op = c("divide"),
  how_to_val = c(3)
)
```

Arguments

inpt_l	is the input list containing all the intervals
min_	is a value you are sure is superior to the maximum step value in all the intervals
sensi	is the decimal accuracy of how the difference between each value n to n+1 in an interval is calculated
sensi2	is the decimal accuracy of how the value with the common step is calculated in all the intervals
how_to_op	is a vector containing the operations to perform to the pre-common step value, defaults to only "divide". The operations can be "divide", "substract", "multiply" or "add". All type of operations can be in this parameter.
how_to_val	is a vector containing the value relatives to the operations in hot_to_op, defaults to 3 output from ex:

Examples

```
[[0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 1.7, 1.8, 1.9, 2.0, 2.1, 2.2, 2.3, 2.4, 2.5, 2.6, 2.7, 2.8, 2.9, 3.0, 3.1, 3.2, 3.3, 3.4, 3.5, 3.6, 3.7, 3.8, 3.9, 4.0], [0.5, 0.6, 0.7, 0.8, 0.9, 1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 1.7, 1.8, 1.9, 2.0, 2.1, 2.2, 2.3, 2.4, 2.5, 2.6, 2.7, 2.8, 2.9, 3.0, 3.1, 3.2, 3.3, 3.4, 3.5, 3.6, 3.7, 3.8, 3.9, 4.0], [1, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 1.7, 1.8, 1.9, 2.0, 2.1, 2.2, 2.3]]
```

letter_to_nb	<i>letter_to_nb</i>
--------------	---------------------

Description

Allow to get the number of a spreadsheet based column by the letter ex: AAA = 703

Usage

```
letter_to_nb(letter)
```

Arguments

letter	is the letter (name of the column)
--------	------------------------------------

list_files	<i>list_files</i>
------------	-------------------

Description

A list.files() based function addressing the need of listing the files with extension a or or extension b ...

Usage

```
list_files(patternc, pathc = ".")
```

Arguments

patternc	is a vector containing all the extensions you want
pathc	is the path, can be a vector of multiple path because list.files() supports it.

lst_flatnr	<i>lst_flatnr</i>
------------	-------------------

Description

Flatten a list to a vector

Usage

```
lst_flatnr(inpt_l)
```

Examples

```
print(lst_flatnr(inpt_l=list(c(1, 2), c(5, 3), c(7, 2, 7))))
[1] 1 2 5 3 7 2 7
```

match_n	<i>match_n</i>
---------	----------------

Description

Allow to get the indexes for the nth occurrence of a value in a vector. Example: c(1, 2, 3, 1, 2), the first occurrence of 1 and 2 is at index 1 and 2 respectively, but the second occurrence is respectively at the 4th and 5th index.

Usage

```
match_n(vec, mc, n = 1, wnb = "#####")
```

Arguments

vec	is th input vector
mc	is a vector containing the values you want to get the index for the nth occurrence in vec
n	is the value of the occurrence
wnb	is a string you are sure is not in mc

match_n2

match_n2

Description

Allow to get the indexes for the nth occurrence of a value in a vector. Example: `c(1, 2, 3, 1, 2)`, the first occurrence of 1 and 2 is at index 1 and 2 respectively, but the second occurrence is respectively at the 4th and 5th index.

Usage

```
match_n2(vec, mc, n, wnb = "#####")
```

Arguments

<code>vec</code>	is the input vector
<code>mc</code>	is a vector containing the values you want to get the index for the nth occurrence in <code>vec</code>
<code>n</code>	is a vector containing the occurrences for each value in <code>mc</code> so if i have <code>mc <- c(3, 27)</code> and <code>n <- c(1, 2)</code> , i want the first occurrence for 3 and the second for 27 in <code>vec</code> . If the length of <code>n</code> is inferior of the length of <code>mc</code> , <code>n</code> will extend with its last value as new arguments. It means that if <code>mc <- c(3, 27)</code> but <code>n <- c(1)</code> so <code>n</code> will extend to <code>c(1, 1)</code> , so we will get the first occurrence of 3 and 27 in <code>vec</code> .
<code>wnb</code>	is a string you are sure is not in <code>mc</code>

multitud

multitud

Description

From a list containing vectors allow to generate a vector following this rule: `list(c("a", "b"), c("1", "2"), c("A", "Z", "E")) -> c("a1A", "a2A", "b1A", "b2A", "a1Z", ...)`

Usage

```
multitud(l, sep_ = "")
```

Arguments

<code>l</code>	is the list
<code>sep_</code>	is the separator between elements (default is set to "" as you see in the example)

nb_to_letter	<i>nb_to_letter</i>
--------------	---------------------

Description

Allow to get the letter of a spreadsheet based column by the number ex: 703 = AAA

Usage

```
nb_to_letter(x)
```

Arguments

x is the number of the column

nestr_df1	<i>nestr_df1</i>
-----------	------------------

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) do 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 det to NA.

Usage

```
nestr_df1(inptf_df, inptt_pos_df, nestr_df, yes_val = T, inptt_neg_df = NA)
```

Arguments

inptf_df is the input dataframe (1b)
 inptt_pos_df is the dataframe (2b) that corresponds to the (1a) values
 nestr_df is the dataframe (2b) that has the special value (2a)
 yes_val is the special value (2a)
 inpt_neg_df is the dataframe (4b) that has the (3a) values, defaults to NA

Examples

```
nestr_df1(inptf_df=data.frame(c(1, 2, 1), c(1, 5, 7)),
inptt_pos_df=data.frame(c(4, 4, 3), c(2, 1, 2)),
inptt_neg_df=data.frame(c(44, 44, 33), c(12, 12, 12)),
nestr_df=data.frame(c(TRUE, FALSE, TRUE), c(FALSE, FALSE, TRUE)), yes_val=TRUE)
```

nestr_df2	<i>nestr_df2</i>
-----------	------------------

Description

Allow to write a special value (1a) in the cells of a dataframe (1b) that correspond (row and column) to those 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_df2(inptf_df, rtn_pos, rtn_neg = NA, nestr_df, yes_val = T)
```

Arguments

- inptf_df is the input dataframe (1b)
- rtn_pos is the special value (1a)
- rtn_neg is the special value (3a)
- nestr_df is the dataframe (2b)
- yes_val is the special value (2a)

Examples

```
nestr_df2(inptf_df=data.frame(c(1, 2, 1), c(1, 5, 7)), rtn_pos="yes",  
rtn_neg="no", nestr_df=data.frame(c(TRUE, FALSE, TRUE), c(FALSE, FALSE, TRUE)), yes_val=T)
```

nest_v	<i>nest_v</i>
--------	---------------

Description

Nest two vectors according to the following parameters.

Usage

```
nest_v(f_v, t_v, step = 1, after = 1)
```

Arguments

- f_v is the vector that will welcome the nested vector t_v
- t_v is the imbricator vector
- step defines after how many elements of f_v the next element of t_v can be put in the output
- after defines after how many elements of f_v, the beginning of t_v can be put

Examples

```
print(nest_v(f_v=c(1, 2, 3, 4, 5, 6), t_v=c("oui", "oui2", "oui3", "oui4", "oui5", "oui6")  
[1] "1"      "2"      "oui"    "3"      "4"      "oui2"   "5"      "6"      "oui3"   "oui4"
```

occu	<i>occu</i>
------	-------------

Description

Allow to see the occurrence of each variable in a vector. Returns a dataframe with, as the first column, the all the unique variable of the vector and , in he second column, their occurrence respectively.

Usage

```
occu(inpt_v)
```

Arguments

`inpt_v` the input dataframe

<code>paste_df</code>	<i>paste_df</i>
-----------------------	-----------------

Description

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

Usage

```
paste_df(inpt_df, sep = "")
```

Arguments

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

Examples

```
print(paste_df(inpt_df=data.frame(c(1, 2, 1), c(33, 22, 55))))
[1] "133" "222" "155"
```

pattern_generator *pattern_generator*

Description

Allow to create patterns which have a part that is varying randomly each time.

Usage

```
pattern_generator(base_, from_, nb, hmn = 1, after = 1, sep = "")
```

Arguments

base_	is the pattern that will be kept
from_	is the vector from which the elements of the random part will be generated
nb	is the number of random pattern chosen for the varying part
hmn	is how many of varying pattern from the same base will be created
after	is set to 1 by default, it means that the varying part will be after the fixed part, set to 0 if you want the varying part to be before
sep	is the separator between all patterns in the returned value

pattern_gettr *pattern_gettr*

Description

Search for pattern(s) contained in a vector in another vector and return a list containing matched one (first index) and their position (second index) according to these rules: First case: Search for patterns strictly, it means that the searched pattern(s) will be matched only if the patterns contained in the vector that is being explored by the function are present like this c("pattern_searched", "other", ..., "pattern_searched") and not as c("other_thing pattern_searched other_thing", "other", ..., "pattern_searched other_thing") Second case: It is the opposite to the first case, it means that if the pattern is partially present like in the first position and the last, it will be considered like a matched pattern

Usage

```
pattern_gettr(
  word_,
  vct,
  occ = c(1),
  strict,
  btwn,
  all_in_word = "yes",
  notatall = "###"
)
```

Arguments

<code>word_</code>	is the vector containing the patterns
<code>vct</code>	is the vector being searched for patterns
<code>occ</code>	a vector containing the occurrence of the pattern in <code>word_</code> to be matched in the vector being searched, if the occurrence is 2 for the <code>nth</code> pattern in <code>word_</code> and only one occurrence is found in <code>vct</code> so no pattern will be matched, put "forever" to no longer depend on the occurrence for the associated pattern
<code>strict</code>	a vector containing the "strict" condition for each <code>nth</code> vector in <code>word_</code> ("strict" is the string to activate this option)
<code>btwn</code>	is a vector containing the condition ("yes" to activate this option) meaning that if "yes", all elements between two matched pattern in <code>vct</code> will be returned , so the patterns you enter in <code>word_</code> have to be in the order you think it will appear in <code>vct</code>
<code>all_in_word</code>	is a value (default set to "yes", "no" to activate this option) that, if activated, won't authorized a previous matched pattern to be matched again
<code>notatall</code>	is a string that you are sure is not present in <code>vct</code> REGEX can also be used as pattern

<code>pattern_tuning</code>	<i>pattern_tuning</i>
-----------------------------	-----------------------

Description

Allow to tune a pattern very precisely and output a vector containing its variations `n` times.

Usage

```
pattern_tuning(pattn, spe_nb, spe_l, exclude_type, hmn = 1, rg = c(0, 0))
```

Arguments

<code>pattn</code>	is the character that will be tuned
<code>spe_nb</code>	is the number of new character that will be replaced
<code>spe_l</code>	is the source vector from which the new characters will replace old ones
<code>exclude_type</code>	is character that won't be replaced
<code>hmn</code>	is how many output the function will return
<code>rg</code>	is a vector with two parameters (index of the first letter that will be replaced, index of the last letter that will be replaced) default is set to all the letters from the source pattern

ptrn_switchr	<i>ptrn_switchr</i>
--------------	---------------------

Description

Allow to switch, copy pattern for each element in a vector. Here a pattern is the values that are separated by a same separator. Example: "xx-xxx-xx" or "xx/xx/xxxx". The xx like values can be switched or copied from whatever index to whatever index. Here, the index is like this 1-2-3 etcetera, it is relative of the separator.

Usage

```
ptrn_switchr(inpt_l, f_idx_l = c(), t_idx_l = c(), sep = "-", default_val = NA)
```

Arguments

inpt_l	is the input vector
f_idx_l	is a vector containing the indexes of the pattern you want to be altered.
t_idx_l	is a vector containing the indexes to which the indexes in f_idx_l are related.
sep	is the separator, defaults to "-"
default_val	is the default value , if not set to NA, of the pattern at the indexes in f_idx_l. If it is not set to NA, you do not need to fill t_idx_l because this is the vector containing the indexes of the patterns that will be set as new values relatively to the indexes in f_idx_l. Defaults to NA.

Examples

```
ptrn_switchr(inpt_l=c("2022-01-11", "2022-01-14", "2022-01-21",  
"2022-01-01"), f_idx_l=c(1, 2, 3), t_idx_l=c(3, 2, 1))  
ptrn_switchr(inpt_l=c("2022-01-11", "2022-01-14", "2022-01-21",  
"2022-01-01"), f_idx_l=c(1), default_val="ee")
```

ptrn_twkr	<i>ptrn_twkr</i>
-----------	------------------

Description

Allow to modify the pattern length of element in a vector according to arguments. What is here defined as a pattern is something like this xx-xx-xx or xx/xx/xxx... So it is defined by the separator

Usage

```
ptrn_twkr(inpt_l, depth = "max", sep = "-", default_val = "0", add_sep = T)
```

Arguments

<code>inpt_l</code>	is the input vector
<code>depth</code>	is the number (numeric) of separator it will keep as a result. To keep the number of separator of the element that has the minimum amount of separator do <code>depth="min"</code> and <code>depth="max"</code> (character) for the opposite. This value defaults to "max".
<code>sep</code>	is the separator of the pattern, defaults to "-"
<code>default_val</code>	is the default val that will be placed between the separator, defaults to "00"
<code>add_sep</code>	defaults to TRUE. If set to FALSE, it will remove the separator for the patterns that are included in the interval between the depth amount of separator and the actual number of separator of the element.

Examples

```
library("stringr")
v <- c("2012-06-22", "2012-06-23", "2022-09-12", "2022")
ptrn_twkr(inpt_l=v, depth="max", sep="-", default_val="00", add_sep=TRUE)
```

rearangr_v

rearangr_v

Description

Rearranges a vector "w_v" according to another vector "inpt_v". inpt_v contains a sequence of number. inpt_v and w_v have the same size and their indexes are related. The output will be a vector containing all the elements of w_v rearranges in descending or asending order according to inpt_v

Usage

```
rearangr_v(inpt_v, w_v, how = "increasing")
```

Arguments

<code>inpt_v</code>	is the vector that contains the sequence of number
<code>w_v</code>	is the vector containing the elements related to inpt_v
<code>how</code>	is the way the elements of w_v will be outputed according to if inpt_v will be sorted ascendigly or descendigly

Examples

```
print(rearangr_v(inpt_v=c(23, 21, 56), w_v=c("oui", "peut", "non"), how="decreasing"))
[1] "non" "oui" "peut"
```

see_df

see_df

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_df(df, condition_l, val_l, conjunction_l = c(), rt_val = T, f_val = F)
```

Arguments

df	is the input dataframe
condition_l	is the vector of the possible conditions ("==", ">", "<", "!=", "%") (equal, greater than, lower than, not equal to, is divisible by), you can put the same condition n times.
val_l	is the list of vectors containing the values related to condition_l (so the vector of values has to be placed in the same order)
conjunction_l	contains the 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.

see_file

see_file

Description

Allow to get the filename or its extension

Usage

```
see_file(string_, index_ext = 1, ext = T)
```

Arguments

string_	is the input string
index_ext	is the occurrence of the dot that separates the filename and its extension
ext	is a boolean that if set to TRUE, will return the file extension and if set to FALSE, will return filename

see_idx

see_idx

Description

Allow to find the indexes of the elements of the first vector in the second. If the element(s) is not found, the element returned at the same index will be "FALSE".

Usage

```
see_idx(v1, v2, exclude_val = "#####", no_more = F)
```

Arguments

v1	is the first vector
v2	is the second vector
exclude_val	is a value you know is not present in the 2 vectors
no_more	is a boolean that, if set to TRUE, will remove all the first found value in the second vector after those has been found. It defaults to FALSE.

see_inside

see_inside

Description

Return a list containing all the column of the files in the current directory with a chosen file extension and its associated file and sheet if xlsx. For example if i have 2 files "out.csv" with 2 columns and "out.xlsx" with 1 column for its first sheet and 2 for its second one, the return will look like this: c(column_1, column_2, column_3, column_4, column_5, unique_separator, "1-2-out.csv", "3-3-sheet_1-out.xlsx", 4-5-sheet_2-out.xlsx)

Usage

```
see_inside(pattern_, path_ = ".", sep_ = c(", "), unique_sep = "#####", rec = F)
```

Arguments

pattern_	is a vector containin the file extension of the spreadsheets ("xlsx", "csv"...)
path_	is the path where are located the files
sep_	is a vector containing the separator for each csv type file in order following the operating system file order, if the vector does not match the number of the csv files found, it will assume the separator for the rest of the files is the same as the last csv file found. It means that if you know the separator is the same for all the csv type files, you just have to put the separator once in the vector.
unique_sep	is a pattern that you know will never be in your input files
rec	is a boolean allows to get files recursively if set to TRUE, defaults to TRUE If x is the return value, to see all the files name, position of the columns and possible sheet name associantied with, do the following: Examples: print(x[(grep(unique_sep, x)1+1):length(x)]) #If you just want to see the columns do the following: print(x1:(grep(unique_sep, x) - 1))

unique_pos	<i>unique_pos</i>
------------	-------------------

Description

Allow to find indexes of the unique values from a vector.

Usage

```
unique_pos(vec)
```

Arguments

vec	is the input vector
-----	---------------------

until_stnl	<i>until_stnl</i>
------------	-------------------

Description

Maxes a vector to a chosen length ex: if i want my vector c(1, 2) to be 5 of length this function will return me: c(1, 2, 1, 2, 1)

Usage

```
until_stnl(vec1, goal)
```

Arguments

vec1	is the input vector
goal	is the length to reach

val_replacer	<i>val_replacer</i>
--------------	---------------------

Description

Allow to replace value from dataframe to another one.

Usage

```
val_replacer(df, val_replaced, val_replacor = T, df_rpt = NA)
```

Arguments

`df` 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`

`df_rpt` is the replacement matrix and has to be the same dimension as `df`. Only the indexes that are equal to TRUE will be authorized indexes for the values to be replaced in the input matrix

vec_in_df	<i>vec_in_df</i>
-----------	------------------

Description

Allow to see if vectors are present in a dataframe ex: 1, 2, 1 3, 4, 1 1, 5, 8 the vector `c(4, 1)` with the coefficient 1 and the start position at the second column is contained in the dataframe

Usage

```
vec_in_df(df_, vec_l, coeff_, strt_l, distinct = "NA")
```

Arguments

`df_` is the input dataframe

`vec_l` is a list the vectors

`coeff_` is the related coefficient of the vector

`strt_l` is a vector containing the start position for each vector

`distinct` is a value you are sure is not in `df_`, defaults to "NA"

vlookup_df	<i>vlookup_df</i>
------------	-------------------

Description

Allow to perform a vlookup on a dataframe

Usage

```
vlookup_df(df, v_id, col_id = 1, included_col_id = "yes")
```

Arguments

df	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)

v_to_df	<i>v_to_df</i>
---------	----------------

Description

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

Usage

```
v_to_df(inpt_v, sep = "-")
```

Arguments

inpt_v	is the input vector
sep	is the separator used to seprate the columns

Examples

```
library("stringr")
v <- c("aa-yy-uu", "zz-gg-hhh", "zz-gg-hhh", "zz-gg-hhh")
v_to_df(inpt_v=v, sep="-")
```

Index

1:(grep(unique_sep, x) - 1), [40](#)
1, [40](#)
all_stat, [3](#)
any_join_df, [3](#)
append_row, [5](#)
appndr, [6](#)

calc_occu_v, [6](#)
can_be_num, [7](#)
change_date, [7](#)
chr_removr, [8](#)
closer_ptrn, [8](#)
closer_ptrn_adv, [11](#)
closest_date, [12](#)
clusterizer_v, [12](#)
cost_and_taxes, [14](#)

data_gen, [15](#)
data_meshup, [16](#)
date_sort, [16](#)
days_from_month, [17](#)
df_tuned, [17](#)
diff_xlsx, [18](#)

equalizer_v, [19](#)
extrm_dates, [19](#)
extrt_only_v, [20](#)

file_rec, [20](#)
file_rec2, [21](#)
fillr, [21](#)
fittr_v, [22](#)
fixer_nest_v, [22](#)
format_date, [23](#)

geo_min, [23](#)
get_rec, [24](#)
globe, [24](#)
groupr_df, [25](#)

incr_fillr, [26](#)
insert_df, [26](#)
inter_max, [27](#)
inter_min, [27](#)

letter_to_nb, [28](#)
list_files, [28](#)
lst_flatnr, [29](#)

match_n, [29](#)
match_n2, [30](#)
multitud, [30](#)

nb_to_letter, [31](#)
nest_v, [32](#)
nestr_df1, [31](#)
nestr_df2, [32](#)

occu, [33](#)

paste_df, [33](#)
pattern_generator, [34](#)
pattern_gettr, [34](#)
pattern_tuning, [35](#)
ptrn_switchr, [36](#)
ptrn_twkr, [36](#)

rearangr_v, [37](#)

see_df, [38](#)
see_file, [38](#)
see_idx, [39](#)
see_inside, [39](#)

unique_pos, [40](#)
until_stn1, [40](#)

v_to_df, [42](#)
val_replacer, [41](#)
vec_in_df, [41](#)
vlookup_df, [42](#)