Package 'edm1'

December 5, 2023

Title edm

Version 1.0

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

Description What the package does (one paragraph).

License GPL-3

description Set of tools in R

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.2.3

NeedsCompilation no

Maintainer First Last <first.last@example.com>

R topics documented:

`append_row` *append_row*

Description

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

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)
```


```
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

can_be_num *can_be_num*

Description

Return TRUE if a variable can be converted to a number and FALSE if not

Return TRUE if a variable can be converted to a number and FALSE if not

Usage

```
can_be_num(x)
```


```
can_be_num(x)
```


Arguments

x	is the input value

change_date *change_date*

Description

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

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"
)
```


```
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, (deault 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"

---

closest_date *closest_date*

Description

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

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
)

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, (deault 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
head	is the number of dates that will be returned (default set to NA so all dates in vec will be returned)
only	is can be changed to "+" or "-" to repectively only return the higher dates and the lower dates (default set to "both")

---

cost_and_taxes *cost and taxes*

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:

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
)

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

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_ = ","
)
```

```
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 (defult 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

new generated data in addition to saving it in the output

---

data_meshup *data_meshup*

Description

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

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 = "-"
)

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: c("*", c("d", "-", "e", "-", "f"), *"", c("a", "a1", "-", "b", "-", "c", "c1")"_")
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 *date_sort*

Description

Allow to ascendely or desendely 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 *days_from_month*

Description

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

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

Usage

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


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


Arguments

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

---

df_tuned *df_tuned*

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))

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")

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.

Allow to see the difference between two datasets and output it into an xlsx file

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
)

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

file_	is the file where the data is
sht	is the sheet where the data is
v_old_begin	is the corrdinate (row, column) where the data to be compared starts
v_old_end	is the same but for its end
v_new_begin	is the coordinates where the comparator data starts
v_new_end	is the same but for its end
df2	is optional, if the comparator dataset is directly a dataframe
overwrite	allow to overwrite differences is (set to T by default)
color_	is the color the differences will be outputed
pattern	is the pattern that will be added to the differences if overwritten is set to TRUE
output	is the name of the outputed xlsx (can be set to NA if no output)
new_val	if overwrite is TRUE, then the differences will be overwritten by the comparator data
pattern_only	will cover differences by pattern if overwritten is set to TRUE

---

file_rec *file_rec*

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

format_date *formate_date*

Description

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

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

Usage

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

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 "-")

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

insert_df *insert_df*

Description

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

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)

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
ins_loc	is a vector containg two parameters (row, column) of the begining for the insertion

letter_to_nb *letter_to_nb*

Description

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

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

Usage

```
letter_to_nb(letter)
```


```
letter_to_nb(letter)
```


Arguments

letter	is the letter (name of the column)

list_files *list_files*

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

match_n *match_n*

Description

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

Usage

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

Arguments

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

match_n2 *match_n2*

Description

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

Usage

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

Arguments

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

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", ...)

From a list containing vectors allow to generate a vector following this rule:

Usage

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

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

Arguments

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

Details

list(c("a", "b"), c("1", "2"), c("A", "Z", "E")) –> c("a1A", "a2A", "b1A", "b2A", "a1Z", ...)

nb_to_letter *nb_to_letter*

Description

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

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

Usage

```
nb_to_letter(x)

nb_to_letter(x)
```

Arguments

x	is the number of the column

pattern_generator *pattern_generator*

Description

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

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

Usage

```
pattern_generator(base_, from_, lngth, hmn = 1, after = 1)

pattern_generator(base_, from_, lngth, hmn = 1, after = 1)
```

Arguments

base_	is the pattern that will be kept
from_	is the vector from which the element of the varying part will be generated
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

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 containded in the vector that is beeing 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

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:

Usage

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

```
)

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

Arguments

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

Details

First case: Search for patterns strictly, it means that the searched pattern(s) will be matched only if the patterns containded in the vector that is beeing 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

---

pattern_tuning *pattern_tuning*

Description

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

Allow to tune a pattern very precisely

Usage

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

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

Arguments

pattrn	is the character that will be tuned
spe_nb	is the number of new character that will be replaced
spe_l	is the source vector from which the new characters will be replace old ones
exclude_type	is character that won't be replaced
hmn	is how many output the function will return
rg	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

---

see_df *see_df*

Description

Allow to return a datafame with TRUE cells where the condition entered are respected and FALSE where these are not

Allow to return a datafame with TRUE cells where the condition entered are respected and FALSE where these are not

Usage

```
see_df(df, condition_l, val_l)

see_df(df, condition_l, val_l)
```

Arguments

df	is the input dataframe
condition_l	is the vector of the possible conditions ("==", ">", "<", "!=", "%")
val_l	is a list with the vector of values related to condition_l (so the 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("&", "|", "&", "&", "&")

Examples

```
see_df(df, c("%%", "=="), list(c(2, 11), c(3)), list("|") will return all the values that
```

see_file *see_file*

Description

Allow to get the filename or its extension

Allow to get the filename or its extension

Usage

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

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

Arguments

string_	is the input string
index_ext	is the occurence 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

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)

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)

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

Arguments

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	alloaw to get files recursively If x is the return value, to see all the files name, position of the columns and possible sheet name associanted with, do the following: 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))
pattern	is a vector containin the file extension of the spreadsheets ("xlsx", "csv"...)
path	is the path where are located the files

Examples

```
print(x[(grep(unique_sep, x)[1]+1):length(x)]).
If you just want to see the columns do the following:
print(x[1:(grep(unique_sep, x) - 1)])
```

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

Description

Return the indexes of the first unique values from a vector

Usage

```
unique_pos(vec)

unique_pos(vec)
```

Arguments

`vec`	is the input vector

`until_stnl` *until_stnl*

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)

Maxes a vector to a chosen length

Usage

```
until_stnl(vec1, goal)

until_stnl(vec1, goal)
```

Arguments

`vec1`	is the input vector
`goal`	is the length to reach

Details

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)

`val_replacer` *val_replacer*

Description

Allow to replace value from dataframe to another one.

Allow to replace value from dataframe to another one.

Usage

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

val_replacer(df, val_replaced = F, 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` *vec_in_df*

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 containded in the dataframe

Allow to see if vectors are present in a dataframe

Usage

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

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

Details

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 containded in the dataframe

`vlookup_df` *vlookup_df*

Description

Alow to perform a vlookup on a dataframe

Alow to perform a vlookup on a dataframe

Usage

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

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)

Index