

# edm1

## Extended Data Management

It is an R package providing a set of tools to manage your data.

Documentation for each functions is written in edm1/man.

To see the documentation in your R shell (**recommended**):

1. `> library("devtools")`
2. `> devtools::load_all(path_to_edm_pkg)`
3. `> ?function_from_edm`

## PKG Requirements

1. stringr
2. openxlsx
3. stringi

## Installation

First, clone this repository `git clone https://github.com/iro0087/edm`

In your R prompt, in "edm/edm1" do the following:

1. `> build()`
2. `> install()`

## Documentation

### diff\_xlsx

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

@param file\_ is the file where the data is

@param sht is the sheet where the data is

@param v\_old\_begin is the corrdinate (row, column) where the data to be compared starts

@param v\_old\_end is the same but for its end

@param v\_new\_begin is the coordinates where the comparator data starts

@param v\_new\_end is the same but for its end

@param df2 is optional, if the comparator dataset is directly a dataframe

@param overwrite allow to overwrite differences is (set to T by default)

@param color\_ is the color the differences will be outputed

@param pattern is the pattern that will be added to the differences if overwritten is set to TRUE

@param output is the name of the outputed xlsx (can be set to NA if no output)

@param new\_val if overwrite is TRUE, then the differences will be overwritten by the comparator data

@param pattern\_only will cover differences by pattern if overwritten is set to TRUE

### **insert\_df**

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

@param df\_in is the dataframe that will be inserted

@param df\_ins is the dataset to be inserted ins\_loc is a vector containing two parameters (row, column) of the beginning for the insertion

### **pattern\_generator**

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

@param base\_ is the pattern that will be kept

@param from\_ is the vector from which the element of the varying part will be generated

@param hmn is how many of varying pattern from the same base will be created

@param 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\_tuning**

Allow to tune a pattern very precisely

@param patrn is the character that will be tuned

@param spe\_nb is the number of new character that will be replaced

@param spe\_l is the source vector from which the new characters will be replace old ones

@param exclude\_type is character that won't be replaced

@param hmn is how many output the function will return

@param 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

### **unique\_pos**

Return the indexes of the first unique values from a vector

@param vec is the input vector

### **can\_be\_num**

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

@param x is the input value

### **data\_gen**

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

@param type\_ is a vector for wich argument is a column, a column can be made of numbers ("number"), string ("string") or both ("mixed")

@param strt\_l is a vector containing for each column the row from which the data will begin to be generated

@param nb\_r is a vector containing for each column, the number of row full from generated data

@param output is the name of the output csv file

@param 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

@param 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

@param str\_source is the source (vector) from which the character creating random string are (default set to the occidental alphabet)

@param round\_l is a vector which, for each column containing number, associate a round value

@param sep\_ is the separator used to write data in the csv

@return new generated data in addition to saving it in the output

### **data\_meshup**

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

@param 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")"\_"))

@param cols is the colnames of the data generated in a csv

@param file\_ is the file to which the data will be outputed

@param sep\_ is the separator of the csv outputed

@param organisation is the way variables include themselves, for instance ,re-suming precedent example, if organisation=c(1, 0) so the data output will be:

d, a  
d, a1  
e, c  
f, c  
f, c1

@param unic\_sep1 is the unic separator between variables (default is "\_")

@param unic\_sep2 is the unic separator between datasets (default is "-")

### **letter\_to\_nb**

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

@param letter is the letter (name of the column)

### **nb\_to\_letter**

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

@param x is the number of the column

### **cost and taxes**

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:

@param qte is the quantity of elements

@param pu is the price of a single elements without taxes

@param prix\_ht is the duty-free price of the whole set of elements

@param tva is the percentage of all taxes

@param prix\_ttc is the price of all the elements with taxes

@param prix\_tva is the cost of all the taxes

@param pu\_ttc is the price of a single element taxes included

@param adjust is the discount percentage

@param prix\_d\_ht is the free-duty price of an element after discount

@param prix\_d\_ttc is the price with taxes of an element after discount

@param pu\_d is the price of a single element after discount and without taxes

@param 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 x-month-xxxx -> xx-xx-xxxx

### **formate\_date**

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

@param f\_dialect are the months from the language of which the month come

@param sentc is the date to convert

@param sep\_in is the separator of the dat input (default is "-")

@param sep\_out is the separator of the converted date (default is "-")

### **until\_stnl**

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)

@param vec1 is the input vector

@param goal is the length to reach

### **vlookup\_df**

Alow to perform a vlookup on a dataframe

@param df is the input dataframe

@param v\_id is a vector containing the ids

@param col\_id is the column that contains the ids (default is equal to 1)

@param included\_col\_id is if the result should return the col\_id (default set to yes)

## **multitud**

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", ...)
```

@param l is the list

@param sep\_ is the separator between elements (default is set to " " as you see in the example)

## **df\_tuned**

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))`

@param df is the input data.frame

@param val\_to\_stop is the vector containing the values to stop

@param index\_rc is the value for the jump (default set to NA so default will be first case)

@param included is if the values to stop has to be also returned in the vectors (defaultn set to "yes")

## **see\_df**

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

@param df is the input dataframe

@param condition\_l is the vector of the possible conditions ("`==`", ">", "<", "`!=`", "`%%`", "`%r`") (equal, greater than, lower than, not equal to, is divisible by, divides), you can put the same condition n times.

@param 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)

@param 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  $\rightarrow$  `c("&", "|", "&", "&", "&")`

@examples see `_df(df, c("%%", "=="), list(c(2, 11), c(3)), list("|"))` will return all the values that are divisible by 2 and 11 and all the values that are equal to 3 from the dataframe

### **days\_from\_month**

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

@param date\_\_ is the input date

@param sep\_\_ is the separator of the input date

### **vec\_in\_df**

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

@param df\_\_ is the input dataframe

@param vec\_l is a list the vectors

@param coeff\_\_ is the related coefficient of the vector

@param strt\_l is a vector containing the start position for each vector turns the number of row that contains vectors (multiple in a list), coeff and strt\_l may vary stinct is the value you are sure is not contained in the matrix

### **closest\_date**

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

@param date\_\_ is the input date

@param vec is a vector containing the dates to be compared to the input date

@param 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"

@param sep\_\_ is the separator for the input date

@param sep\_vec is the separator for the dates contained in vec

@param only is can be changed to “+” or “-” to repectively only return the higher dates and the lower dates (default set to “both”)

@param head is the number of dates that will be returned (default set to NA so all dates in vec will be returned)

### **change\_date**

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

@param date\_ is the input date

@param sep\_ is the date separator

@param day\_ is the day to add (can be negative)

@param month\_ is the month to add (can be negative)

@param year\_ is the year to add (can be negative)

@param hour\_ is the hour to add (can be negative)

@param min\_ is the minute to add (can be negative)

@param second\_ is the second to add (can be negative)

@param 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”

### **pattern\_gettr**

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 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

@param word\_ is the vector containing the patterns

@param vct is the vector being searched for patterns

@param 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

@param strict a vector containing the “strict” condition for each nth vector in word\_ (“strict” is the string to activate this option) @param btwn is a vector



containing the condition (“yes” to activate this option) meaning that if “yes”, all elements between two matched pattern in vct will be returned

@param 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

@param notatall is a string you are sure is not present in vct (default set to “###”)

@param btwn is a vector containing the condition (“yes” to activate this option) meaning that if “yes”, all elements between two matched pattern 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

@param 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

REGEX can also be used in the searched patterns

### **see\_inside**

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)

@param pattern is a vector containin the file extension of the spreadsheets (“xlsx”, “csv”...)

@param path is the path where are located the files

@param 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.

@param unique\_sep is a pattern that you know will never be in your input files

@param rec allow 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(x[1:(grep(unique\_sep, x) - 1)])

### **val\_replacer**

Allow to replace value from dataframe to another one.

@param df is the input dataframe

@param val\_replaced is a vector of the value(s) to be replaced

@param val\_replacor is the value that will replace val\_replaced

@param 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