# Package 'diffrprojects'

November 6, 2016

**Title** Projects for Text Version Comparison and Analytics in R

**Date** 2016-11-06

**Version** 0.1.14

**Description** Provides data structures and methods for measuring, coding, and analysing text within text corpora. The package allows for manual as well computer aided coding on character, token and text pair level.

**Depends** R (>= 3.0.0), stringb (>= 0.1.13), rtext (>= 0.1.20)

**License** MIT + file LICENSE

**LazyData** TRUE

**Imports** R6 (>= 2.1.2), hellno (>= 0.0.1), dplyr(>= 0.5.0), Rcpp (>= 0.12.6), stringdist (>= 0.9.4.1), RSQLite (>= 1.0.0), magrittr, stats

**Suggests** testthat, knitr, rmarkdown

**BugReports** https://github.com/petermeissner/diffrprojects/issues

**URL** https://github.com/petermeissner/diffrprojects

**RoxygenNote** 5.0.1

**LinkingTo** Rcpp

**NeedsCompilation** yes

**Author** Peter Meissner [aut, cre], Ulrich Sieberer [cph], University of Konstanz [cph]

**Maintainer** Peter Meissner <retep.meissner@gmail.com>

**Repository** CRAN

**Date/Publication** 2016-11-06 22:32:04

## R topics documented:

---

`as.data.frame.alignment_data_list` *as.data.frame method for for named lists of data.frames*

---

## Description

as.data.frame method for for named lists of data.frames

## Usage

```
## S3 method for class 'alignment_data_list'
as.data.frame(x, row.names = NULL,
  optional = FALSE, ...)
```

## Arguments

| | |
|---|---|
| `x` | any R object. |
| `row.names` | `NULL` or a character vector giving the row names for the data frame. Missing values are not allowed. |
| `optional` | logical. If `TRUE`, setting row names and converting column names (to syntactic names: see `make.names`) is optional. Note that all of R's **base** package `as.data.frame()` methods use `optional` only for column names treatment, basically with the meaning of `data.frame(*, check.names = !optional)`. |
| `...` | additional arguments to be passed to or from methods. |

## as.data.frame.alignment_list

*as.data.frame method for for named lists of data.frames*

### Description

as.data.frame method for for named lists of data.frames

### Usage

```
## S3 method for class 'alignment_list'
as.data.frame(x, row.names = NULL,
  optional = FALSE, ...)
```

### Arguments

| | |
|---|---|
| x | any R object. |
| row.names | NULL or a character vector giving the row names for the data frame. Missing values are not allowed. |
| optional | logical. If TRUE, setting row names and converting column names (to syntactic names: see make.names) is optional. Note that all of R's **base** package as.data.frame() methods use optional only for column names treatment, basically with the meaning of data.frame(*, check.names = !optional). |
| ... | additional arguments to be passed to or from methods. |

## as.data.frame.named_df_list

*as.data.frame method for for named lists of data.frames*

### Description

as.data.frame method for for named lists of data.frames

### Usage

```
## S3 method for class 'named_df_list'
as.data.frame(x, row.names = NULL, optional = FALSE,
  dfnamevar = "name", ...)
```

### Arguments

| | |
|---|---|
| x | any R object. |
| row.names | NULL or a character vector giving the row names for the data frame. Missing values are not allowed. |
| optional | logical. If TRUE, setting row names and converting column names (to syntactic names: see make.names) is optional. Note that all of R's **base** package as.data.frame() methods use optional only for column names treatment, basically with the meaning of data.frame(*, check.names = !optional). |
| dfnamevar | in which variable should list item names be saved |
| ... | additional arguments to be passed to or from methods. |

---

`choose_options` *(choose from a number of pre-sorted options) takes a vector pair of toki1 / toki2 and a vector pair of res_token_i_1 / res_token_i_2 and chooses so that each 1st and exh 2nd value only is used where res_token_i_x identiefies already used items.*

---

### Description

(choose from a number of pre-sorted options) takes a vector pair of toki1 / toki2 and a vector pair of res_token_i_1 / res_token_i_2 and chooses so that each 1st and exh 2nd value only is used where res_token_i_x identiefies already used items.

### Usage

```
choose_options(toki1, toki2, res_token_i_1, res_token_i_2)
```

### Arguments

| | |
|---|---|
| toki1 | first number of number pair to choose from |
| toki2 | second number of number pair to choose from |
| res_token_i_1 | already used first numbers |
| res_token_i_2 | already used second numbers // @keywords internal |

`diffrproject` *class for diffrproject*

## Description

class for diffrproject

## Usage

```
diffrproject
```

## Format

`R6Class` creator object.

## Value

Object of `diffrproject`

## The diffrprojects class family

Diffrproject consists of an set of R6 classes that are conencted by inheritance. Each class handles a different set of functionalities that are modular.

**R6_rtext_extended** A class that has nothing to do per se with diffrprojects. It merely adds some basic features to the base R6 class (debugging, hashing, getting fields and handling warnings and messages as well as listing content). This class is imported from rtext package

**dp_base** [inherits from rtext::R6_rtext_extended] This class forms the foundation of all diffrpojects (dp_xxx) classes by implementing data fields for meta data, texts, data on texts, links between texts, alignment of text tokens, and data on the alignment of text tokens. Furthermore it implements methods add, delete, code, and link texts or to aggregate text data on text token level.

**dp_loadsave** [inherits from dp_base] This class allows for loading and saving diffrprojects from and to Rdata files.

**dp_export** [inherits from dp_loadsave] This class provides methods for exporting and importing to and from RSQLite.

**dp_align** [inherits from dp_export] This is one of the workhorses of diffrprojects. The methods of this class allow for adding, deleting or computing alignments between text tokens (e.g. words or lines or sentences or characters or paragraphs, or some other way to split text into chunks). Furthermore it allows to also assign data to individual alignments (a connection beween two token of text from different text versions).

**dp_inherit** [inherits from dp_align] The text_data_inherit method added by this class allows to copy text data from one token of a text version to another token of another text version channeled through aligments with zero distance. Conflicting codings (a text might have multiple codings stemming from several links and from direct coding of the text) are resolved by the fact that text codings are accompanied by a hierarchy level that defaults to zero and gets decreased by one every time the coding is inherited by a token.

**diffrproject** [inherits from dp_inherit] Just a wrapper inheriting from dp_inherit to have a less technical name at the end of the inheritance chain.

## Examples

```
## Creating a Diffrprojects Instance

# To create a diffrproject we use the diffrproject creator object -
# its simply an object with an function that knows how to create a project.

# Creating a project looks like this:

library(diffrprojects)
dp <- diffrproject$new()


# Et viola - we created a first, for now empty, project that we will
# use throughout the tutorial.




## Some Help Please

# To get a better idea about what this thing called *diffrproject* really is
# you can consult its help page which gives a broad overview over its
# capabilities:

?diffrproject


# Another way is to call the ls() method. This will present us with a
# data frame listing all fields where data is stored and all the methods
# (aka object specific functions) of our diffrprojects instance.
# Those methods and fields located in *private* are not for the user
# to mess around with while non-private (*self* aka public) data fields
# can be read by the user and public methods can be triggered by the
# user to manipulate the data or retrieve data in a specific format.


dp$ls()


# The base R class() function furthermore reveals from which classes the
# diffrproject class inherits:

class(dp)
```

```
## Adding Texts to Projects

# Our diffrproject (`dp`) has one method called `text_add()` that allows to
# add texts to the project. Basically the method can be used in three
# different flavors: adding character vectors, adding texts stored on disk,
# or by adding rtext objects (see rtext package:
# https://CRAN.R-project.org/package=rtext; rtext objects are the way
# individual texts are represented within diffrprojects).
# For each of these used cases there is one option:
# `text`, `text_file`, `rtext`; respectively.

# Below are shown examples using each of these methods:


#  **adding text files**

test_file1 <- stringb:::test_file("rc_1_ch1.txt")
test_file2 <- stringb:::test_file("rc_2_ch1.txt")
dp$text_add(text_file = c(test_file1, test_file2) )


# **adding rtext objects**

test_file <- stringb:::test_file("rc_1_ch1.txt")
rt <- rtext$new( text_file = test_file)
dp$text_add(rtext = rt)


# **adding character vectors**

test_file1 <- stringb:::test_file("rc_1_ch1.txt")
test_file2 <- stringb:::test_file("rc_2_ch1.txt")
cv <- ""
cv[1] <- text_read(test_file1, NULL)
cv[2] <- text_read(test_file2, NULL)
dp$text_add(text = cv)

# In the last case make sure to put each text in one separate line.
# Functions like readLines() or text_read() read in texts such that
# each line corresponds to one element in a character vector. With e.g.
# text_read()'s tokenize parameter to NULL the text will be read in as one
# long string.




## Piping Methods

# Now is a good time to mention a feature of diffrprojects that comes in
# handy: All functions that do not explicitly extract data
# (those usually have some 'get' as part of their name) do return the
```

```
# object itself so that one can pipe together a series of method calls.

# Consider the following example where we initiate a new diffrprojects
# instance and add two texts in just one pipe:

dp <-
  diffrproject$
  new()$
  text_add(text_version_1, name = "version1")$
  text_add(text_version_2, name = "version2")

length(dp$text)




## Getting Infos About Texts

# If we want to get some general overview about the texts gathered in our
# project, we can use the text_meta_data() method to do so.
# The method has no parameters and returns a data.frame with several
# variables informing us about its source, length, encoding used for
# storage, and its name.


dp$text_meta_data()




## Showing Text

# If you want to have a look at your texts you may do so by using the
# text's own text_show methods. Per default this method only shows the
# first 500 characters, but it can be set to higher numbers as well.


dp$text$version1$text_show(length=1000)
dp$text$version2$text_show(length=1000)




## Getting And Setting Infos About the Project

# Similar to the text_meta_data() method we can access the projects
# meta data via data fields meta and options. But contrary to the
# text_meta_data() method that gathers data from all the texts within the
# project and does not allow for manipulation of the data, the data
# fields allow reading and writing.
```

```
# First let us have a look and thereafter turn off the message
# notification service:

# **getting data fields**

dp$options


# **setting data fields**

dp$options$verbose <- FALSE


# (note, ask is deprecated and only remains for compatibility
# reasons but has no function anymore)

# Now it's time to have a look at the projects meta data.
# It tells us when the project was created, which path to use for
# SQLite exports, which path to use for saving data as in RData
# format and what is the projects id. The id is a hash of a time stamp
# as well as session information which should ensure uniqueness across
# space and time.

# All these values can be manipulated by the user to her liking.


dp$meta

dp$meta$file_path = "./diffrproject.RData"







## Deleting Texts

# Of cause we can not only add texts but delete them from the project as
# well. For this purpose there is the text_delete() method.

# Let's just add two texts and delete one by providing its index number and
# the second by providing its name to the text_delete() method.


dp$text_add(text = "nonesense", "n1")
dp$text_add(text = "nonesense", "n2")

dp$text_delete(3)
dp$text_delete("n2")

length(dp$text)
```

```
names(dp$text)




## Defining Relationships Between Texts: Linking

# The purpose of diffrprojects is to enable data collection on the
# difference of texts. Having filled a project with various texts,
# there are endless possibilities to form pairs of text for comparison
# and change measurement - where endless actually is equal to: $n^2-n$.

# Linking can be done via the text_link method which accepts either
# index numbers or text names for its from and to arguments
# (a third argument delete will delete a specified link if set to TRUE).


dp$text_link(from = 1, to = 2)
dp$text_link(from = 1, to = 2, delete = TRUE)


# If no arguments are specified, text_link will link the first text to
# the second, the third to the fourth, the fourth to the fifths and so on.


dp$text_link()



# To get an idea of what links are currently specified, we can
# directly access the link data field or/and ask R to transform the
# list found there into a data.frame.


dp$link

dp$link %>% as.data.frame()







## Aligning Texts and Measuring Change

# At the heart of each diffrproject lies the text_align method.
```

```
# This method compares two texts and tries to align parts
# of one text with parts of the other text. The first two
# arguments (`t1` and `t2`) are for specifying which pair
# of texts to compare - if left as-is, all text pairs that
# are specified within the link data field will be aligned.

# Text parts are arbitrary character spans defined by the
# `tokenizer` argument. This argument expects a function splitting
# text into a token data.frame. If the tokenizer argument
# is left as-is, it will default to text_tokenize_lines function
# from the stringb package.

# Text tokens can be pre-processed before alignment. The `clean`
# argument allows to hand over a function tranforming a charactr
# vector of text tokens into their clean counterparts.

# The `ignore` arguments expects a function that is able to
# transform a character vector of tokens into a logical vector
# of same length, indicating which tokens to ignore throughout
# the alignment process and which to consider.

# The next argument - `distance` - specifies which distance
# metrics to use to calculate distances between strings.

# Since the text_align method basically is a wrapper around
# diff_align you can get more information via `?diff_align`
# and since again diff_align is a wrapper around stringdist
# from the stringdist package `?stringdist::stringdist` and
# also ``?stringdist::`stringdist-metrics` `` will provide
# further insights about possible metrics and how to use the
# rest of the arguments to text_align (these are passed through
# to stringdist).

# Let's have an example using the Levenshtein distance to
# calculate distances between tokens (lines per default).
# Furthermore we allow the distance between two aligned tokens
# to be as large as 15. Tokens which do not find a partner
# below that distance are considered to have been deleted
# or respectively inserted. Tokens which find a partner with
# a non-zero distance which is not above the threshhold are
# considered changes - transformations of one token into the other.

# The following shows the resulting list of alignment data.frames.


dp$text_align(distance = "lv", maxDist = 15)

dp$alignment



# To measure the change between those two texts we can e.g. aggregate
# the distances by change type:
```

```
sum_up_changes <- function(x){
x %>%
dplyr::group_by(type) %>%
dplyr::summarise(sum_of_change = sum(distance))
}

lapply( dp$alignment, sum_up_changes)
```

```
## Coding Texts

# Now let us put some data into our diffrproject.

# The most basic method to do so is simply called text_code.
# Text_code takes up
# to five arguments (the first three are mandatory), where one
# specifies the text to be coded (`text`, either by index
# number or by name), how the variable to store the information
# is called (`x`), and the index number or a vector of those
# indicating which characters of the text should be coded.
# The last two parameters are optional and specify which value
# the variable should hold (`val`) and at which hierarchy
# level the coding is placed (`hl`, higher or equal hierarchy
# levels will overwrite existing codings of lower hierarchy
# level for the same text, character span, and variable).


dp$text_code(text = 1, x = "start", i=1:5, val = TRUE, hl = 0)
dp$text_code(text = "version2", x = "start", i=1:5, val = TRUE, hl = 0)


# The text_code method is quite verbose and in most cases more suited
# to be accessed by a machine or algorithm than by a human.
# Therefore, there are three other methods to code text:
# text_code_regex, text_code_alignment_token,
# text_code_alignment_token_regex.

# The text_code_regex method allows to search for text patterns and
# code a whole pattern instead of assigning codes character by
# character - the `i` argument of text_code gets replaced by a
# `pattern` argument. The in addition further arguments can be
# passed to the pattern search functions via `...` - see e.g.
# `?grep` for possible further arguments and
# https://stat.ethz.ch/R-manual/R-devel/library/base/html/regex.html for a
# description of regular expressions in R.

# In this example we are searching for the word *"it"* in text 1 and code
```

```
# each instance.


dp$text_code_regex(text = 1, x = "it", pattern = "\\bit\\b", ignore.case=TRUE)


# Another variant of coding text is by using alignment tokens.
# Having alignment data availible, this allows for selecting:
# link, alignment and text while the other arguments from above stay the
# same.



# having a look at alignment number 4

dp$alignment[[1]][4,]


# coding text connected by alignment number 4

dp$text_code_alignment_token(
  link        = 1,
  alignment_i = 4,
  text1       = TRUE,
  text2       = TRUE,
  x   = "token_coding",
  val = 4,
  hl  = 0
)







## Getting Text Codings

# The most basic way to get text data is to use the text_data method.
# This method will go through all or only selected texts, gather all
# the data stored there and put it into a neat data.frame where name
# identifies the text from which the data comes per name, char informs
# us about the character that was coded, and i refers to the characters
# position within the text. All other variables hold the data we added
# during the examples above.


dp$text_data(text = 1) %>% head()
```

```
## Aggregating Text Codings

# The usage of text_data has its merits but often one is more
# interested in text data aggregated to a specific level.
# The following three aggregation functions offer a solution
# to this problem: tokenize_text_data_lines, tokenize_text_data_words,
# and tokenize_text_data_regex. These three methods make use
# of the similiary named methods provided by the rtext package.

# One important thing to keep in mind is that using these methods
# implies aggregating several data values on character level
# into one data value at token level. Therefore there has
# to be some aggregation function to be involved. The default
# is to use the value that occurs most often on character
# level, if more than one distinct values occur more than
# once the first is choosen.

# The aggregation function can be changed to whatever function the
# user seems appropriate by passing it to `aggregate_function`
# - as long as it
# reduces a vector of values into a vector with only one value.

# The `join` argument allows to decide how text and data are
# joined
# into the resulting data.frame - left: all token, right: all data, full:
# token with or without data and data with or without token.


dp$tokenize_text_data_lines(
text = 1,
join = "right",
aggregate_function =
function(x){
paste(x[1:3], collapse = ",")
}
)







## Text Coding Inheritence

# Having aligned two texts via token pairs another functionality of
# diffrprojects becomes availible: text coding inheritance via no-change
# tokens. This means that text codings can get copied to those tokens they
# are aligned with, given that they are considered the same - i.e. the
# distance equals zero and the change type therefore is no-change.
```

```
# To show this feature we use the text_inherit method and we will
# start with a fresh example. A new project with two texts. The first text
# gets some codings, then they are aligned, and in a last step codings are
# transfered from one text to the other via the text_data_inherit method.



dp <-
diffrproject$new()$
text_add(text_version_1)$
text_add(text_version_2)$
text_code_regex(
text    = 1,
x       = "test1",
pattern = "This part.*?change",
val = "inherited"
)$
text_code_regex(
text    = 1,
x       = "test2",
pattern = "This part.*?change",
val = "inherited"
)

dp$tokenize_text_data_lines(1)


dp$
text_link()$
text_align()$
text_data_inherit(
link      = 1,
direction = "forward"
)

dp$tokenize_text_data_lines(2)




## Saving and Loading Projects

# Diffrprojects also allow for storing and loading project to and
# from disk.

# note, uncomment code lines to run

# save to file
# dp$save(file = "dp_save.RData")

# remove object
# rm(dp)
```

```
# create new object and load saved data into new object
# dp <- diffrproject$new()
# dp$load("dp_save.RData")
# dp$tokenize_text_data_lines(2)
```

---

`diff_align` *algining texts*

---

## Description

Function aligns two texts side by side as a data.frame with change type and distance given as well

## Usage

```
diff_align(text1 = NULL, text2 = NULL, tokenizer = NULL, ignore = NULL,
  clean = NULL, distance = c("lv", "osa", "dl", "hamming", "lcs", "qgram",
  "cosine", "jaccard", "jw", "soundex"), useBytes = FALSE, weight = c(d = 1,
  i = 1, s = 1, t = 1), maxDist = 0, q = 1, p = 0,
  nthread = getOption("sd_num_thread"), verbose = TRUE, ...)
```

## Arguments

| | |
|---|---|
| `text1` | first text |
| `text2` | second text |
| `tokenizer` | defaults to NULL which will trigger linewise tokenization; accepts a function that turns a text into a token data frame; a token data frame has at least three columns: from (first character of token), to (last character of token) token (the token) |
| `ignore` | defaults to NULL which means that nothing is ignored; function that accepts a token data frame (see above) and returns a possibly subseted data frame of hte same form |
| `clean` | defaults to NULL which means that nothing cleaned; accepts a function that takes a vector of tokens and returns a vector of same length - potentially clean up |
| `distance` | defaults to Levenshtein ("lv"); see amatch, stringdist-metrics, stringdist |
| `useBytes` | Perform byte-wise comparison, see `stringdist-encoding`. |

| | |
|---|---|
| weight | For method='osa' or 'dl', the penalty for deletion, insertion, substitution and transposition, in that order. When method='lv', the penalty for transposition is ignored. When method='jw', the weights associated with characters of a, characters from b and the transposition weight, in that order. Weights must be positive and not exceed 1. weight is ignored completely when method='hamming', 'qgram', 'cosine', 'Jaccard', 'lcs', or soundex. |
| maxDist | [DEPRECATED AND WILL BE REMOVED|2016] Currently kept for backward compatibility. It does not offer any speed gain. (In fact, it currently slows things down when set to anything different from Inf). |
| q | Size of the $q$-gram; must be nonnegative. Only applies to method='qgram', 'jaccard' or 'cosine'. |
| p | Penalty factor for Jaro-Winkler distance. The valid range for p is 0 <= p <= 0.25. If p=0 (default), the Jaro-distance is returned. Applies only to method='jw'. |
| nthread | Maximum number of threads to use. By default, a sensible number of threads is chosen, see stringdist-parallelization. |
| verbose | should function report on its doings via messages or not |
| ... | further arguments passed through to distance function |

## Value

dataframe with tokens aligned according to distance

---

dp_align *class for dp_align*

---

## Description

class for dp_align

## Usage

```
dp_align
```

## Format

R6Class object.

## Value

Object of dp_align

## See Also

diffrproject

## dp_base *class for dp_base*

### Description

class for dp_base

### Usage

```
dp_base
```

### Format

R6Class object.

### Value

Object of dp_base

### See Also

diffrproject

## dp_export *R6 class - linking text and data*

### Description

R6 class - linking text and data

### Usage

```
dp_export
```

### Format

R6Class object.

### Value

Object of R6Class

### See Also

diffrproject

## dp_inherit *class for dp_inherit*

### Description

class for dp_inherit

### Usage

```
dp_inherit
```

### Format

R6Class object.

### Value

Object of dp_align

### See Also

diffrproject

## dp_loadsave *class for dp_base*

### Description

class for dp_base

### Usage

```
dp_loadsave
```

### Format

R6Class object.

### Value

Object of dp_loadsave

### See Also

diffrproject

## dp_text_base_data *function providing basic information on texts within diffrproject*

### Description

function providing basic information on texts within diffrproject

### Usage

```
dp_text_base_data(dp)
```

### Arguments

| | |
|---|---|
| dp | a diffrproject object |

## dummyimport *imports*

### Description

imports

### Usage

```
dummyimport()
```

## get_private *accessing private from R6 object*

### Description

accessing private from R6 object

### Usage

```
get_private(x)
```

### Arguments

| | |
|---|---|
| x | R6 object to access private from |

### Source

http://stackoverflow.com/a/38578080/1144966

`push_text_char_data` *push char_data of one rtext objet to another*

## Description

Function that takes a rtext object pulls specific char_data from it and pushes this information to another rtext object.

## Usage

```
push_text_char_data(from_text = NULL, to_text = NULL, from_token = NULL,
  to_token = NULL, from_i = NULL, to_i = NULL, x = NULL, warn = TRUE)
```

## Arguments

| | |
|---|---|
| `from_text` | text to pull data from |
| `to_text` | text to push data to |
| `from_token` | token of text to pull data from (e.g.: data.frame(from=1, to=4)) |
| `to_token` | token of text to push data to (e.g.: data.frame(from=1, to=4)) |
| `from_i` | index of characters to pull data from |
| `to_i` | index of characters to push data to |
| `x` | name of the char_data variable to pull and push - defaults to NULL which will result in cycling through all availible variables |
| `warn` | should function warn about non-uniform pull values (those will not be pushed to the other text) |

## Details

Note, that this is an intelligent function.

It will e.g. always decrease the hierarchy level (hl) found when pulling and decrease it before pushing it forward therewith allowing that already present coding might take priority over those pushed.

Furthermore, the function will only push values if the pulled values are all the same. Since, character index lengths that are used for pulling and pushing might differ in length there is no straight forward rule to translate non uniform value sequences in value sequnces of differing length. Note, that of cause the values might differ between char_data variables but not within. In case of non-uniformity the function will simply do nothing.

## sort_alignment *function sorting alignment data according to token index*

### Description

function sorting alignment data according to token index

### Usage

```
sort_alignment(x, ti1 = NULL, ti2 = NULL, first = TRUE)
```

### Arguments

| | |
|---|---|
| x | data.frame to be sorted |
| ti1 | either NULL (default): first column of x is used as first token index for sorting; a character vector specifying the column to be used as first token index; or a numeric vector of length nrow(x) to be use as first token index |
| ti2 | either NULL (default): second column of x is used as second token index for sorting; a character vector specifying the column to be used as second token index; or a numeric vector of length nrow(x) to be use as second token index |
| first | should first text or second text be given priority |

## text_version_1 *text_version_1 a first version of a text*

### Description

text_version_1 a first version of a text

### Usage

```
text_version_1
```

### Format

An object of class `character` of length 1.

### Source

Source of Text: Diff. (2014, August 26). In Wikipedia, The Free Encyclopedia. Retrieved 10:14, September 24, 2014, from http://en.wikipedia.org/w/index.php?title=Diff&oldid=622929855

## text_version_2 *text_version_2 a second version of a text*

### Description

text_version_2 a second version of a text

### Usage

```
text_version_2
```

### Format

An object of class `character` of length 1.

### Source

Source of Text: Diff. (2014, August 26). In Wikipedia, The Free Encyclopedia. Retrieved 10:14, September 24, 2014, from http://en.wikipedia.org/w/index.php?title=Diff&oldid=622929855

## write_numerous_parts_to_table *function writing numerous parts of table to database*

### Description

function writing numerous parts of table to database

### Usage

```
write_numerous_parts_to_table(x, con, table_name, meta = data.frame())
```

### Arguments

| | |
|---|---|
| `x` | parts to be written |
| `con` | connection to database |
| `table_name` | of the table |
| `meta` | additional information to be attachesd to table parts |

# Index