

# Package: sumup (via r-universe)

May 26, 2026

**Title** Utilizing Automated Text Analysis to Support Interpretation of Narrative Feedback

**Version** 1.0.2.9000

**Description** Combine topic modeling and sentiment analysis to identify individual students' gaps, and highlight their strengths and weaknesses across predefined competency domains and professional activities.

**License** GPL-3 | file LICENSE

**Encoding** UTF-8

**Roxygen** list(markdown = TRUE)

**RoxygenNote** 7.3.2

**Imports** data.table, dplyr, jsonlite, magrittr, rlang, udpipe, reticulate, stats, stringr, tibble, tidyr, tidytext, topicmodels, utils

**Suggests** matrixStats, reshape2, sentimentr, slam, stopwords, stringi, textstem, tokenizers

**LazyData** true

**Depends** R (>= 3.5)

**Config/pak/sysreqs** libgs10-dev libicu-dev libpng-dev libxml2-dev python3

**Repository** <https://jmoonen.r-universe.dev>

**Date/Publication** 2026-02-25 11:19:40 UTC

**RemoteUrl** <https://github.com/jmoonen/sumup>

**RemoteRef** HEAD

**RemoteSha** 077df409c1a1ba7ada821470d89cad9af4d1e2eb

## Contents

append_stopwords . . . . .	2
check_all_settings . . . . .	3
check_setting . . . . .	3

clean_text . . . . .	4
correct_text . . . . .	4
create_dataset_narratives . . . . .	5
create_output . . . . .	5
default_stopwords . . . . .	6
example_data . . . . .	6
get_corrections . . . . .	8
get_polarity . . . . .	8
lemmatize . . . . .	9
obtain_word_counts . . . . .	10
replace_abbr . . . . .	11
run_sumup . . . . .	11
sentiment_analysis_sentimentr . . . . .	12
set_default_settings . . . . .	13
text_clean . . . . .	14
tidy_text . . . . .	15
topic_classification . . . . .	16
topic_modeling . . . . .	18
update_educational_framework_settings . . . . .	19
update_setting . . . . .	20

## Index 21

---

append_stopwords	<i>Append Stopwords</i>
------------------	-------------------------

---

### Description

This function is used to append additional stopwords to an existing list.

### Usage

```
append_stopwords(data_stopwords, stopwords_to_append)
```

### Arguments

`data_stopwords` A character vector containing the initial set of stopwords to which more stopwords can be appended.

`stopwords_to_append`  
A character vector containing additional stopwords to be appended to the existing `data_stopwords` vector.

### Details

The `append_stopwords` function takes an existing list of stopwords (`data_stopwords`) and appends a new set of stopwords (`stopwords_to_append`) to it. Additionally, stopwords from the `stopwords` package (using the "dutch" language option) are also appended to the list. The final list is then returned with duplicate stopwords removed by calling `unique()`.

**Value**

Returns a character vector of unique stopwords after appending the specified stopwords and additional stopwords from the stopwords package.

---

check_all_settings	<i>Check all setting</i>
--------------------	--------------------------

---

**Description**

Check for every setting the type and value

**Usage**

```
check_all_settings(settings)
```

**Arguments**

settings	A current list of settings (list)
----------	-----------------------------------

**Value**

Either an error or TRUE

---

check_setting	<i>Check setting</i>
---------------	----------------------

---

**Description**

Check the value of one particular setting

**Usage**

```
check_setting(settings, setting_name, setting_value)
```

**Arguments**

settings	A current list of settings (list)
setting_name	The name of the setting that needs updating (string)
setting_value	The value of the setting

**Value**

TRUE if correct, FALSE if incorrect

---

clean_text	<i>Clean Text</i>
------------	-------------------

---

**Description**

Cleans and standardizes text by handling HTML tags, punctuation, spacing, and abbreviations.

**Usage**

```
clean_text(input_text, corrections)
```

**Arguments**

input_text	A character vector containing the text to be cleaned.
corrections	A data frame containing typo corrections.

**Value**

A cleaned character vector.

---

correct_text	<i>Correct Text</i>
--------------	---------------------

---

**Description**

Applies typo corrections to an input text using a given correction table.

**Usage**

```
correct_text(input_text, corrections)
```

**Arguments**

input_text	A character vector containing the text to be corrected.
corrections	A data frame containing two columns: typoEdit (pattern) and replacement (correction).

**Value**

A character vector with corrected text.

---

create\_dataset\_narratives  
*Create Narratives Dataset*

---

**Description**

Restructures a dataset by normalizing and transforming its textual content.

**Usage**

```
create_dataset_narratives(data)
```

**Arguments**

data                    A data.table or data.frame containing narrative data.

**Value**

A structured data.table with cleaned text.

---

create\_output            *Create Output for Sum Up*

---

**Description**

Processes sentence-level Sum Up's output, assigns sentences to topics, calculates topic strengths, and organizes data into structured feedback categories.

**Usage**

```
create_output(sentence_complete_output, sentences_lda, settings)
```

**Arguments**

sentence\_complete\_output            A data frame containing sentence-level topic polarity scores.

sentences\_lda            An LDA model object containing topic-term distributions.

settings            A list of settings containing parameters such as the number of topics, threshold values, scoring scale, and output format.

**Details**

This function processes Sum Up's output to assign sentences to topics based on a probability threshold. It calculates the topic strength based on sentence polarity and recency, categorizes topics by feedback quantity and quality, and includes unassigned feedback in a separate category.

**Value**

A structured list containing topic-wise feedback, strength, and quality assessments. If `settings$output_json` is TRUE, returns a JSON string.

---

<code>default_stopwords</code>	<i>Load Stopwords</i>
--------------------------------	-----------------------

---

**Description**

Load stopwords from a file

**Usage**

```
default_stopwords(stopwords_file)
```

**Arguments**

`stopwords_file` A string representing the path to a file containing a list of stopwords, with one word per line. This file is expected to have tab-separated values with the stopwords in the first column.

**Details**

The `default_stopwords` function loads a list of stopwords from a specified file, with each stopword being listed on a new line. It returns a character vector of these stopwords.

**Value**

Returns a character vector of stopwords read from the specified file.

---

<code>example_data</code>	<i>Example example_data</i>
---------------------------	-----------------------------

---

**Description**

`example_data` contains an example of narrative feedback data and scores of formative assessments from a student's portfolio in the Master education in Medicine from a Dutch university, translated to English.

**Usage**

```
example_data
```

**Format**

A data frame with 6 rows and 15 columns:

**submissionid** ID of the assessment form from which the data originates (NB: as the data is gathered per competency, the same submissionid can be used for multiple rows of data).

**formid** ID of the type of assessment form (NB: set to 1 if not applicable).

**templatename** Name of the type of assessment form (NB: set to same 'dummy' name for all rows, if not applicable).

**assistant** Identifier of the student. Needed when data from multiple portfolios is loaded. Set all to same value if not applicable.

**authority** Identifier of the assessor. NB: set all to same value if not applicable.

**specialty** Textual identifier of the specialty from which the assessment originates (NB: set to same 'dummy' name for all rows if not applicable).

**hospital** Textual identifier of the hospital from which the assessment originates (NB: set to same 'dummy' name for all rows if not applicable).

**portfolioid** Identifier of the sub-portfolio, e.g. when using rotations/clerkships. Define the IDs in the settings. NB: set all to 1 if not applicable.

**competencyid** Identifier of the competency used in the educational framework. Define the IDs in the settings. NB: set all to 1 if not applicable.

**sterk** Narrative feedback in case it is described in a predefined area for 'strengths' in the performance of the student.

**verbeter** Narrative feedback in case it is described in a predefined area for 'improvement areas' in the performance of the student.

**feedback** Narrative feedback in case it is NOT described in a predefined area for 'strengths' or 'improvement area'.

**score** Numeric score on the same assessment form. The scale can be set in settings. NB: set to NA or NULL if not applicable.

**datereferenced** Date of the assessment in format yyyy-mm-dd.

**zorgpunten** Comma-separated list of competencies that are marked as 'concerns' in the performance of the student. Make sure the names of competencies match the values in setting 'competencies'. NB: leave empty if not applicable.

**Source**

Example data for package documentation

---

get_corrections	<i>Load Corrections</i>
-----------------	-------------------------

---

**Description**

Reads a typo correction file and prepares it for use in text cleaning.

**Usage**

```
get_corrections(corrections_file)
```

**Arguments**

corrections_file	Path to a CSV file with typo corrections.
------------------	---

**Value**

A data frame containing typoEdit (pattern) and replacement (correction).

---

get_polarity	<i>Get Polarity from Grasp</i>
--------------	--------------------------------

---

**Description**

This function computes the polarity score for a given sentence using the Grasp sentiment analysis model. It applies specific logic based on the comment type (e.g., improvement suggestions) and language (Dutch, English, German, or French).

**Usage**

```
get_polarity(grasp, sentence, comment_type, language)
```

**Arguments**

grasp	The imported Grasp model, which is used to compute sentiment polarity.
sentence	A character string representing the sentence for which sentiment polarity needs to be calculated.
comment_type	A string that indicates the type of feedback (e.g., "verbeter" for improvement). This influences sentiment adjustment in Dutch language.
language	A string indicating the language of the sentence ("nl" for Dutch, "en" for English, "de" for German, "fr" for French).

**Details**

For Dutch sentences, the function applies specific transformations (such as removing certain words like "naar" and "gemaakt") before calculating sentiment. It also handles feedback sentences marked as "verbeter" (improvement suggestions) by adding the word "suboptimal" to adjust the sentiment analysis for improvement-type feedback.

**Value**

A numeric value representing the sentiment polarity of the sentence. Positive values indicate positive sentiment, negative values indicate negative sentiment, and zero indicates neutral sentiment.

---

lemmatize	<i>Lemmatize Sentences Using a UDPipe Model</i>
-----------	---

---

**Description**

This function processes a dataset of sentences using a UDPipe model to perform lemmatization, applies corrections to the lemmas, and associates metadata (e.g., submission ID, document) with the processed sentences. If UDPipe is not available, package NLP is used.

**Usage**

```
lemmatize(
  sentences,
  udpiper_model_file,
  corrections_file,
  language,
  use_seeds = TRUE
)
```

**Arguments**

sentences	A data frame containing sentences, with at least a sentenceid, sentence, document, and submissionid column.
udpipe_model_file	A character string representing the path to the UDPipe model file used for annotation.
corrections_file	A character string representing the path to a CSV file containing corrections to be applied to the lemmas.
language	A character string representing the language of the dataset ('nl', 'en', 'de' or 'fr')
use_seeds	A logical value indicating whether to use the seeds (e.g. of an educational framework)

**Details**

This function loads a UDPipe model, annotates the input sentences, corrects lemmas based on a provided corrections file, and optionally filters by noun tokens. It returns a data frame with the sentence-level annotations.

**Value**

A data frame containing the lemmatized sentences with the following columns:

- `doc_id`: Document identifier.
- `lemma`: The lemmatized form of each word.
- `upos`: Universal part-of-speech tag.
- `sentenceid`: The sentence ID for the sentence from which the lemma was extracted.
- `document`: Document associated with the sentence.
- `submissionid`: Submission ID associated with the sentence.

---

<code>obtain_word_counts</code>	<i>Obtain Word Counts</i>
---------------------------------	---------------------------

---

**Description**

This function calculates the word counts for each document in the annotated dataset, excluding stopwords. It unites the annotated dataset, applies stopwords filtering, and counts the occurrences of each word.

**Usage**

```
obtain_word_counts(data_annotated, data_stopwords, stopwords_to_append)
```

**Arguments**

`data_annotated` A data frame containing the annotated text data. The data must include columns like `document`, `submissionid`, `sentenceid`, and `lemma`.

`data_stopwords` A vector of stopwords that will be excluded from the word count.

`stopwords_to_append`  
A vector of additional stopwords to be appended to `data_stopwords` before counting words.

**Details**

This function works by first uniting the `document`, `submissionid`, and `sentenceid` columns into a new identifier `mID`. Then, it filters out stopwords from the word count, counts the frequency of words, and returns the result.

**Value**

A tibble containing the word counts for each unique combination of mID (document, submissionid, sentenceid) and word. The tibble has columns:

- mID: A unique identifier combining document, submissionid, and sentenceid.
- word: The word itself.
- n: The frequency of the word.

---

replace_abbr	<i>Replace Abbreviations</i>
--------------	------------------------------

---

**Description**

Replaces periods in abbreviations to ensure correct spacing and formatting.

**Usage**

```
replace_abbr(abbr)
```

**Arguments**

abbr                    A character string containing the abbreviation.

**Value**

A corrected abbreviation string.

---

run_sumup	<i>Run Sum Up</i>
-----------	-------------------

---

**Description**

This function runs a series of text processing and analysis steps including text cleaning, tokenization, lemmatization, topic modeling, and sentiment analysis. It then classifies sentences into topics and generates an output summarizing the results.

**Usage**

```
run_sumup(dataset, settings = NULL)
```

**Arguments**

dataset                A data frame containing the text data to be analyzed. It should include at least the following columns: sentenceid, sentence, portfolioid, competencyid, feedbacktype, and datereferenced.

settings               A list containing settings for various processing steps. If not provided, default settings are used.

## Details

This function performs the following steps:

- Cleans the input text data using `text_clean`.
- Tokenizes the text into sentences and removes stopwords.
- Lemmatizes and annotates the sentences using a UDPipe model.
- Counts word frequencies and excludes stopwords.
- Performs topic modeling on the word counts.
- Runs sentiment analysis based on the specified method (Grasp or SentimentR).
- Classifies sentences into topics using the topic classification model.
- Generates output summarizing the topics and sentiment.

## Value

A list or JSON output (depending on settings) containing the processed text data classified by topics and sentiment, as well as various metrics related to topics, such as strength and feedback count.

## Examples

```
data(example_data)
ex_data <- example_data
ex_settings <- set_default_settings()
ex_settings <- update_setting(ex_settings , "language", "en")
ex_settings <- update_setting(ex_settings , "use_sentiment_analysis", "sentimentr")
result <- run_sumup(ex_data, ex_settings )
```

---

sentiment\_analysis\_sentimentr

*Sentiment Analysis using sentimentr*

---

## Description

This function performs sentiment analysis on sentences using a predefined lexicon and the `sentimentr` package. The sentiment score is calculated based on a dictionary of words with associated sentiment values.

## Usage

```
sentiment_analysis_sentimentr(  
  documents,  
  sentences,  
  dictionary_file,  
  use_dictionary_file_in_sentimentr  
)
```

**Arguments**

documents	A character vector of document identifiers for which sentiment analysis needs to be performed. This is included for consistency, but is not directly used in the analysis.
sentences	A data frame containing sentence-level data. It should include the following columns: <ul style="list-style-type: none"> <li>• sentenceid: The unique identifier of the sentence.</li> <li>• sentence: The sentence text that needs to be analyzed.</li> <li>• document: The identifier of the document to which the sentence belongs.</li> </ul>
dictionary_file	A string representing the path to a CSV file containing the sentiment dictionary. The file should have two columns: <ul style="list-style-type: none"> <li>• word: The words in the lexicon.</li> <li>• value: The sentiment value associated with the word.</li> </ul>
use_dictionary_file_in_sentimentr	A boolean determining whether the dictionary file is used (TRUE) or the built-in lexicon (FALSE)

**Details**

This function loads a sentiment dictionary, processes the sentences, and calculates a sentiment score for each sentence using the `sentimentr` package. The dictionary is expected to have words associated with sentiment values that influence the sentiment score calculation. Positive sentiment scores indicate a positive sentiment, while negative values indicate negative sentiment.

The dictionary file (`SumUp_Dictionary.csv`) is read from the `data/` directory by default. The file should have a column of words and corresponding sentiment values.

**Value**

A data table with the following columns:

- sentenceid: The ID of the sentence.
- sentence: The sentence being analyzed.
- polarity: The sentiment polarity score for the sentence.
- document: The document identifier to which the sentence belongs.

---

set\_default\_settings    *Settings functionality for package 'sumup'*

---

**Description**

This file contains helper functions used throughout package 'sumup' to set and update the settings of the algorithm.

Create a list of settings for the algorithms in 'sumup'

**Usage**

```
set_default_settings()
```

**Value**

A list containing all settings used in 'sumup'

**Author(s)**

Joyce M.W. Moonen - van Loon

This file defines functions used to set and update the settings used in the 'sumup' algorithms Functions in this file:

- `set_default_settings()`: Set the default settings list of the package (optimized for a Dutch Master in Medicine education using CanMeds competencies and professional activities)
- `update_educational_framework_settings(settings)`: Update settings related to the educational framework
- `update_setting(settings, setting_name, new_value)`: Updates setting 'setting\_name' with value 'new\_value', returning the new list of settings
- `check_all_settings(settings)`: Check whether all settings are of the correct type and fit the required ranges
- `check_setting(settings, setting_name, setting_value)`: Check whether setting\_value is allowed for setting 'setting\_name'

Usage: Load this package and call the functions directly.

Note: This file is part of the 'sumup' package.

---

text\_clean

*Text Cleaning and Processing Functions*

---

**Description**

This file contains helper functions for text processing and cleaning used in the 'sumup' package. The functions provide capabilities for text correction, cleaning, dataset preparation, and handling abbreviation replacements.

Cleans narrative text columns (sterk, verbeter, feedback) in a dataset.

**Usage**

```
text_clean(data, corrections_file)
```

**Arguments**

`data` A data.table or data.frame containing text data.  
`corrections_file` Path to a CSV file containing typo corrections.

**Value**

A cleaned dataset with standardized text.

**Author(s)**

Joyce M.W. Moonen - van Loon

This file defines several functions:

- `text_clean()`: Cleans multiple columns in a dataset.
- `correct_text()`: Applies typo corrections using a predefined correction list.
- `clean_text()`: Cleans and formats input text, handling HTML tags, spacing, punctuation, and typos.
- `get_corrections()`: Reads a correction file and processes typo replacements.
- `create_dataset_narratives()`: Prepares a dataset by restructuring and normalizing its textual content.
- `replace_abbr()`: Corrects abbreviations.

---

tidy\_text

*Tidy and Split Narrative Text*

---

**Description**

This function processes narrative data by splitting the text into sentences or simply subsetting the data based on specific comment types. It ensures consistency across various comment types and removes unwanted columns and duplicates.

**Usage**

```
tidy_text(narratives, split_in_sentences = TRUE)
```

**Arguments**

`narratives` A data frame or `data.table` containing the narratives to be processed. The dataset should include columns representing different types of comments (e.g., `sterk`, `verbeter`, `feedback`).

`split_in_sentences` A logical value indicating whether to split the text into individual sentences. If `TRUE`, the function will split the narratives into sentences using regular expressions. If `FALSE`, it will only subset the data based on comment types.

### Details

The `tidy_text` function processes a dataset of narratives, splitting them into individual sentences (if `split_in_sentences = TRUE`) or subsetting them based on comment types (if `split_in_sentences = FALSE`). The comment types are predefined as `sterk`, `verbeter`, and `feedback`.

- When `split_in_sentences = TRUE`, the function unnests the narrative data into sentences using regular expressions to identify sentence boundaries (periods, question marks, and exclamation marks).
- When `split_in_sentences = FALSE`, the function subsets the data by comment type and ensures that the relevant columns are kept. The function also performs text cleaning by squishing spaces and removing HTML tags.

### Value

A `data.table` containing sentences (or narrative data) with the following columns:

- `document`: The document ID.
- `submissionid`: The submission ID.
- `competencyid`: The competency ID.
- `assistant`: The assistant information.
- `portfolioid`: The portfolio ID.
- `sentenceid`: A unique identifier for each sentence (or narrative entry).
- `sentence`: The cleaned-up sentence text.

---

`topic_classification` *Topic Classification for Narrative Data*

---

### Description

This function classifies sentences from narrative data into topics based on Latent Dirichlet Allocation (LDA) topic modeling results. It integrates additional features like sentiment analysis, annotated data, and seeded topics to enhance the classification.

### Usage

```
topic_classification(  
  data,  
  narratives,  
  sentences,  
  sentences_lda,  
  sentence_polarity,  
  data_annotated,  
  use_beta,  
  use_seeds,  
  nr_topics,  
  seeded_topics,  
  competencies  
)
```

**Arguments**

<code>data</code>	A data frame or <code>data.table</code> containing metadata and other relevant data for topic classification.
<code>narratives</code>	A data frame or <code>data.table</code> containing the narrative data, including comments and feedback.
<code>sentences</code>	A data frame or <code>data.table</code> containing the sentences to be classified, including the text and metadata.
<code>sentences_lda</code>	A result object from LDA topic modeling, which contains the topic distribution for each term.
<code>sentence_polarity</code>	A data frame or <code>data.table</code> containing sentence-level polarity scores (sentiment analysis results).
<code>data_annotated</code>	A data frame or <code>data.table</code> containing the lemmatized text and other annotations.
<code>use_beta</code>	A logical value indicating whether to use the beta values from the LDA model for topic classification.
<code>use_seeds</code>	A logical value indicating whether to incorporate seeded topics for topic classification.
<code>nr_topics</code>	An integer specifying the number of topics to classify.
<code>seeded_topics</code>	A list of character vectors representing the topics with predefined seed words.
<code>competencies</code>	A list of names of the competencies per id as used in the data set

**Details**

The `topic_classification` function integrates several sources of data to classify sentences into topics:

- It uses LDA topic modeling results to assign a probability for each topic.
- The function can incorporate seeded topics, which enhance the classification by matching predefined keywords to the topics.
- Sentiment analysis is used to add polarity scores to the sentence data.
- The output includes additional metadata for each sentence, such as feedback type, score, and associated comments.

**Value**

A `data.table` containing the classified sentences with the following columns:

- `sentenceid`: Unique identifier for each sentence.
- `sentence`: The sentence text.
- `polarity`: The sentiment polarity score of the sentence.
- `max_probability`: The highest topic probability for each sentence.
- Additional columns corresponding to each topic, representing the probability of the sentence belonging to that topic.
- Metadata fields such as `document`, `submissionid`, `competencyid`, `feedbacktype`, `score`, and `comment`.

---

 topic\_modeling

*Topic Modeling with Latent Dirichlet Allocation (LDA)*


---

### Description

This function performs topic modeling on word count data using Latent Dirichlet Allocation (LDA). It supports both standard LDA and seeded LDA, where predefined topics can guide the topic modeling process.

### Usage

```
topic_modeling(
  word_counts,
  seeded_topics,
  seed_weight,
  nr_topics,
  set_seed,
  lda_seed,
  lda_alpha,
  lda_best,
  lda_burnin,
  lda_verbose,
  lda_iter,
  lda_thin
)
```

### Arguments

word_counts	A data frame or data.table containing word counts, with columns for the document ID (mID), word (word), and count (n).
seeded_topics	A list of character vectors representing predefined terms for each seed topic. If provided, seeded LDA will be performed.
seed_weight	A numeric value indicating the weight assigned to the seeded terms in the LDA model. This parameter influences how strongly the predefined seed topics affect the topic modeling.
nr_topics	An integer specifying the number of topics to be modeled by the LDA algorithm.
set_seed	A numeric value setting the seed for the topic modeling algorithm. Default set to 1234.
lda_seed	A numeric seed to be set for Gibbs Sampling. Default set to 1000.
lda_alpha	A numeric value that set the initial value for alpha.
lda_best	If TRUE only the model with the maximum (posterior) likelihood is returned, by default equals TRUE.
lda_burnin	A number of omitted Gibbs iterations at beginning, by default equals 0.
lda_verbose	A numeric value. If a positive integer, then the progress is reported every verbose iterations. If 0 (default), no output is generated during model fitting.

lda_iter	Number of Gibbs iterations (after omitting the burnin iterations), by default equals 2000.
lda_thin	Number of omitted in-between Gibbs iterations, by default equals iter.

### Details

The `topic_modeling` function performs topic modeling using Latent Dirichlet Allocation (LDA) on a document-term matrix (DTM). If `seeded_topics` is provided, a seeded LDA approach is used where predefined topics help guide the model's generation of topics. The function supports:

- **Standard LDA:** Uses the traditional Gibbs sampling approach to estimate topics from word counts.
- **Seeded LDA:** Incorporates predefined seed terms into the LDA model by assigning a weight (`seed_weight`) to these terms.

The function uses the `topicmodels` package for LDA and the `slam` package to manipulate sparse matrices. The results are captured in an LDA model object, which contains topic-word distributions and document-topic assignments.

### Value

A `topicmodels` LDA object containing the result of the topic modeling. This object includes the topic distribution for each document and the terms associated with each topic.

---

update\_educational\_framework\_settings

*Update settings related to the educational framework*

---

### Description

When changing the `seeds_file` or `use_seeds` settings, update all parameters relating to the educational framework

### Usage

```
update_educational_framework_settings(settings)
```

### Arguments

`settings` A current list of settings (list)

### Value

An updated list containing all settings used in 'sumup'

---

update_setting	<i>Update setting</i>
----------------	-----------------------

---

**Description**

Replace the value of one particular setting value

**Usage**

```
update_setting(settings, setting_name, new_value)
```

**Arguments**

settings	A current list of settings (list)
setting_name	The name of the setting that needs updating (string)
new_value	The new value of the setting

**Value**

A list containing all settings used in 'sumup'

# Index

## \* datasets

- example\_data, [6](#)
- append\_stopwords, [2](#)
- check\_all\_settings, [3](#)
- check\_setting, [3](#)
- clean\_text, [4](#)
- correct\_text, [4](#)
- create\_dataset\_narratives, [5](#)
- create\_output, [5](#)
- default\_stopwords, [6](#)
- example\_data, [6](#)
- get\_corrections, [8](#)
- get\_polarity, [8](#)
- lemmatize, [9](#)
- obtain\_word\_counts, [10](#)
- replace\_abbr, [11](#)
- run\_sumup, [11](#)
- sentiment\_analysis\_sentimentr, [12](#)
- set\_default\_settings, [13](#)
- text\_clean, [14](#)
- tidy\_text, [15](#)
- topic\_classification, [16](#)
- topic\_modeling, [18](#)
- update\_educational\_framework\_settings,  
[19](#)
- update\_setting, [20](#)