Package ‘manifestoR’

October 13, 2022

Encoding UTF-8

Title Access and Process Data and Documents of the Manifesto Project

Date 2020-11-29

Version 1.5.0

Description Provides access to coded election programmes from the Manifesto
Corpus and to the Manifesto Project's Main Dataset and routines to analyse this
data. The Manifesto Project <https://manifesto-project.wzb.eu> collects and
analyses election programmes across time and space to measure the political
preferences of parties. The Manifesto Corpus contains the collected and
annotated election programmes in the Corpus format of the package 'tm' to enable
easy use of text processing and text mining functionality. Specific functions
for scaling of coded political texts are included.

Depends R (>= 3.1.0), NLP (>= 0.1-3), tm (>= 0.6)

Imports utilts, stats, methods, magrittr, httr (>= 1.0.0), jsonlite (>=
0.9.12), functional (>= 0.6), zoo (>= 1.7-11), psych,
base64enc, htmlwidgets (>= 0.6), DT (>= 0.2), htmltools, purrr
(>= 0.2.4), readr (>= 1.2.0), dplyr (>= 0.7.5), tidyselect (>=
1.0.0), tibble (>= 2.0.0)

Suggests knitr, rmarkdown, testthat (>= 1.0.2), R.rsp, haven (>=
1.0.0), readxl (>= 1.0.0), devtools (>= 1.7.0), formatR, highr

VignetteBuilder R.rsp

Collate manifestoR-package.r manifestoR-defunct.R globals.R
dataset.R codebook.R dedication.R

License GPL (>= 3)

URL https://github.com/ManifestoProject/manifestoR,
https://manifesto-project.wzb.eu/

BugReports https://github.com/ManifestoProject/manifestoR/issues

LazyData true
RoxygenNote 7.1.1

NeedsCompilation no

Author Jirka Lewandowski [aut],
   Nicolas Merz [aut],
   Sven Regel [aut, cre],
   Pola Lehmann [ctb],
   Paul Muscat [ctb]

Maintainer Sven Regel <sven.regel@wzb.eu>

Repository CRAN

Date/Publication 2020-11-29 23:00:09 UTC

R topics documented:

aggregate_pers ............................................. 3
aggregate_pers_cee ......................................... 4
attach_year .................................................. 5
clarity_dimensions ......................................... 5
codes ......................................................... 6
count_codes ................................................ 6
formatids ..................................................... 7
formatmpds .................................................. 8
franzmann_kaiser ........................................... 8
get_mpdb ...................................................... 9
get_viacache ................................................ 10
iff .......................................................... 11
issue_attention_diversity .................................. 11
ManifestoAvailability ....................................... 12
ManifestoCorpus ............................................. 13
ManifestoDocument ......................................... 13
ManifestoDocumentMeta ..................................... 14
manifestoR ................................................... 15
ManifestoSource ............................................ 15
median_voter ................................................ 16
mpdb_api_request .......................................... 18
mp_availability ............................................. 18
mp_bootstrap ............................................... 19
mp_check_for_corpus_update ................................ 20
mp_cite ..................................................... 21
mp_clarity .................................................. 21
mp_codebook ............................................... 22
mp_coreversions .......................................... 23
mp_corpus .................................................. 24
mp_corpusversions ......................................... 25
mp_dedication ............................................. 26
mp_emptycache ............................................. 26
mp_interpolate ............................................. 26
**aggregate_pers**

Aggregate category percentages in groups

**Description**

`aggregate_pers` is a general function to aggregate percentage variables by creating a new variable holding the sum. If a variable with the name for the aggregate already exists, it is overwritten, giving a warning if it is changed, not NA, not zero and not named "peruncod".

**Usage**

```r
aggregate_pers(
  data,
  groups = v5_v4_aggregation_relations(),
  na.rm = FALSE,
  keep = FALSE,
  overwrite = names(groups)
)
```
aggregate_pers_c ee

Arguments

- `data` : dataset to use in aggregation
- `groups` : (named) list of variable name vectors to aggregate to a new one (as given in the name); see default value for an example of the format
- `na.rm` : passed on to `sum`
- `keep` : keep variables that were aggregated in result?
- `overwrite` : Names of the variables that are allowed to be overwritten by aggregate. Defaults to all aggregate variable names. If a variable is overwritten, a message is issued in any case.

See Also

- `aggregate_pers_c ee`

aggregate_pers_c ee  Aggregate cee-categories to main categories

Description

Adds the code frequencies in a dataset of the 4 digit per-variables (per1011 to per7062 - mostly used in codings of Central and Eastern European countries) to the main categories in the coding scheme (3 digits).

Usage

```r
aggregate_pers_c ee(data)
```

Arguments

- `data` : dataset to use in aggregation

Details

A wrapper of `aggregate_pers` using `cee_aggregation_relations`.

See Also

- `aggregate_pers`
**attach_year**  
Compute year from date variable in MPDS

**Description**  
Compute year from date variable in MPDS

**Usage**  
attach_year(mpds)

**Arguments**  
mpds a dataframe in format of Manifesto Project Main Dataset

**Value**  
input data with year variable attached

**clarity_dimensions**  
Default programmatic clarity dimensions from Giebler/Lacewell/Regel/Werner 2015.

**Description**  
Default programmatic clarity dimensions from Giebler/Lacewell/Regel/Werner 2015.

**Usage**  
clarity_dimensions()

**References**  
codes

Access the codes of a Manifesto Document or Corpus

descriptions

With the accessor the codes of a Manifesto Document can be read and modified. The codes of a Manifesto Corpus can only be read, modification needs to be done document-wise.

Usage
codes(x, layer = "cmp_code")

## S3 method for class 'ManifestoDocument'
codes(x, layer = "cmp_code")

## S3 method for class 'ManifestoCorpus'
codes(x, layer = "cmp_code")

codes(x, layer = "cmp_code") <- value

## S3 replacement method for class 'ManifestoDocument'
codes(x, layer = "cmp_code") <- value

code_layers(x)

Arguments

x document or corpus to get the codes from
layer layer of codings to access, defaults to cmp_code, alternative: eu_code
value new codes

count_codes

Count the codings from a ManifestoDocument

descriptions

Count the codings from a ManifestoDocument

Usage
count_codes(
doc,
  code_layers = c("cmp_code"),
  with_eu_codes = "auto",
  prefix = "per",
)
relative = TRUE,
include_codes = if ("cmp_code" %in% code_layers) { v4_categories() } else {
c()
},
aggregate_v5_subcategories = TRUE
)

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>doc</code></td>
<td>ManifestoDocument, ManifestoCorpus or vector of codes</td>
</tr>
<tr>
<td><code>code_layers</code></td>
<td>vector of names of code layers to use, defaults to cmp_code; Caution: The layer eu_code is handled separately in the parameter with_eu_codes due to its different logic</td>
</tr>
<tr>
<td><code>with_eu_codes</code></td>
<td>Whether to include special EU code layer; by default (&quot;auto&quot;) taken from the document’s metadata</td>
</tr>
<tr>
<td><code>prefix</code></td>
<td>prefix for naming the count/percentage columns in the resulting data.frame</td>
</tr>
<tr>
<td><code>relative</code></td>
<td>If true, percentages are returned, absolute counts else</td>
</tr>
<tr>
<td><code>include_codes</code></td>
<td>Vector of categories that should be included even if they are not present in the data; the value of the created variables then defaults to 0.0 (or NA if no codes are present at all);</td>
</tr>
<tr>
<td><code>aggregate_v5_subcategories</code></td>
<td>if TRUE, for handbook version 5 subcategories, the aggregate category’s count/percentage is computed as well</td>
</tr>
</tbody>
</table>

**Value**

A data.frame with one row and the counts/percentages as columns

---

**formatids**  
*Format ids for web API queries*

**Description**

Formats a data.frame of ids such that it can be used for querying the Manifesto Project Database. That is, it must have non-NA-fields party and date.

**Usage**

`formatids(ids)`

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>ids</code></td>
<td>ids data.frame, information used: party, date, edate</td>
</tr>
</tbody>
</table>
frazmannekaiser

---

formatmpds

**Format the main data set**

**Description**

Creates the format that is visible to the R user from the internal data.frames files (in cache or from the API)

**Usage**

formatmpds(mpds)

**Arguments**

- mpds: A data.frame with a main data set version to be formatted

---

franzmann_kaiser

**Left-Right Scores based on Franzmann & Kaiser Method**

**Description**

Computes left-right scores based on the Franzmann & Kaiser Method (see reference below). The issue structures are not calculated from scratch but taken as given from Franzmann 2009 (or later updates). Note that they are not available for the entire Manifesto Project Dataset, but only for a subset of countries and elections.

**Usage**

franzmann_kaiser(
  data,
  basevalues = TRUE,
  smoothing = TRUE,
  vars = grep("per\d{3}\$", names(data), value = TRUE),
  issue_structure = read_fk_issue_structure(mean_presplit = mean_presplit),
  party_system_split = split_belgium,
  mean_presplit = TRUE,
  ...
)

read_fk_issue_structure(
  path = system.file("extdata", "fk_issue_structure_2019.csv", package = "manifestoR"),
  mean_presplit = TRUE,
  format_version = 2
)

fk_smoothing(data, score_name, use_period_length = TRUE, ...)

---
get_mpdb

Arguments

- **data**: A data.frame with cases to be scaled, variables named "per..."
- **basevalues**: flag for transforming data to be relative to the minimum
- **smoothing**: flag for using smoothing
- **vars**: Variables/Categories to use for computation of score. Defaults to all available handbook version 4 categories.
- **issue_structure**: issue structure to use for Franzmann & Kaiser method, default to most recent bundled version (for details see read_fk_issue_structure)
- **party_system_split**: function to recode the country variable to re-partition party systems. Defaults to splitting Belgium into two halves as done in Franzmann 2009
- **mean_presplit**: if TRUE, for Belgium as a whole (before the split into two party systems) the mean of the issue weights is used (which is equal to taking the mean of the output values, since all subsequent transformations are linear). This step is required to replicate the Franzmann 2009 dataset. If the issue structures already contain values for Belgium as a whole they are overwritten by the newly generated ones.
- **path**: path from were to read issue structures (as csv data file). Defaults to the most recent file bundled in the manifestoR package.
- **format_version**: can be 1 or 2 to switch between different structural versions of the issue structures file (1 for files containing "structure"-columns, 2 for files containing "per"-columns)
- **score_name**: name of variable with LR Score values to be smoothed
- **use_period_length**: whether to use electoral period length in weighting

References


get_mpdb  Download content from the Manifesto Database

Description

Internal implementation. For more convenient access and caching use one of mp_corpus, mp_availability, mp_maindataset.
Usage

get_mpdb(type, parameters = c(), versionid = NULL, apikey = NULL)

Arguments

- **type**: string of "meta", "text", "original", "main", "versions" to indicate type of content to get
- **parameters**: content filter parameters specific to type
- **versionid**: character string specifying the corpus version to use, either a name or tag as in the respective columns of the value of `mp_corpusversions` and the API
- **apikey**: API key to use, defaults to NULL, which means the key currently stored in the variable `apikey` of the environment `mp_globalenv` is used.

---

**get_viacache**  
*Get API results via cache*

Description

Get API results via cache

Usage

get_viacache(type, ids = c(), cache = TRUE, versionid = NULL, ...)

Arguments

- **type**: type of objects to get (metadata, documents, ...) as a string. Types are defined as constants in `globals.R`
- **ids**: identifiers of objects to get. Depending on the type a data.frame or vector of identifiers.
- **cache**: whether to use (TRUE) or bypass (FALSE) cache, defaults to TRUE
- **versionid**: string identifier of version to use
- **...**: additional parameters handed over to `get_mpdb`

Details

This function is internal to manifestoR and not designed for use from other namespaces
iff  

Apply a function if and only if test is TRUE

Description
otherwise return input value unchanged

Usage
iff(obj, test, fun, ...)
iffn(obj, test, fun, ...)

Arguments
obj    object to apply test and fun to
test   logical or function to apply to test
fun    function to apply
...    passed on to test

Details
iffn is ... if and only if test is FALSE

issue_attention_diversity

Issue Attention Diversity

Description
Effective number of Manifesto Issues suggested by Zac Greene. When using the measure please cite Greene 2015 (see reference below)

Usage
issue_attention_diversity(
data,
method = "shannon",
prefix = "per",
include_variables = paste0(prefix, setdiff(v4_categories(), "uncod")),
aggregate_categories = list(c(101, 102), c(104, 105), c(107, 109), c(108, 110),
c(203, 204), c(301, 302), c(406, 407), c(409, 414), c(504, 505), c(506, 507), c(601,
602), c(603, 604), c(607, 608), c(701, 702))
)
Arguments

data: a data.frame in format of Manifesto Project Main Dataset
method: entropy measure used for the effective number of manifesto issues. Possible options are "shannon" for Shannon's H and "herfindahl" for the Herfindahl-Index.
prefix: Prefix of variable names to use (usually "per")
include_variables: names of variables to include
aggregate_categories: list of category groups to aggregate into one issue. Default to selection used in Greene 2015

References


ManifestoAvailability

Manifesto Availability Information class

Description

Objects returned by `mp_availability`.

Details

ManifestoAvailability objects are data.frames with variables `party` and `date` identifying the requested manifestos as in the Manifesto Project’s Main & South America Datasets. The additional variables specify whether a machine readable document is available (`manifestos`), whether digital CMP coding annotations are available (`annotations`) or whether an original PDF is available (`originals`).

Additional a ManifestoAvailability object has attributes `query`, containing the original id set which was queried, `corpus_version`, specifying the Corpus version ID used for the query, and `date` with the timestamp of the query.

Examples

```r
## Not run:
wanted <- data.frame(party=c(41320, 41320), date=c(200909, 200509))
mp_availability(wanted)
## End(Not run)
```
ManifestoCorpus

**Manifesto Corpus class**

**Description**

Objects of this class are returned by `mp_corpus`.

**Usage**

```r
ManifestoCorpus(csource = ManifestoJSONSource())
```

**Arguments**

- `csource` a `ManifestoJSONSource`, see `Source`

**Details**

A `tm Corpus` storing `ManifestoDocuments`

For usage and structure of the stored documents see `ManifestoDocument`.

**Examples**

```r
## Not run: corpus <- mp_corpus(subset(mp_maindataset(), countryname == "Russia"))
```

---

ManifestoDocument

**Manifesto Document**

**Description**

A `ManifestoDocument` represents a document from the Manifesto Corpus and contains text, coding and meta information. `ManifestoDocument` objects need not be constructed manually but are the content of the `ManifestoCorpus` objects downloaded from the Manifesto Corpus Database API via `mp_corpus`.

`ManifestoDocuments` subclass the `TextDocument` class from the package `tm`. Hence they can be and usually are collected in a `tm Corpus` to interface easily with text mining and other linguistic analysis functions. `manifestoR` uses the subclass `ManifestoCorpus` of `tm`s `Corpus`, but `ManifestoDocuments` can be stored in any kind of `Corpus`.

As in `tm` any `ManifestoDocument` has metadata which can be accessed and modified via the `meta` function, as well as content, accessible via `content`. Additionally, via `codes()`, the coding of the (quasi-)sentence according to the CMP category scheme can be accessed (and modified). The CMP category scheme can be found online at [https://manifesto-project.wzb.eu/coding_schemes/mp_v4](https://manifesto-project.wzb.eu/coding_schemes/mp_v4) (version 4) or [https://manifesto-project.wzb.eu/coding_schemes/mp_v5](https://manifesto-project.wzb.eu/coding_schemes/mp_v5) (version 5).
ManifestoDocument

Usage

```
ManifestoDocument(
  content = data.frame(),
  id = character(0),
  meta = ManifestoDocumentMeta()
)
```

Arguments

- `content` - data.frame of text and codes for the ManifestoDocument to be constructed. There can be multiple columns of codes, but by default the accessor method `codes` searches for the column named "cmp_code".
- `id` - an id to identify the Document
- `meta` - an object of class `ManifestoDocumentMeta` containing the metadata for this document

Details

Internally, a ManifestoDocument is a data.frame with a row for every quasi-sentence and the columns `text` and `code`.

Examples

```r
## Not run:
corpus <- mp_corpus(subset(mp_maindataset(), countryname == "New Zealand"))
doc <- corpus[[1]]
print(doc)
## End(Not run)
```

ManifestoDocumentMeta

Manifesto Document Metadata

Description

Manifesto Document Metadata

Usage

```
ManifestoDocumentMeta(meta = list(), id = character(0))
```

Arguments

- `meta` - a named list with tag-value pairs of document meta information
- `id` - a character giving a unique identifier for the text document
manifestoR

Access and process data and documents of the Manifesto Project

Description

Provides access to coded election programmes from the Manifesto Corpus and to the Manifesto Project’s Main Dataset and routines to analyse this data. The Manifesto Project https://manifesto-project.wzb.eu collects and analyses election programmes across time and space to measure the political preferences of parties. The Manifesto Corpus contains the collected and annotated election programmes in the Corpus format of the package ‘tm’ to enable easy use of text processing and text mining functionality. Specific functions for scaling of coded political texts are included.

Details

manifestoR R package
Access and process data and documents of the Manifesto Project

<table>
<thead>
<tr>
<th>Package:</th>
<th>manifestoR</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type:</td>
<td>Package</td>
</tr>
<tr>
<td>License:</td>
<td>GPL (&gt;= 3)</td>
</tr>
<tr>
<td>LazyLoad:</td>
<td>yes</td>
</tr>
</tbody>
</table>

Author(s)

Jirka Lewandowski <jirka.lewandowski@wzb.eu>

See Also

Useful links:

- https://manifesto-project.wzb.eu: additional tutorials, documentation, data, and election programmes
- https://github.com/ManifestoProject/manifestoR: manifestoR on GitHub
- Report bugs at https://github.com/ManifestoProject/manifestoR/issues

ManifestoSource

Data Source for Manifesto Corpus

Description

Data Source for Manifesto Corpus
Usage

ManifestoSource(texts)

ManifestoJSONSource(
  texts = list(manifesto_id = c(), items = c()),
  query_meta = data.frame()
)

Arguments

texts    texts of the manifesto documents
query_meta metadata to attach to document by joining on manifesto_id

Details

Used internally for constructing ManifestoCorpus objects.

---

median_voter  Median Voter position

Description

The position of the median voter, calculated after Kim and Fording (1998; 2003), with possible adjustment after McDonald 2002.

Usage

median_voter(
  positions,
  voteshares = "pervote",
  scale = "rile",
  groups = c("country", "edate"),
  ...
)

median_voter_single(
  positions,
  voteshares,
  adjusted = FALSE,
  scalemin = -100,
  scalemax = 100
)
**median_voter**

**Arguments**

- `positions`: either a vector of values or (possible only for `median_voter`) a data.frame containing a column as named in argument scale (default: rile) and one as named in argument voteshares (default: pervote);
- `voteshares`: either a vector of values or (possible only for `median_voter`) the name of a column in the data.frame positions that contains the vote shares;
- `scale`: variable of which to compute the median voter position (default: rile);
- `groups`: names of grouping variables to use for aggregation, default results in one median voter position per election;
- `...`: further arguments passed to `median_voter_single`;
- `adjusted`: flag for adjustment after McDonald 2002;
- `scalemin`: The minimum of the scale of the positions, used for computing the voter position intervals;
- `scalemax`: The maximum of the scale of the positions, used for computing the voter position intervals.

**Details**

`median_voter` is able to compute the median voter positions for multiple elections at once, while `median_voter_single` treats data as coming from a single election.

Calculated according to the formula by Kim and Fording (1998; 2003)

\[ m = L + \frac{K - C}{F} W \]

Where \( m \) is the median voter position, \( L \) is lower end of the interval containing the median, \( K \) is \( 0.5 \times \text{sum(voteshare)} \), \( C \) is the cumulative vote share up to but not including the interval containing the median, \( F \) is the vote share in the interval containing the median and \( W \) is the width of the interval containing the median.

Different parties with the same left-right position (e.g. alliances) are treated as one party with the cumulative vote share.

In the adjusted formula the midpoint is "mirrored" from the midpoint of the other side: "Rather than assuming the party’s voters are so widely dispersed, this variable assumes they are spread in a symmetrical interval around the party’s position. For example, for a leftmost party at -15 and a 0 midpoint between it and an adjacent party on the right, we assume the left boundary of that party’s voters is -30." (McDonald 2002)

**References**


mpdb_api_request  
*Manifesto Project DB API request*

**Description**

gets the requested url and passes HTTP header error codes on to raise R errors with the same text

**Usage**

```r
mpdb_api_request(file, body)
```

**Arguments**

- **file**: file to request below apiroot url
- **body**: body text of the posted request: should contain the parameters as specified by the Manifesto Project Database API

---

mpAvailability  
*Availability information for election programmes*

**Description**

Availability information for election programmes

**Usage**

```r
mp_availability(ids, apikey = NULL, cache = TRUE)
```

**Arguments**

- **ids**: Information on which documents to get. This can either be a list of partys (as ids) and dates of elections as given to `mp_metadata` or a `ManifestoMetadata` object (data.frame) as returned by `mp_metadata`. Alternatively, ids can be a logical expression specifying a subset of the Manifesto Project’s main dataset. It will be evaluated within the data.frame returned by `mp_maindataset` such that all its variables and functions thereof can be used in the expression.
- **apikey**: API key to use. Defaults to NULL, resulting in using the API key set via `mp_setapikey`.
- **cache**: Boolean flag indicating whether to use locally cached data if available.

**Value**

an object of class `ManifestoAvailability` containing availability information. Can be treated as a `data.frame` and contains detailed availability information per document
Examples

```r
## Not run:
mp_availability(countryname == "New Zealand")

wanted <- data.frame(party=c(41320, 41320), date=c(200909, 200509))
mp_availability(wanted)
```

## End(Not run)

---

**mp_bootstrap**

*Compute bootstrap distributions for scaling functions*

**Description**

Bootstrapping of distributions of scaling functions as described by Benoit, Mikhaylov, and Laver (2009). Given a dataset with percentages of CMP categories, for each case the distribution of categories is resampled from a multinomial distribution and the scaling function computed for the resampled values. Arbitrary statistics of the resulting bootstrap distribution can be returned, such as standard deviation, quantiles, etc.

**Usage**

```r
mp_bootstrap(
  data,            # A data.frame with cases to be scaled and bootstrapped
  fun = rile,      # function of a data row the bootstraped distribution of which is of interest
  col_filter = "^per\(|\d\{3\}|\d\{4\}|uncod\)$",  # Regular expression matching the column names that should be permuted for the resampling (usually and by default the handbook v4_categories (plus cee_categories) per variables)
  statistics = list(sd),  # A list (!) of statistics to be computed from the bootstrap distribution; defaults to standard deviation (sd). Must be functions or numbers, where numbers are interpreted as quantiles.
  N = 1000,        # number of resamples to use for bootstrap distribution
  ignore_na = TRUE,  # if TRUE (default), for each observation drop silently the columns that have an NA value for the permutation
  ...               # if TRUE (default), for each observation drop silently the columns that have an NA value for the permutation
)
```

**Arguments**

- **data**: A data.frame with cases to be scaled and bootstrapped
- **fun**: function of a data row the bootstraped distribution of which is of interest
- **col_filter**: Regular expression matching the column names that should be permuted for the resampling (usually and by default the handbook v4_categories (plus cee_categories) per variables)
- **statistics**: A list (!) of statistics to be computed from the bootstrap distribution; defaults to standard deviation (sd). Must be functions or numbers, where numbers are interpreted as quantiles.
- **N**: number of resamples to use for bootstrap distribution
- **ignore_na**: if TRUE (default), for each observation drop silently the columns that have an NA value for the permutation
rescale if TRUE (default), rescale the permuted values after the permutation to the sum of the values of the col_filter columns instead of 100

... more arguments passed on to fun

References


---

mp_check_for_corpus_update

*Check for Updates of Corpus in Manifesto Project DB*

**Description**

*mp_check_for_corpus_update* checks if the currently cached version of corpus text and metadata is older than the most recent version available via the Manifesto Project DB API.

**Usage**

```r
mp_check_for_corpus_update(apikey = NULL, only_stable = TRUE)

mp_which_corpus_version(cache_env = mp_cache())

mp_which_dataset_versions(cache_env = mp_cache())

mp_update_cache(apikey = NULL, only_stable = TRUE)
```

**Arguments**

- **apikey** API key to use. Defaults to NULL, resulting in using the API key set via *mp_set_apikey*.
- **only_stable** Consider only for versions marked as stable by the Manifesto Project Team, defaults to TRUE.
- **cache_env** Cache environment

**Details**

*mp_update_cache* checks if a new corpus version is available and loads the new version via: *mp_use_corpus_version*. That is, the internal cache of manifestoR will automatically be updated to newer version and all future calls to the API will request for the newer version.

Note that this versioning applies to the corpus’ texts and metadata, and not the versions of the core dataset. For this see *mp_coreversions*.
**Value**

`mp_update_cache` returns a list with a boolean `update_available` and `versionid`, a character string identifying the most recent online version available.

`mp_which_corpus_version` returns the current version id of the corpus and metadata stored in the cache.

`mp_which_dataset_versions` returns the names of the main dataset versions which are in the cache, i.e. have been downloaded.

`mp_update_cache` returns the character identifier of the version updated to.

---

**mp_cite**

*Print Manifesto Corpus citation information*

**Description**

Print Manifesto Corpus citation information.

**Usage**

```r
mp_cite(
  corpus_version = mp_which_corpus_version(),
  core_versions = mp_which_dataset_versions(),
  apikey = NULL
)
```

**Arguments**

- **corpus_version**: corpus version for which citation should be printed.
- **core_versions**: core version for which citation should be printed.
- **apikey**: API key to use. Defaults to NULL, resulting in using the API key set via `mp_setapikey`.

---

**mp_clarity**

*Programmatic clarity measures (PC)*

**Description**

Computes party clarity measures suggested by Giebler/Lacewell/Regel/Werner 2015.

**Usage**

```r
mp_clarity(
  data,
  weighting_kind = "manifesto",
  weighting_source = NULL,
  auto_rescale_weight = TRUE,
  auto_rescale_variables = TRUE,
  dimensions = clarity_dimensions()
)
```
Arguments

data: a dataframe in format of Manifesto Project Main Dataset
weighting_kind: manifesto or election-specific weighting of the dimensions
weighting_source: name of variable with party importance (likely its importance within an election)
weighting (can be rmgs, pervote)
auto_rescale_weight: rescale party importance weighting within elections to 0-1
auto_rescale_variables: rescale dimension variables to 0-1
dimensions: dimensions to be used, must be in the format of the return value of `clarity_dimensions`

Value

a vector of clarity values

References


mp_codebook

Access to the Codebook for the Manifesto Project Main Dataset

Description

These functions provide access to machine- and human-readable versions of the Codebook (variable descriptions) of the Manifesto Project Main Dataset, as can be found in PDF form under https://manifesto-project.wzb.eu/datasets. As of this manifestoR release only the content-analytical variables (categories) are accessible. Note also that the codebook contains only condensed descriptions of the categories. For detailed information on coding instructions, you can refer to the different handbook versions under https://manifesto-project.wzb.eu/information/documents/handbooks. Only codebooks from version MPDS2017b on are accessible via the API.

Usage

mp_codebook(version = "current", cache = TRUE, chapter = "categories")

mp_describe_code(
  code,
  version = "current",
  columns = c("title", "description_md"),
  print = TRUE
)

mp_view_codebook(version = "current", columns = c("type", "code", "title"))
Arguments

version version of the Manifesto Project Main Dataset for which the codebook is requested. Note that only codebooks from version MPDS2017b on are available via the API/manifestoR. Defaults to "current", which fetches the most recent codebook version. Must be formatted as e.g. "MPDS2017b".

cache Whether result of API call should be cached locally (defaults to TRUE)

chapter Which part of the codebook should be returned. As of this manifestoR release, only the content-analytical variables (parameter value "categories") are accessible via the API.

code specific code(s) (as character (vector)) to display information about.

columns Information to display about each variable. Given as a vector of selected column names from: "type", "domain_code", "domain_name", "code", "variable_name", "title", "description_md", "label"

print if TRUE (default), print the information, but as the function also returns invisible a tibble containing the information, you can set print to FALSE for alternative uses.

Details

mp_codebook returns the codebook as a tibble, ideal for further automatic processing.

mp_describe_code pretty prints with information about the requested code(s), ideal for quick interactive use, but also returns invisible the code(s) information as a tibble

mp_view_codebook displays a searchable table version of the codebook in the Viewer pane.

mp_coreversions

List the available versions of the Manifesto Project’s Main Dataset

Description

List the available versions of the Manifesto Project’s Main Dataset

Usage

mp_coreversions(apikey = NULL, cache = TRUE, kind = "main")

Arguments

apikey API key to use. Defaults to NULL, resulting in using the API key set via mp_setapikey.

cache Boolean flag indicating whether to use locally cached data if available.

kind one of "main" (default) or "south_america" to discriminate the Main Dataset and the South America Dataset

Details

For the available versions of the corpus, see mp_corpusversions
Examples

### Not run: mp_coreversions()

## mp_corpus

### Description

Documents are downloaded from the Manifesto Project Corpus Database. If CMP coding annotations are available, they are attached to the documents, otherwise raw texts are provided. The documents are cached in the working memory to ensure internal consistency, enable offline use and reduce online traffic.

### Usage

```r
mp_corpus(
  ids,
  apikey = NULL,
  cache = TRUE,
  codefilter = NULL,
  codefilter_layer = "cmp_code"
)
```

### Arguments

- **ids**: Information on which documents to get. This can either be a list of parties (as ids) and dates of elections as given to `mp_metadata` or a `ManifestoMetadata` object (data.frame) as returned by `mp_metadata`. Alternatively, ids can be a logical expression specifying a subset of the Manifesto Project’s main dataset. It will be evaluated within the data.frame returned by `mp_maindataset` such that all its variables and functions thereof can be used in the expression.

- **apikey**: API key to use. Defaults to NULL, resulting in using the API key set via `mp_setapikey`.

- **cache**: Boolean flag indicating whether to use locally cached data if available.

- **codefilter**: A vector of CMP codes to filter the documents: only quasi-sentences with the codes specified in codefilter are returned. If NULL, no filtering is applied.

- **codefilter_layer**: layer to which the codefilter should apply, defaults to cmp_code

### Details

See `mp_save_cache` for ensuring reproducibility by saving cache and version identifier to the hard drive. See `mp_update_cache` for updating the locally saved content with the most recent version from the Manifesto Project Database API.

### Value

an object of `Corpus`’s subclass `ManifestoCorpus` holding the available of the requested documents
mp_corpusversions

Examples

```r
## Not run:
corpus <- mp_corpus(party == 61620 & rile > 10)

wanted <- data.frame(party=c(41320, 41320), date=c(200909, 201309))
mp_corpus(wanted)

mp_corpus(subset(mp_maindataset(), countryname == "France"))

partially_available <- data.frame(party=c(41320, 41320), date=c(200909, 200509))
mp_corpus(partially_available)

## End(Not run)
```

---

**mp_corpusversions**  
*List the available versions of the Manifesto Project’s Corpus*

**Description**

The Manifesto Project Database API assigns a new version code whenever changes to the corpus texts or metadata are made.

**Usage**

```r
mp_corpusversions(apikey = NULL)
```

**Arguments**

- `apikey`  
  API key to use. Defaults to `NULL`, resulting in using the API key set via `mp_setapikey`.

**Details**

This function always bypasses the cache.

**Value**

A character vector with the available version ids
**mp_dedication**  
*Print manifestoR package dedication*

**Description**
Print manifestoR package dedication

**Usage**
mp_dedication()

**Value**
mp_dedication returns the package dedication

---

**mp_emptycache**  
*Empty the manifestoR's cache*

**Description**
Empty the manifestoR’s cache

**Usage**
mp_emptycache()

---

**mp_interpolate**  
*Interpolate values within election periods*

**Description**
As the Manifesto Project’s variables are collected election-wise, values for the time/years in between elections are not naturally available. *mp_interpolate* allows to approximate them by several methods from the adjacent observations.

**Usage**

```r
mp_interpolate(
  df,
  vars = "(^rile$)|(^per((\d{3}(_\d)?)|\d{4})$),
  by = "year",
  approx = zoo::na.approx,
  ...
)
```
Arguments

- df: a data.frame with observations to be interpolated
- vars: a regular expression matching the names of the variables to be interpolated
- by: increment of the interpolation sequence, passed to `seq.Date`
- approx: Interpolation function, defaults to zoo’s `na.approx`
- ...: Further arguments, passed on to approx

Examples

```r
## Not run:
mp_interpolate(mp_maindataset(), method = "constant")
mp_interpolate(mp_maindataset(), approx = na.spline, maxgap = 3)
## End(Not run)
```

---

**mp_load_cache**

*Load manifestoR’s cache*

Description

Load a cache from a variable or file to manifestoR’s current working environment.

Usage

```r
mp_load_cache(cache = NULL, file = "mp_cache.RData")
```

Arguments

- cache: an environment that should function as manifestoR’s new cache. If this is NULL, the environment is loaded from the file specified by argument file.
- file: a file name from where the cache environment should be loaded

Examples

```r
## Not run: mp_load_cache() ## loads cache from file "mp_cache.RData"
```
Access the Manifesto Project’s Main Dataset

Description

Gets the Manifesto Project’s Main Dataset from the project’s web API or the local cache, if it was already downloaded before.

Usage

```r
mp_maindataset(
  version = "current",
  south_america = FALSE,
  download_format = NULL,
  apikey = NULL,
  cache = TRUE
)
```

```r
mp_southamerica_dataset(...)
```

Arguments

- **version**: Specify the version of the dataset you want to access. Use "current" to obtain the most recent, or use `mp_coreversions` for a list of available versions.
- **south_america**: Flag whether to download corresponding South America dataset instead of Main Dataset
- **download_format**: Download format. If not NULL, instead of the dataset being returned as an R data.frame, a file path to a temporary file in the specified binary format is returned. Can be one of c("dta", "xlsx", "sav"). With the "dta" option, labeled columns can be obtained.
- **apikey**: API key to use. Defaults to NULL, resulting in using the API key set via `mp_setapikey`.
- **cache**: Boolean flag indicating whether to use locally cached data if available.
- **...**: all arguments of `mp_southamerica_data` are passed on to `mp_maindataset`

Details

`mp_southamerica_dataset` is a shorthand for getting the Manifesto Project’s South America Dataset (it is equivalent to `mp_maindataset(..., south_america = TRUE)`).

Value

The Manifesto Project Main Dataset with classes `data.frame` and `tbl_df`
Examples

```r
## Not run:
mpds <- mp_maindataset()
head(mpds)
median(subset(mpds, countryname == "Switzerland")$rile, na.rm = TRUE)

## End(Not run)
## Not run:
mp_maindataset(download_format = "dta") %>% read_dta() ## requires package haven

## End(Not run)
```

---

**mp_metadata**

*Get meta data for election programmes*

Description

Get meta data for election programmes

Usage

```r
mp_metadata(ids, apikey = NULL, cache = TRUE)
```

Arguments

- **ids**: list of parties (as ids) and dates of elections, paired. Dates must be given either in the `date` or the `edate` variable, formatted in the way they are in the main data set in this package (date: as.numeric, YYYYMM, edate: as.Date()), see `mp_maindataset`. Alternatively, `ids` can be a logical expression specifying a subset of the Manifesto Project’s main dataset. It will be evaluated within the data.frame returned by `mp_maindataset` such that all its variables and functions thereof can be used in the expression.

- **apikey**: API key to use. Defaults to NULL, resulting in using the API key set via `mp_setapikey`.

- **cache**: Boolean flag indicating whether to use locally cached data if available.

Details

Meta data contain information on the available documents for a given party and election date. This information comprises links to the text as well as original documents if available, language, versions checksums and more.

Value

An object of class `ManifestoMetadata`, subclassing `data.frame` as well as `tbl_df` and containing the requested metadata in rows per election programme.
mp_nicheness

## Not run:
mp_metadata(party == 21221)

wanted <- data.frame(party=c(41320, 41320), date=c(200909, 200509))
mp_metadata(wanted)

## End(Not run)

### mp_nicheness

**Party nieness measures**

Computes party nieness measures suggested by Bischof 2015 and Meyer and Miller 2013.

#### Usage

```r
mp_nicheness(data, method = "bischof", ...)

nicheness_meyer_miller(
  data,
  groups = meyer_miller_2013_policy_dimensions(),
  transform = NULL,
  smooth = FALSE,
  weights = "pervote",
  party_system_normalization = TRUE,
  only_non_zero = TRUE
)

nicheness_bischof(
  data,
  out_variables = c("party", "date", "specialization", "nicheness", "nicheness_two"),
  groups = bischof_issue_groups(),
  diversification_bounds = c(0, rep(1/length(groups), length(groups))) %>% { 1 - (. * log(.)) } %>% sum(),
  smooth = function(x) { (x + lag(x, default = first(first(x))))/2 }
)
```

#### Arguments

- **data**: a dataframe or matrix in format of Manifesto Project Main Dataset
- **method**: choose between bischof and meyermiller
- **...**: parmaeters passed on to specialized functions for differnet methods

### Description

Computes party nieness measures suggested by Bischof 2015 and Meyer and Miller 2013.
**mp_nicheness**

- **groups**: groups of issues to determine niches/policy dimensions; formatted as named lists variable names. For Meyer & Miller: Defaults to adapted version of Baeck et. al 2010 Policy dimensions (without industry, as used in the original paper by Meyer & Miller). For Bischof: defaults to issue groups used in the Bischof 2015 paper.

- **transform**: transform to apply to each of the group indicators. Can be a function, character "bischof" to apply log(x + 1), or NULL for no transformation.

- **smooth**: Smoothing of policy dimension values before nicheness computation, as suggested and used by Bischof 2015.

- **weights**: vector of the length nrow(data) or the name of a variable in data; is used to weight mean party system position and nicheness; defaults to "pervote" as in Meyer & Miller 2013.

- **party_system_normalization**: normalize nicheness result within election (subtract weighted mean nicheness).

- **only_non_zero**: When dividing by the number of policy dimensions used for nicheness estimation, ignore dimensions that are zero for all parties (election-wise).

- **out_variables**: names of variables to return in data.frame. Can be any from the input or that are generated during the computation of Bischof’s nicheness measure. See details for a list.

- **diversification_bounds**: Bounds of the range of the diversification measure (Shannon’s entropy $s_p$ in Bischof 2015), used for inversion and normalization; default to the theoretical bounds of the entropy of a distribution on 5 discrete elements. If "empirical", the empirical max and min of the diversification measure are used.

**Details**

List of possible outputs of nicheness_bischof:

- diversification: Shannon’s entropy $s_p$ in Bischof 2015
- max_divers: used maximum for diversification
- min_divers: used minimum for diversification
- specialization: inverted diversification
- specialization_stand: standardized specialization
- nicheness: nicheness according to Meyer & Miller 2013 without vote share weighting
- nicheness_stand: standardized nicheness
- nicheness_two: sum of nicheness_stand and specialization_stand as proposed by Bischof 2015

**References**


---

**mp_rmps**  
*Relative measure of party size (RMPS)*

**Description**

Computes the relative measure of party size as suggested by Giebler/Lacewell/Regel/Werner 2015.

**Usage**

```r
mp_rmps(data, adapt_zeros = TRUE, ignore_na = TRUE, threshold_sum = 75)
```

**Arguments**

- `data`: a numerical vector with vote shares
- `adapt_zeros`: a boolean to switch on the conversion of zero values to 0.01 to avoid issues concerning division by zero
- `ignore_na`: a boolean to switch on ignoring NA entries, otherwise having NA entries will lead to only NA values in the result
- `threshold_sum`: the threshold of the sum of all vote shares for allowing the calculation

**Details**

Hint: In a dataset with multiple elections the usage of the function might require to calculate the measure per election (eg. using group_by)

**Value**

a vector of rmps values

**References**

mp_save_cache

Description
Saves manifestoR’s cache to the file system. This function can and should be used to store downloaded snapshots of the Manifesto Project Corpus Database to your local hard drive. They can then be loaded via mp_load_cache. Caching data in the file system ensures reproducibility of the scripts and analyses, enables offline use of the data and reduces unnecessary traffic and waiting times.

Usage
mp_save_cache(file = "mp_cache.RData")

Arguments
file a file from which to load the cache environment

Examples
## Not run: mp_save_cache() ## save to "mp_cache.RData" in current working directory

cmp_scale

Scaling annotated manifesto documents

Description
Since scaling functions such as scale_weighted only apply to data.frames with code percentages, the function mp_scale makes them applies them to a ManifestoCorpus or ManifestoDocument.

Usage
mp_scale(data, scalingfun = rile, scalingname = as.character(substitute(scalingfun)), recode_v5_to_v4 = (scalingname == "rile"), ...

document_scaling(scalingfun, returndf = FALSE, scalingname = "scaling", recode_v5_to_v4 = FALSE, ...

corpus_scaling(scalingfun, scalingname = "scaling", ...)

Arguments

- **data**: ManifestoDocument or ManifestoCorpus with coding annotations or a data.frame with category percentages
- **scalingfun**: a scaling function, i.e. a function that takes a data.frame with category percentages and returns scaled positions, e.g. `scale_weighted`.
- **scalingname**: the name of the scale which will be used as a column name when a data.frame is produced
- **recode_v5_to_v4**: recode handbook version 5 scheme to version 4 before scaling; this parameter is only relevant if data is a ManifestoDocument or ManifestoCorpus, but not for data.frames with code percentages
- **...**: further arguments passed on to the scaling function scalingfun, or `count_codes`
- **return_df**: if this flag is TRUE, a data.frame with category percentage values, scaling result and, if available party and date is returned by the returned function

See Also

- `scale`

---

**mp_setapikey**

*Set the API key for the Manifesto Documents Database.*

**Description**

If you do not have an API key for the Manifesto Documents Database, you can create one via your profile page on [https://manifesto-project.wzb.eu](https://manifesto-project.wzb.eu). If you do not have an account, you can register on the webpage.

**Usage**

```r
mp_setapikey(key.file = NULL, key = NA_character_)
```

**Arguments**

- **key.file**: file name containing the API key
- **key**: new API key

**Details**

The key is read from the file specified in key.file. If this argument is NULL, the key given in the argument key is used.
mp_use_corpus_version  Use a specific version of the Manifesto Project Corpus

Description
The internal cache of manifestoR will be updated to the specified version and all future calls to the API will request for the specified version. Note that this versioning applies to the corpus’ texts and metadata, and not the versions of the core dataset. For this see `mp_coreversions`.

Usage
```r
mp_use_corpus_version(versionid, apikey = NULL)
```

Arguments
- `versionid`: character id of the version to use (as received from API and `mp_corpusversions`)
- `apikey`: API key to use. Defaults to `NULL`, resulting in using the API key set via `mp_setapikey`.

mp_view_originals  View original documents from the Manifesto Corpus Database

Description
Original documents are opened in the system’s browser window. All original documents are stored on the Manifesto Project Website and the URLs opened are all from this site.

Usage
```r
mp_view_originals(ids, maxn = 5, apikey = NULL, cache = TRUE)
```

Arguments
- `ids`: Information on which originals to view. This can either be a list of parties (as ids) and dates of elections as given to `mp_metadata` or a ManifestoMetadata object (data.frame) as returned by `mp_metadata`. Alternatively, ids can be a logical expression specifying a subset of the Manifesto Project’s main dataset. It will be evaluated within the data.frame returned by `mp_maindataset` such that all its variables and functions thereof can be used in the expression.
- `maxn`: maximum number of documents to open simultaneously in browser, defaults to 5.
- `apikey`: API key to use. Defaults to `NULL`, resulting in using the API key set via `mp_setapikey`.
- `cache`: Boolean flag indicating whether to use locally cached data if available. The original documents themselves are not cached locally, but the metadata required to find them is.
Examples

```r
## Not run:
mp_view_originals(party == 41320 & date == 200909)

## End(Not run)
```

---

### na_replace

**Replace NAs in vector with fixed value**

#### Description

Replace NAs in vector with fixed value

#### Usage

```r
na_replace(vec, value = 0L)
```

#### Arguments

- `vec`: vector to replace NAs in
- `value`: value to inject for NA

---

### null_to_na

**Convert NULL to NA**

#### Description

Convert NULL to NA

#### Usage

```r
null_to_na(x)
```

#### Arguments

- `x`: element

#### Value

NA if the element is NULL, the element otherwise
prefix

Prefix a string of text

Description
Convenience function to use with magrittr wraps `paste0`, hence vectorised as `paste0`

Usage
```r
prefix(text, ...)
```

Arguments

- **text**: goes to the end, rest
- **...**: goes to the front.

readManifesto

Reader for `ManifestoSource`

Description
Reader for `ManifestoSource`

Usage
```r
readManifesto(elem, language, id)
```

Arguments

- **elem**: a named list with the component content
- **language**: is ignored
- **id**: a character giving a unique identifier for the created text document

Details
Used internally for constructing `ManifestoCorpus` objects. For the general mechanism refer to `tm`s `Reader` documentation.
rancode_kee_codes  *Process CMP codings*

**Description**
Several functions to process the CMP codings

**Usage**

```r
code_kee_codes(x)
aggregate_kee_codes(x)
recode_v5_to_v4(x)
```

**Arguments**

- `x` Vector of codes, ManifestoDocument or ManifestoCorpus

**Details**

- `rancode_kee_codes` recode the sub-categories used in coding several manifestos in Central and Eastern Europe (4 digits) to the main categories in the coding scheme (3 digits).
- `recode_v5_to_v4` recode the CMP codings according to the more specialized Coding Handbook Version 5 to the more general categories of Handbook Version 4. Codes 202.2, 605.2 and 703.2 are converted to 000, while all other subcategory codes with an appended dot and fourth digit are aggregated to the corresponding three-digit main category.

rep.data.frame  *Replicates cases in a data.frame*

**Description**
Replicates cases in a data.frame

**Usage**

```r
## S3 method for class 'data.frame'
rep(x, times = 1, ...)
```

**Arguments**

- `x` data.frame to replicate
- `times` number of replications
- `...` unused
Value

data.frame with cases replicated

Description

Simple linear rescaling of positions

Usage

rescale(pos, newmin = -1, newmax = 1, oldmin = min(pos), oldmax = max(pos))

Arguments

pos position data to be rescaled
newmin indicates the minimum of the new scale (default is -1)
newmax indicates the maximum of the new scale (default is +1)
oldmin indicates the minimum of the existing scale. Can be used to rescale from a known theoretical scale (e.g. -100). If left empty the empirical minimum is used.
oldmax indicates the maximum of the existing. See above.

rile

RILE

Description

Computes the RILE or other bipolar linear scaling measures for each case in a data.frame or ManifestoCorpus

Usage

rile(x)
logit_rile(x)

Arguments

x A data.frame with cases to be scaled, variables named "per..."
scale_weighted  Scaling functions

Description

Scaling functions take a data.frame of variables with information about political parties/text and position the cases on a scale, i.e. output a vector of values. For applying scaling functions directly to text documents, refer to mp_scale.

Usage

scale_weighted(
  data,
  vars = grep("per\(\(\d{3}(\_\d{2})?\|\d{4}(\|uncod)\)$", names(data), value = TRUE),
  weights = 1
)

scale_logit(data, pos, neg, N = data[, "total"], zero_offset = 0.5, ...)

scale_bipolar(data, pos, neg, ...)

scale_ratio_1(data, pos, neg, ...)

scale_ratio_2(data, pos, neg, ...)

Arguments

data  A data.frame with cases to be scaled

vars  variable names that should contribute to the linear combination; defaults to all CMP category percentage variables in the Manifesto Project's Main Dataset

weights  weights of the linear combination in the same order as 'vars'.

pos  variable names that should contribute to the numerator ("positively")

neg  variable names that should contribute to the denominator ("negatively")

N  vector of numbers of quasi sentences to convert percentages to counts

zero_offset  Constant to be added to prevent 0/0 and log(0); defaults to 0.5 (smaller than any possible non-zero count)

...  further parameters passed on to scale_weighted

Details

scale_weighted scales the data as a weighted sum of the variable values

If variable names used for the definition of the scale are not present in the data frame they are assumed to be 0. scale_weighted scales the data as a weighted sum of the category percentages

scale_logit scales the data on a logit scale as described by Lowe et al. (2011).
scale_bipolar scales the data by adding up the variable values in pos and subtracting the variable values in neg.

scale_ratio_1 scales the data taking the ratio of the difference of the sum of the variable values in pos and the sum of the variable values in neg to the sum of the variable values in pos and neg as suggested by Kim and Fording (1998) and by Laver & Garry (2000).

scale_ratio_2 scales the data taking the ratio of the sum of the variable values in pos and the sum of the variable values in neg.

References


See Also

mp_scale

split_belgium  Split Belgium party system into separate groups

Description

Recodes the country variable of a dataset to 218 (Flanders parties) and 219 (Wallonia parties) from 21 for Belgium

Usage

split_belgium(
  data,
  wallonia_parties = c(21111, 21322, 21422, 21423, 21425, 21426, 21522, 21911),
  brussels_parties = c(21424, 21912),
  belgium_parties = c(21320, 21420, 21520),
  flanders_parties = c(21112, 21221, 21321, 21330, 21421, 21430, 21521, 21913, 21914,
    21915, 21916, 21917),
  presplit_countrycode = 21,
  ...
)
Arguments

- data: data.frame in format of the Manifesto Project’s Main Dataset
- wallonia_parties: Party codes for the Wallonia half
- brussels_parties: Party codes for Brussel specific parties, are recoded to NA
- belgium_parties: Party codes for complete system, coded as presplit_countrycode
- flanders_parties: Party codes for the Flanders half
- presplit_countrycode: Country code for the belgium_parties
- ...: ignored

v4_categories

Lists of categories and category relations

Description

Code numbers of the Manifesto Project’s category scheme. For documentation see https://manifesto-project.wzb.eu/datasets.

Usage

v4_categories()

v5_categories(include_parents = TRUE)

cee_categories()

v5_v4_aggregation_relations()

cee_aggregation_relations()

rile_r()

rile_l()

Arguments

- include_parents: include v5-categories that have subcategories
vanilla

Vanilla Scaling by Gabel & Huber

Description
Computes scores based on the Vanilla method suggested by Gabel & Huber. A factor analysis identifies the dominant dimension in the data. Factor scores using the regression method are then considered as party positions on this dominant dimension.

Usage
vanilla(
  data,
  vars = grep("per\d{3}$", names(data), value = TRUE),
  invert = FALSE
)

Arguments

data A data.frame with cases to be scaled, variables named "per..."
vars variable names that should be used for the scaling (usually the variables per101, per102,....)
invert invert scores (to change the direction of the dimension to facilitate comparison with other indices) (default is FALSE)

References
Index

aggregate_cee_codes (recode_cee_codes), 38
aggregate_pers, 3, 4
aggregate_pers_cee, 4, 4
attach_year, 5
clee_aggregation_relations
   (v4_categories), 42
clee_categories (v4_categories), 42
clearity_dimensions, 5, 22
code_layers (codes), 6
codes, 6, 13, 14
codes<= (codes), 6
Corpus, 13, 24
corpus_scaling (mp_scale), 33
count_codes, 6, 34
document_scaling (mp_scale), 33
fk_smoothing (franzmann_kaiser), 8
formatids, 7
formatmpds, 8
franzmann_kaiser, 8
get_mpdb, 9
get_viacache, 10
iff, 11
iffn (iff), 11
issue_attention_diversity, 11
logit_rile (rile), 39
ManifestoAvailability, 12, 18
ManifestoCorpus, 13, 16, 24, 37
ManifestoDocument, 13, 13
ManifestoDocumentMeta, 14, 14
ManifestoJSONSource, 13
ManifestoJSONSource (ManifestoSource), 15
manifestoR, 15
ManifestoSource, 15, 37
median_voter, 16
median_voter_single, 17
median_voter_single (median_voter), 16
mp_availability, 9, 12, 18
mp_bootstrap, 19
mp_check_for_corpus_update, 20
mp_cite, 21
mp_clarity, 21
mp_codebook, 22
mp_coreversions, 20, 23, 28, 35
mp_corpus, 9, 13, 24
mp_corpusversions, 10, 23, 25, 35
mp_dedication, 26
mp_describe_code (mp_codebook), 22
mp_emptycache, 26
mp_interpolate, 26
mp_load_cache, 27, 33
mp_maindataset, 9, 18, 24, 28, 29, 35
mp_metadata, 18, 24, 29, 35
mp_nicheness, 30
mp_rmps, 32
mp_save_cache, 24, 33
mp_scale, 33, 40, 41
mp_setapikey, 18, 20, 21, 23–25, 28, 29, 34, 35
mp_southamerica_dataset
   (mp_maindataset), 28
mp_update_cache, 24
mp_update_cache
   (mp_check_for_corpus_update), 20
mp_use_corpus_version, 20, 35
mp_view_codebook (mp_codebook), 22
mp_view_originals, 35
mp_which_corpus_version
   (mp_check_for_corpus_update), 20
mp_which_dataset_versions
INDEX

(mp_check_for_corpus_update), 20
mpdb_api_request, 18

na.approx, 27
na_replace, 36
nicheness_bischof (mp_nicheness), 30
nicheness_meyer_miller (mp_nicheness), 30
null_to_na, 36

paste0, 37
prefix, 37

read_fk_issue_structure
  (franzmann_kaiser), 8
Reader, 37
readManifesto, 37
recode_cee_codes, 38
recode_v5_to_v4 (recode_cee_codes), 38
rep.data.frame, 38
rescale, 39
rile, 39
rile_l (v4_categories), 42
rile_r (v4_categories), 42

scale, 34
scale_bipolar (scale_weighted), 40
scale_logit (scale_weighted), 40
scale_ratio_1 (scale_weighted), 40
scale_ratio_2 (scale_weighted), 40
scale_weighted, 33, 34, 40, 40
sd, 19
seq.Date, 27
Source, 13
split_belgium, 41
sum, 4

tbl_df, 28, 29
TextDocument, 13

v4_categories, 42
v5_categories (v4_categories), 42
v5_v4_aggregation_relations
  (v4_categories), 42
vanilla, 43