Package: dbc (via r-universe)

October 24, 2024

Title Dictionary-Based Cleaning

Version 0.0.0.9000

Description Tools for dictionary-based data cleaning.

License MIT + file LICENSE

Encoding UTF-8

LazyData true

Roxygen list(markdown = TRUE)

RoxygenNote 7.2.3

Depends R (>= 2.10)

Imports dplyr, tidyr, rlang, janitor, lubridate, stringi, stats, queryr

Suggests testthat (>= 3.0.0)

Remotes epicentre-msf/queryr

Repository https://epicentre-msf.r-universe.dev

RemoteUrl https://github.com/epicentre-msf/dbc

RemoteRef HEAD

RemoteSha 33b48fba8deb45e47ac46c9a25b8cff950ced8bb

Contents

`check_categorical` *Produce a dictionary of non-valid categorical values within a dataset, for use in subsequent data cleaning*

Description

Values are compared against a user-provided dictionary specifying the allowed values of each categorical variable, after text standardization to account for minor differences in character case, spacing, and punctuation.

The resulting cleaning dictionary can then be manually reviewed to fill in appropriate replacement values for each non-valid categorical value, or a missing-value keyword indicating that the value should be converted to `NA`.

Usage

```
check_categorical(
  x,
  dict_allowed,
  dict_clean = NULL,
  vars_id = NULL,
  col_allowed_var = "variable",
  col_allowed_value = "value",
  fn = std_text,
  allow_na = TRUE,
  na = ".na",
  populate_na = FALSE,
  return_all = FALSE
)
```

Arguments

`x`	A data frame with one or more columns to check
`dict_allowed`	Dictionary of allowed values for each variable of interest. Must include columns for "variable" and "value" (the names of which can be modified with args `col_allowed_var` and `col_allowed_value`).
`dict_clean`	Optional dictionary of value-replacement pairs (e.g. from a previous run of this function). Must include columns "variable", "value", "replacement", and, if specified as an argument, all columns specified by `vars_id`.

`vars_id`	Optional vector of one or more ID columns within `x` on which corrections should be conditional. If not specified the cleaning dictionary contains one entry for each unique combination of variable and non-valid value. If specified the cleaning dictionary contains one entry for each unique combination of variable, non-valid value, and ID variable.
`col_allowed_var`	Name of column in `dict_allowed` giving variable name (defaults to "variable")
`col_allowed_value`	Name of column in `dict_allowed` giving allowed values (defaults to "value")
`fn`	Function to standardize raw values in both the dataset and dictionary prior to comparing, to account for minor variation in character case, spacing, punctuation, etc. Defaults to `std_text`. To omit the standardization step can use e.g. `as.character` or an identity function `function(x) x`.
`allow_na`	Logical indicating whether missing values should always be treated as 'allowed' even if not explicitly specified in `dict_allowed`. Defaults to `TRUE`.
`na`	Keyword to use within column "replacement" for values that should be converted to `NA`. Defaults to ".na". The keyword is used to distinguish between "replacement" values that are missing because they have yet to be manually verified, and values that have been verified and really should be converted to `NA`.
`populate_na`	Logical indicating whether to pre-populate column "replacement" with values specified by keyword `na`. If most non-valid values in `x` are non-correctable, pre-populating the keyword `na` can save time during the manual verification/correction phase. Defaults to `FALSE`.
`return_all`	Logical indicating whether to return all non-valid values including those already specified in argument `dict_clean` (if specified) (`TRUE`), or only the new non-valid entries not already specified in `dict_clean` (`FALSE`). Defaults to `FALSE`.

Value

Data frame representing a dictionary of non-valid values, to be used in a future data cleaning step (after specifying the corresponding replacement values). Columns include:

- columns specified in `vars_id`, if given
- `variable`: column name of variable within `x`
- `value`: non-valid value
- `replacement`: correct value that should replace a given non-valid value
- `new`: logical indicating whether the entry is new (TRUE) or already specified in argument `dict_clean` (`<NA>`)

Examples

```
# load example dataset, and dictionary of allowed categorical values
data(ll1)
data(dict_categ1)
```

```
# basic output
check_categorical(ll1, dict_allowed = dict_categ1)
```

---

check_dates — *Produce a dictionary of non-valid date values within a dataset, for use in subsequent data cleaning*

Description

The resulting cleaning dictionary can be manually reviewed to fill in appropriate replacement values for each non-valid date value, or a missing-value keyword indicating that the value should be converted to NA, and then used with function clean_dates.

Similar to check_numeric, values are considered 'non-valid' if they cannot be coerced using a given function. The default date-coercing function is parse_dates, which can handle a wide variety of date formats, but the user could alternatively specify a simpler function like as.Date. The user may also specify additional expressions that would indicate a non-valid date value. For example, the expression date_admit > Sys.Date() could be used to check for admission dates in the future.

Usage

```
check_dates(
  x,
  vars,
  vars_id,
  queries = list(),
  dict_clean = NULL,
  fn = parse_dates,
  na = ".na",
  populate_na = FALSE
)
```

Arguments

x	A data frame with one or more columns to check
vars	Names of date columns within x to check
vars_id	Vector of one or more ID columns within x on which corrections should be conditional.
queries	Optional list of expressions to check for non-valid dates. May include a .x selector which is a stand-in for any of the date variables specified in argument vars. E.g.

```
list(
  date_admit > date_exit,  # admission later than exit
  .x > Sys.Date()          # any date in future
)
```

`dict_clean`	Optional dictionary of value-replacement pairs (e.g. produced by a prior run of `check_dates`). Must include columns "variable", "value", "replacement", and all columns specified by `vars_id`.
`fn`	Function to parse raw date values. Defaults to `parse_dates`. Any value not coercible by `fn` will be flagged as a "Non-valid date".
`na`	Keyword to use within column "replacement" for values that should be converted to `NA`. Defaults to ".na". The keyword is used to distinguish between "replacement" values that are missing because they have yet to be manually verified, and values that have been verified and really should be converted to `NA`.
`populate_na`	Logical indicating whether to pre-populate column "replacement" with values specified by keyword `na`, for queries of type "Non-valid date". If most non-valid dates in `x` are non-correctable, pre-populating the keyword `na` can save time during the manual verification/correction phase. Defaults to `FALSE`.

Value

Data frame representing a dictionary of non-valid values, to be used in a future data cleaning step (after specifying the corresponding replacement values). Columns include:

- columns specified in `vars_id`
- `variable`: column name of date variable within `x`
- `value`: raw date value
- `date`: parsed date value
- `replacement`: correct value that should replace a given non-valid value
- `query`: which query was triggered by the given raw date value (if any)

Note that, unlike functions `check_numeric` and `check_categorical`, which only return rows corresponding to non-valid values, this function returns all date values corresponding to any observation (i.e. row) with at least one non-valid date value. This is to provide context for the non-valid value and aid in making the appropriate correction.

Examples

```
# load example dataset
data(ll1)

# basic output
check_dates(
  ll1,
  vars = c("date_onset", "date_admit", "date_exit"),
  vars_id = "id"
)

# add additional queries to evaluate
check_dates(
  ll1,
  vars = c("date_onset", "date_admit", "date_exit"),
  vars_id = "id",
  queries = list(
    date_onset > date_admit,
    date_admit > date_exit,
    .x > as.Date("2021-01-01")
  )
)
```

---

check_numeric	*Produce a dictionary of non-valid numeric values within a dataset, for use in subsequent data cleaning*

Description

The resulting cleaning dictionary can then be manually reviewed to fill in appropriate replacement values for each non-valid numeric value, or a missing-value keyword indicating that the value should be converted to NA.

Usage

```
check_numeric(
  x,
  vars,
  vars_id = NULL,
  queries = list(),
  dict_clean = NULL,
  fn = as.numeric,
  na = ".na",
  populate_na = FALSE,
  return_all = FALSE
)
```

Arguments

x	A data frame with one or more columns to check
vars	Names of columns within x to check
vars_id	Optional vector of one or more ID columns within x on which corrections should be conditional. If not specified the cleaning dictionary contains one entry for each unique combination of variable and non-valid value. If specified the cleaning dictionary contains one entry for each unique combination of variable, non-valid value, and ID variable.
queries	Optional list of expressions to check for non-valid values. May include a .x selector which is a stand-in for any of the numeric variables specified in argument vars. E.g.

```
list(
  age > 110,  # age greater than 110
  .x < 0      # any numeric value less than 0
)
```

`dict_clean`	Optional dictionary of value-replacement pairs (e.g. from a previous run of this function). Must include columns "variable", "value", "replacement", and, if specified as an argument, all columns specified by `vars_id`.
`fn`	Function to convert values to numeric. Defaults to `as.numeric`.
`na`	Keyword to use within column "replacement" for values that should be converted to `NA`. Defaults to ".na". The keyword is used to distinguish between "replacement" values that are missing because they have yet to be manually verified, and values that have been verified and really should be converted to `NA`.
`populate_na`	Logical indicating whether to pre-populate column "replacement" with values specified by keyword `na`. If most non-valid values in `x` are non-correctable, pre-populating the keyword `na` can save time during the manual verification/correction phase. Defaults to `FALSE`.
`return_all`	Logical indicating whether to return all non-valid values including those already specified in argument `dict_clean` (if specified) (`TRUE`), or only the new non-valid entries not already specified in `dict_clean` (`FALSE`). Defaults to `FALSE`.

Value

Data frame representing a dictionary of non-valid values, to be used in a future data cleaning step (after specifying the corresponding replacement values). Columns include:

- columns specified in `vars_id`, if given
- `variable`: column name of variable within `x`
- `value`: non-valid value
- `replacement`: correct value that should replace a given non-valid value
- `new`: logical indicating whether the entry is new (TRUE) or already specified in argument `dict_clean` (<NA>)

Examples

```
# load example dataset
data(ll1)
data(clean_num1)

# basic output
check_numeric(ll1, c("age", "contacts"))

# include id var "id"
check_numeric(ll1, c("age", "contacts"), vars_id = "id")

# add custom query
check_numeric(ll1, c("age", "contacts"), vars_id = "id", queries = list(age > 90))
```

```
# prepopulate column 'replacement'
check_numeric(ll1, c("age", "contacts"), vars_id = "id", populate_na = TRUE)

# use dictionary of pre-specified corrections
check_numeric(ll1, c("age", "contacts"), dict_clean = clean_num1)
```

`clean_categ1` *A cleaning dictionary for categorical variables in example dataset* `ll1`

Description

A cleaning dictionary for categorical variables in example dataset `ll1`

Usage

```
clean_categ1
```

Format

A data.frame with 7 rows and 4 variables:

variable Column name within dataset

value Non-valid numeric value

replacement Replacement value for given non-valid value

new Logical indicating whether the dictionary entry is new

`clean_categorical` *Clean categorical variables within a dataset based on a dictionary of value-replacement pairs*

Description

Applies a dictionary of value-replacement pairs to clean and standardize values of categorical variables. Includes options for text standardization to standardize minor differences in character case, spacing, and punctuation.

Usage

```
clean_categorical(
  x,
  dict_allowed,
  dict_clean = NULL,
  vars_id = NULL,
  col_allowed_var = "variable",
  col_allowed_value = "value",
  non_allowed_to_missing = TRUE,
  fn = std_text,
  na = ".na"
)
```

Arguments

`x`	A data frame with one or more columns to clean
`dict_allowed`	Dictionary of allowed values for each variable of interest. Must include columns for "variable" and "value" (the names of which can be modified with args `col_allowed_var` and `col_allowed_value`).
`dict_clean`	Optional dictionary of value-replacement pairs (e.g. produced by `check_categorical`). Must include columns "variable", "value", "replacement", and, if specified as an argument, all columns specified by `vars_id`.
`vars_id`	Optional vector of one or more ID columns within `x` on which corrections should be conditional. If not specified the cleaning dictionary contains one entry for each unique combination of variable and non-valid value. If specified the cleaning dictionary contains one entry for each unique combination of variable, non-valid value, and ID variable.
`col_allowed_var`	Name of column in `dict_allowed` giving variable name (defaults to "variable")
`col_allowed_value`	Name of column in `dict_allowed` giving allowed values (defaults to "value")
`non_allowed_to_missing`	Logical indicating whether to replace values that remain non-allowed, even after cleaning and standardization, to NA. Defaults to TRUE. If no dictionary is provided, will simply standardize columns to match allowed values specified in `dict_allowed`.
`fn`	Function to standardize raw values in both the dataset and dictionary prior to comparing, to account for minor variation in character case, spacing, punctuation, etc. Defaults to `std_text`. To omit the standardization step can use e.g. `as.character` or an identity function `function(x) x`.
`na`	Keyword to use within column "replacement" for values that should be converted to `NA`. Defaults to ".na". The keyword is used to distinguish between "replacement" values that are missing because they have yet to be manually verified, and values that have been verified and really should be converted to `NA`.

Value

The original data frame x but with cleaned versions of the categorical variables specified in argument dict_allowed

Examples

```
# load example dataset, dictionary of allowed categorical values, and
# cleaning dictionary
data(ll1)
data(dict_categ1)
data(clean_categ1)

# dictionary-based corrections to categorical vars
clean_categorical(
  ll1,
  dict_allowed = dict_categ1,
  dict_clean = clean_categ1
)

# require exact matching, including character case
clean_categorical(
  ll1,
  dict_allowed = dict_categ1,
  dict_clean = clean_categ1,
  fn = identity
)

# apply standardization to dict_allowed but no additional dict-based cleaning
clean_categorical(
  ll1,
  dict_allowed = dict_categ1
)
```

clean_dates — *Clean date variables within a dataset based on a dictionary of value-replacement pairs*

Description

Applies a dictionary of value-replacement pairs and a conversion function (defaults to parse_dates) to clean and standardize values of date variables. To use this approach the date columns of the original dataset should generally be imported as type "text" or "character" so that non-valid values are not automatically coerced to missing values on import.

Usage

```
clean_dates(x, vars, vars_id, dict_clean = NULL, fn = parse_dates, na = ".na")
```

Arguments

`x`	A data frame with one or more date columns to clean
`vars`	Names of date columns within `x` to clean
`vars_id`	Vector of one or more ID columns within `x` on which corrections should be conditional.
`dict_clean`	Optional dictionary of value-replacement pairs (e.g. produced by a prior run of `check_dates`). Must include columns "variable", "value", "replacement", and all columns specified by `vars_id`.
`fn`	Function to parse raw date values. Defaults to `parse_dates`.
`na`	Keyword to use within column "replacement" for values that should be converted to `NA`. Defaults to ".na". The keyword is used to distinguish between "replacement" values that are missing because they have yet to be manually verified, and values that have been verified and really should be converted to `NA`.

Value

The original data frame `x` but with cleaned versions of the date variables specified in argument `vars`

Examples

```
# load example dataset and cleaning dictionary
data(ll1)
data(clean_dates1)

# clean dates using only date coercion function
clean_dates(
  ll1,
  vars = c("date_onset", "date_admit", "date_exit"),
  vars_id = "id"
)

# clean dates using dictionary and coercion function
clean_dates(
  ll1,
  vars = c("date_onset", "date_admit", "date_exit"),
  vars_id = "id",
  dict_clean = clean_dates1
)
```

`clean_dates1` *A cleaning dictionary for categorical variables in example dataset* `ll1`

Description

A cleaning dictionary for categorical variables in example dataset `ll1`

Usage

```
clean_dates1
```

Format

A data.frame with 12 rows and 6 variables:

id ID column within raw dataset

variable Column name within raw dataset

value Raw date value

date Parsed date value

replacement Replacement value for given non-valid value

query Which query was triggered by the given raw date value (if any)

`clean_num1` *A cleaning dictionary for numeric variables in example dataset* `ll1`

Description

A cleaning dictionary for numeric variables in example dataset `ll1`

Usage

```
clean_num1
```

Format

A data.frame with 4 rows and 4 variables:

variable Column name within dataset

value Non-valid numeric value

replacement Replacement value for given non-valid value

new Logical indicating whether the dictionary entry is new

`clean_numeric` *Clean numeric variables within a dataset based on a dictionary of value-replacement pairs*

Description

Applies a dictionary of value-replacement pairs and a conversion function (defaults to `as.numeric`) to clean and standardize values of numeric variables. To use this approach the numeric columns of the original dataset should generally be imported as type "text" or "character" so that non-valid values are not automatically coerced to missing values on import.

Usage

```
clean_numeric(
  x,
  vars,
  vars_id = NULL,
  dict_clean = NULL,
  fn = as.numeric,
  na = ".na"
)
```

Arguments

`x`	A data frame with one or more columns to clean
`vars`	Names of columns within `x` to clean
`vars_id`	Optional vector of one or more ID columns within `x` on which corrections should be conditional. If not specified the cleaning dictionary contains one entry for each unique combination of variable and non-valid value. If specified the cleaning dictionary contains one entry for each unique combination of variable, non-valid value, and ID variable.
`dict_clean`	Optional dictionary of value-replacement pairs (e.g. produced by `check_numeric`). If provided, must include columns "variable", "value", "replacement", and, if specified as an argument, all columns specified by `vars_id`. If no dictionary is provided, will simply apply the conversion function to all columns specified in `vars`.
`fn`	Function to convert values to numeric. Defaults to `as.numeric`.
`na`	Keyword to use within column "replacement" for values that should be converted to `NA`. Defaults to ".na". The keyword is used to distinguish between "replacement" values that are missing because they have yet to be manually verified, and values that have been verified and really should be converted to `NA`.

Value

The original data frame `x` but with cleaned versions of columns `vars`

Examples

```
# load example dataset and dictionary of value-replacement pairs
data(ll1)
data(clean_num1)

# dictionary-based corrections to numeric vars 'age' and 'contacts'
clean_numeric(
  ll1,
  vars = c("age", "contacts"),
  dict_clean = clean_num1
)

# apply standardization with as.integer() rather than default as.numeric()
clean_numeric(
  ll1,
  vars = c("age", "contacts"),
  dict_clean = clean_num1,
  fn = as.integer
)

# apply standardization but no dictionary-based cleaning
clean_numeric(
  ll1,
  vars = c("age", "contacts")
)
```

`dict_categ1` *A dictionary of allowed values for categorical variables in example dataset* `ll1`

Description

A dictionary of allowed values for categorical variables in example dataset `ll1`

Usage

```
dict_categ1
```

Format

A data.frame with 13 rows and 2 variables:

variable Column name within dataset

value Allowed categorical values for given column

`ll1` *An example messy dataset to clean*

Description

An example messy dataset to clean

Usage

```
ll1
```

Format

A data.frame with 7 rows and 10 variables:

id Patient identifier

age Age value

age_unit Age units

sex Patient sex

status Patient status

contacts Number of epidemiological contacts

date_onset Date of symptom onset

date_admit Date of admission to hospital

date_exit Date of exit from hospital

exit_status Patient outcome status

`ll2` *An example messy dataset to clean, an extension of* `ll1`

Description

An example messy dataset to clean, an extension of `ll1`

Usage

```
ll2
```

Format

A data.frame with 10 rows and 10 variables:

id Patient identifier

age Age value

age_unit Age units

sex Patient sex

status Patient status

contacts Number of epidemiological contacts

date_onset Date of symptom onset

date_admit Date of admission to hospital

date_exit Date of exit from hospital

exit_status Patient outcome status

`parse_dates` *Parse dates*

Description

Parse dates

Usage

```
parse_dates(
  x,
  convert_excel = TRUE,
  orders = c("Ymd", "dmY", "dmy", "mdY", "Ymd HMS")
)
```

Arguments

`x`	A character or numeric vector of dates
`convert_excel`	Logical indicating whether to convert Excel-encoded date values (e.g. "42370") into dates, using janitor::excel_numeric_to_date
`orders`	a character vector of date-time formats. Each order string is a series of formatting characters as listed in `base::strptime()` but might not include the `"%"` prefix. For example, "ymd" will match all the possible dates in year, month, day order. Formatting orders might include arbitrary separators. These are discarded. See details for the implemented formats. If multiple order strings are supplied, they are applied in turn for `parse_date_time2()` and `fast_strptime()`. For `parse_date_time()` the order of applied formats is determined by `select_formats` parameter.

Value

A vector of class "Date". Values that cannot be converted to valid dates will be returned as `<NA>`.

Examples

```
x <- c("44087", "12//02/2019", "2020_05_14", "2021-01-30 14:00:04")
parse_dates(x)
```

`std_text` *Standardize text prior to matching to account for minor variation in character case, spacing, punctuation, or use of accents*

Description

Implements the following transformations:

1. standardize case (`base::tolower`)
2. remove diacritic/accent characters (`stringi::stri_trans_general`)
3. remove sequences of space or punctuation characters at start or end of string
4. replace repeated whitespace characters with a single space

Usage

```
std_text(x)
```

Arguments

`x`	A vector of strings

Value

The standardized version of `x`

Examples

```
std_text(c("CONFIRMED", "Conf.", "confirmed"))
std_text(c("R\u00e9publique d\u00e9mocratique du  Congo", "Nigeria_"))
```

Index