TRACEing simulist and cleanepi

Karim Mané

2025-06-10

OUTLINE

Data preparation functionalities

- scan through the data
- column names cleaning
- replace missing data with NA
- constant data removal

Data standardisation and transformation functionalities

- date standardisation
- sequence of date events verification
- time span
- dictionary-based cleaning
- conversion to numeric

Other functionalities

Loading webR...

1. Get the input data

Loading webR...

```
# get simulated data
set.seed(1)
test_data <- simulist::sim_linelist() |>
  simulist::messy_linelist(inconsistent_dates = TRUE)

dim(test_data)
```

Input data structure

id	case_name	case_type	sex	age	date_onset	date_reporting	date_admission	outcome	date_outcome	date_first_contact	date_last_contact	ct_value
1	Lolette Phillips	suspected	NA	59	01 January 2023	NA	09 January 2023	died	13 January 2023	NA	NA	NA
two	James Jack	suspected	NA	90	01-01-2023	01-01-2023	NA	recovered	NA	29-12-2022	03-01-2023	NA
3	Chen Kantha	confirued	M	four	NA	NA	NA	recovered	NA	NA	01 January 2023	24.8
five	NA	probable	NA	twenty-nine	04-01-2023	04-01-2023	NA	NA	NA	28-12-2022	04-01-2023	NA
6	David Ponzio	confirmed	myle	fourteen	05 Jan 2023	05 Jan 2023	09 Jan 2023	died	23 Jan 2023	NA	04 Jan 2023	24.6
seven	Christopher Ward	probable	mmle	eighty-five	06-01-2023	06-01-2023	08-01-2023	recovered	NA	31-12-2022	06-01-2023	NA
10	Laura Ilaoa	NA	NA	twenty-five	13 January 2023	13 January 2023	NA	recovered	NA	02 January 2023	07 January 2023	NA
11	Morgan Mason	suspected	f	NA	11 Jan 2023	11 Jan 2023	24 Jan 2023	died	27 Jan 2023	03 Jan 2023	07 Jan 2023	NA
12	Cornelius Turner	confirmed	M	eighty-nine	NA	13-01-2023	NA	recovered	NA	03-01-2023	11-01-2023	24.7
fourteen	Shaddaad el-Younes	suspected	Male	63	2023/01/11	2023/01/11	NA	recovered	NA	2023/01/04	2023/01/09	NA
eighteen	Carlo Ceazar Corpuz	conhirmed	M	74	2023-01-14	2023-01-14	NA	recovered	NA	2023-01-09	2023-01-17	NA

Scan through the data

Loading webR...

```r
scan_result <- cleanepi::scan_data(
  data = test_data
)
```

- Function name: `scan_data()`
- Get % numeric, date, character, logical and missing values in all `character` columns of the data frame
- Identify potential columns where cleaning is required

Data scanning output

Field_names	missing	numeric	date	character	logical
id	0.0805	0.4425	0.0000	0.4770	0
case_name	0.1379	0.0000	0.0000	0.8621	0
case_type	0.0920	0.0000	0.0000	0.9080	0
sex	0.1379	0.0000	0.0000	0.8621	0
age	0.1149	0.4425	0.0000	0.4425	0
date_onset	0.1149	0.0000	0.8851	0.0000	0
date_reporting	0.1782	0.0000	0.8218	0.0000	0
date_admission	0.8103	0.0000	0.1897	0.0000	0

What to consider?

- Syntax in column names
- Columns with multiple data types

What to do?

- Check your data dictionary to make a decision
- Use {cleanepi} functionalities to clean the messy columns 🚀

Standardise column names

Loading webR...

```r
# PRINT COLUMN NAMES BEFORE
print(names(test_data))

# KEEP 'date_admission' AS IS,
# RENAME 'id' AND 'sex' TO 'case_id' AND 'gender' RESPECTIVELY
cleaned_data <- cleanepi::standardize_column_names(
  data = test_data,
  keep = "date_admission",
  rename = c(case_id = "id", gender = "sex")
)

# PRINT COLUMN NAMES AFTER
print(names(cleaned_data))
```

- Function name: `standardize_column_names()`
- Standardise column names on snake-case
- Offers flexibility to specify a subset of:
  - focal columns to preserve their original format using the `keep` argument
  - columns to be renamed using the `rename` argument

Replace missing values with NA

Loading webR...

```
test_data <- cleaned_data
cleaned_data <- cleanepi::replace_missing_values(
  data = test_data,
  na_strings = NULL,
  target_columns = NULL
)
```

- Use the code below to show the default missing value strings

Loading webR...

```
cleanepi::common_na_strings
```

- Function name: `replace_missing_values()`
- R functions easily handle `NA`
- `target_columns` to specify a vector of column names to be considered
- `na_strings` to specify the strings representing missing values in your data

Remove constant data

Loading webR...

```
test_data <- cleaned_data
# REMOVE THE CONSTANT COLUMNS, AND EMPTY ROWS AND COLUMNS
cleaned_data <- cleanepi::remove_constants(
  data = test_data,
  cutoff = 1
)
```

- Function name: `remove_constants()`
- Constant data: `empty rows and columns`, and `constant columns`
- `Iteratively` remove empty rows and columns as well as constant columns
- `cutoff`: to define the %constant data above which rows and columns should be deleted (varies between **0** and **1**)
- Delete rows and columns which do not add any variability to the data

Area of improvement

It's currently impossible to apply the filtration on only rows or only columns. Use `{janitor}` for such filtration.

Standardise dates

Loading webR...

```
date_columns <- cleaned_data |>
  dplyr::select(dplyr::starts_with("date_"))
```

date_onset	date_reporting	date_admission	date_outcome	date_first_contact	date_last_contact
01 January 2023	NA	09 January 2023	13 January 2023	NA	NA
01-01-2023	01-01-2023	NA	NA	29-12-2022	03-01-2023
NA	NA	NA	NA	NA	01 January 2023
04-01-2023	04-01-2023	NA	NA	28-12-2022	04-01-2023
05 Jan 2023	05 Jan 2023	09 Jan 2023	23 Jan 2023	NA	04 Jan 2023
06-01-2023	06-01-2023	08-01-2023	NA	31-12-2022	06-01-2023
13 January 2023	13 January 2023	NA	NA	02 January 2023	07 January 2023
11 Jan 2023	11 Jan 2023	24 Jan 2023	27 Jan 2023	03 Jan 2023	07 Jan 2023
NA	13-01-2023	NA	NA	03-01-2023	11-01-2023
2023/01/11	2023/01/11	NA	NA	2023/01/04	2023/01/09
2023-01-14	2023-01-14	NA	NA	2023-01-09	2023-01-17
NA	12-01-2023	NA	NA	07-01-2023	14-01-2023
12 January 2023	12 January 2023	NA	NA	07 January 2023	17 January 2023
2023-01-13	2023-01-13	NA	NA	2023-01-08	2023-01-15
2023-01-14	2023-01-14	2023-01-19	2023-01-27	2023-01-09	2023-01-16
21 January 2023	21 January 2023	NA	NA	08 January 2023	14 January 2023
NA	2023/01/19	NA	NA	2023/01/07	2023/01/13

Loading webR...

```r
test_data <- cleaned_data
# STANDARDIZE THE SPECIFIED DATE COLUMNS
# SET TO NA ANY VALUE THAT IS OUTSIDE OF THE SPECIFIED TIMEFRAME
cleaned_data <- cleanepi::standardize_dates(
  data = test_data,
  target_columns = c("date_onset", "date_reporting", "date_admission",
                     "date_outcome", "date_first_contact",
                     "date_last_contact"),
  format = NULL,
  timeframe = as.Date(c("2022-12-30", "2023-04-15")),
  error_tolerance = 0.4,
  orders = list(
    world_named_months = c("Ybd", "dby"),
    world_digit_months = c("dmy", "Ymd"),
    US_formats = c("Omdy", "YOmd")
  )
)
```

- Function name: `standardize_dates()`
- Convert date values into ISO8601 format: `YYYY-mm-dd` suitable for handling date values in R
- `target_columns`: to provide a vector of column to be converted.
- `format`: to specify the date format in the target columns if known
- `timeframe`: to define the expected time frame within which the date values should fall
- `error_tolerance`: to define the maximum percentage of `NA` values (non date values) that can be allowed in a converted column.

- Default `orders` list

Loading webR...

```
orders <- list(
  quarter_partial_dates = c("Y", "Ym", "Yq"),
  world_digit_months = c("ymd", "ydm", "dmy", "mdy", "myd", "dym", "Ymd", "Ydm",
                         "dmY", "mdY", "mYd", "dYm"),
  world_named_months = c("dby", "dyb", "bdy", "byd", "ybd", "ydb", "dbY", "dYb",
                         "bdY", "bYd", "Ybd", "Ydb"),
  us_format = c("Omdy", "YOmd")
)
```

- To give priority to american-style dates

Loading webR...

```
us_ord <- orders[c(4, 1, 2, 3)]
```

- To allow for the conversion of values such as “2014_04_05_23:15:43”

Loading webR...

```
orders$ymdhms <- c("Ymdhms", "Ymdhm")
```

Date Standardisation output

date_onset	date_reporting	date_admission	date_outcome	date_first_contact	date_last_contact
2023-01-01	NA	2023-01-09	2023-01-13	NA	NA
2023-01-01	2023-01-01	NA	NA	NA	2023-01-03
NA	NA	NA	NA	NA	2023-01-01
2023-01-04	2023-01-04	NA	NA	NA	2023-01-04
2023-01-05	2023-01-05	2023-01-09	2023-01-23	NA	2023-01-04
2023-01-06	2023-01-06	2023-01-08	NA	2022-12-31	2023-01-06
2023-01-13	2023-01-13	NA	NA	2023-01-02	2023-01-07
2023-01-11	2023-01-11	2023-01-24	2023-01-27	2023-01-03	2023-01-07

What to consider?

- `error_tolerance`: when the %NA (non-date values) in a character column > this threshold, the column will be returned as it is.
- `date guesser`: fails in some occasions
  - Preferably use `target_columns` for better performance
- Check the report to see the values that:
  - fall outside of the specified timeframe
  - comply with more than one specified format

Check the sequence of date events

Loading webR...

```r
# DETECT ROWS WITH INCORRECT DATE SEQUENCE
test_data <- cleaned_data
cleaned_data <- cleanepi::check_date_sequence(
  data = test_data,
  target_columns = c("date_first_contact", "date_last_contact")
)
```

- Function name: `check_date_sequence()`
- Spot out the values that do not follow the desired chronological order in the specified date columns
- Target columns should be `stansardised` and `listed in the expected order of occurrence` that reflects the chronological sequence of events

Calculate timespan

Loading webR...

```r
# CALCULATE TIME SPAN BETWEEN FIRST AND LAST CONTACT
test_data <- cleaned_data
cleaned_data <- cleanepi::timespan(
  data = test_data,
  target_column = "date_first_contact",
  end_date = "date_last_contact",
  span_column_name = "first_to_last_contact_time",
  span_unit = "months",
  span_remainder_unit = "days"
)
```

- Function name: `timespan()`
- Calculate the time span between two date variables
- `target_column`: the name of the target column
- `end_date`: a date column from the input data or a vector of date values or a single date value
- `span_unit`: the unit in which the time span should be expressed
- `span_column_name`: the name of the column added to the input data
- `span_remainder_unit`: the unit in which the remainder of the time span calculation will be returned

Timespan calculation output

date_first_contact	date_last_contact	first_to_last_contact_time	remainder_days
NA	NA	NA	NA
NA	2023-01-03	NA	NA
NA	2023-01-01	NA	NA
NA	2023-01-04	NA	NA
NA	2023-01-04	NA	NA
2022-12-31	2023-01-06	0	6
2023-01-02	2023-01-07	0	5
2023-01-03	2023-01-07	0	4

Convert character columns into numeric

case_id	age
1	59
two	90
3	four
five	twenty-nine
6	fourteen
seven	eighty-five
10	twenty-five
11	NA

Loading webR...

```
# CONVERT THE 'case_id' AND 'age' columns into numeric
test_data <- cleaned_data
cleaned_data <- cleanepi::convert_to_numeric(
  data = test_data,
  target_columns = c("case_id", "age"),
  lang = "en"
)
```

- `lang`: the language in which the letters are written. Currently one of `"en"`, `"fr"`, or `"es"` for `English`,

Conversion to numeric output

case_id	age
1	59
2	90
3	4
5	29
6	14
7	85
10	25
11	NA

Dictionary-based data substitution

- Function name: `clean_using_dictionary()`
- Replace the options in a data frame or linelist with their corresponding values stored in a data dictionary
- The structure of the data dictionary should adhere to the standards expected by the {matchmaker} package

Loading webR...

```r
# DISPLAY UNIQUE VALUES THE 'case_type' COLUMN
unique_options <- unique(cleaned_data$case_type)
unique_options

# CREATE THE DATA DICTIONARY
options <- unique_options[!is.na(unique_options)]
values <- c("suspected", "confirmed", "probable", "confirmed", "confirmed",
            "probable", "confirmed", "probable", "probable", "probable",
            "probable", "confirmed", "probable", "confirmed", "confirmed",
            "suspected", "probable", "confirmed")
dictionary <- data.frame(
  options = options,
  values = values,
  grp = rep("case_type", length(values)),
  orders = 1:length(values)
)
head(dictionary)
```

- Perform the substitution

Loading webR...

```
test_data <- cleaned_data
cleaned_data <- cleanepi::clean_using_dictionary(
  data = test_data,
  dictionary = dictionary
)

# print out the new values in the 'case_type' column
unique(cleaned_data$case_type)
```

- We will allow for string matching substitution in the next version

Other cleanepi functionalities

- Create simulated data

Loading webR...

```
set.seed(1)
test_data <- simulist::sim_linelist() |>
  simulist::messy_linelist(inconsistent_dates = TRUE)
```

- Scan through the data

Loading webR...

```
scan_res <- cleanepi::scan_data(test_data)
```

Perform several cleaning operations

Loading webR...

```r
cleaned_data <- test_data |>
  cleanepi::standardize_column_names(
    keep = "date_admission",
    rename = c(case_id = "id", gender = "sex")
  ) |>
  cleanepi::replace_missing_values(target_columns = NULL,na_strings = NULL) |>
  cleanepi::remove_constants(cutoff = 1.0) |>
  cleanepi::standardize_dates(
    target_columns = c("date_onset", "date_reporting", "date_admission",
                       "date_outcome", "date_first_contact",
                       "date_last_contact"),
    format = NULL,
    timeframe = as.Date(c("2022-12-30", "2023-04-15")),
    error_tolerance = 0.4,
    orders = list(
      world_named_months = c("Ybd", "dby"),
      world_digit_months = c("dmy", "Ymd"),
      US_formats = c("Omdy", "YOmd")
    )
  )
```

Loading webR...

```r
cleaned_data <- cleaned_data |>
  cleanepi::remove_duplicates(target_columns = NULL)
  cleanepi::convert_to_numeric(
    target_columns = c("case_id", "age"),
    lang = "en"
  ) |>
  cleanepi::check_subject_ids(
    target_columns = "case_id",
    prefix = NULL,
    suffix = NULL,
    range = c(1, 100),
    nchar = NULL
  ) |>
  cleanepi::clean_using_dictionary(dictionary = dictionary)
```

- Add the data scanning result to the report

Loading webR...

```
cleaned_data <- cleanepi::add_to_report(
   x = cleaned_data,
   key = "scanning_result",
   value = scan_res
)
```

- Print the report

Loading webR...

```
cleanepi::print_report(
  data = cleaned_data,
  report_title = "{cleanepi} data cleaning report",
  output_file_name = NULL,
  format = "html",
  print = TRUE
)
```

Useful resources

- Epiverse_TRACE github repo: https://github.com/epiverse-trace
- {cleanepi} documentation: https://epiverse-trace.github.io/cleanepi/
- raise an issue at: https://github.com/epiverse-trace/cleanepi/issues