data.org

RuReady?

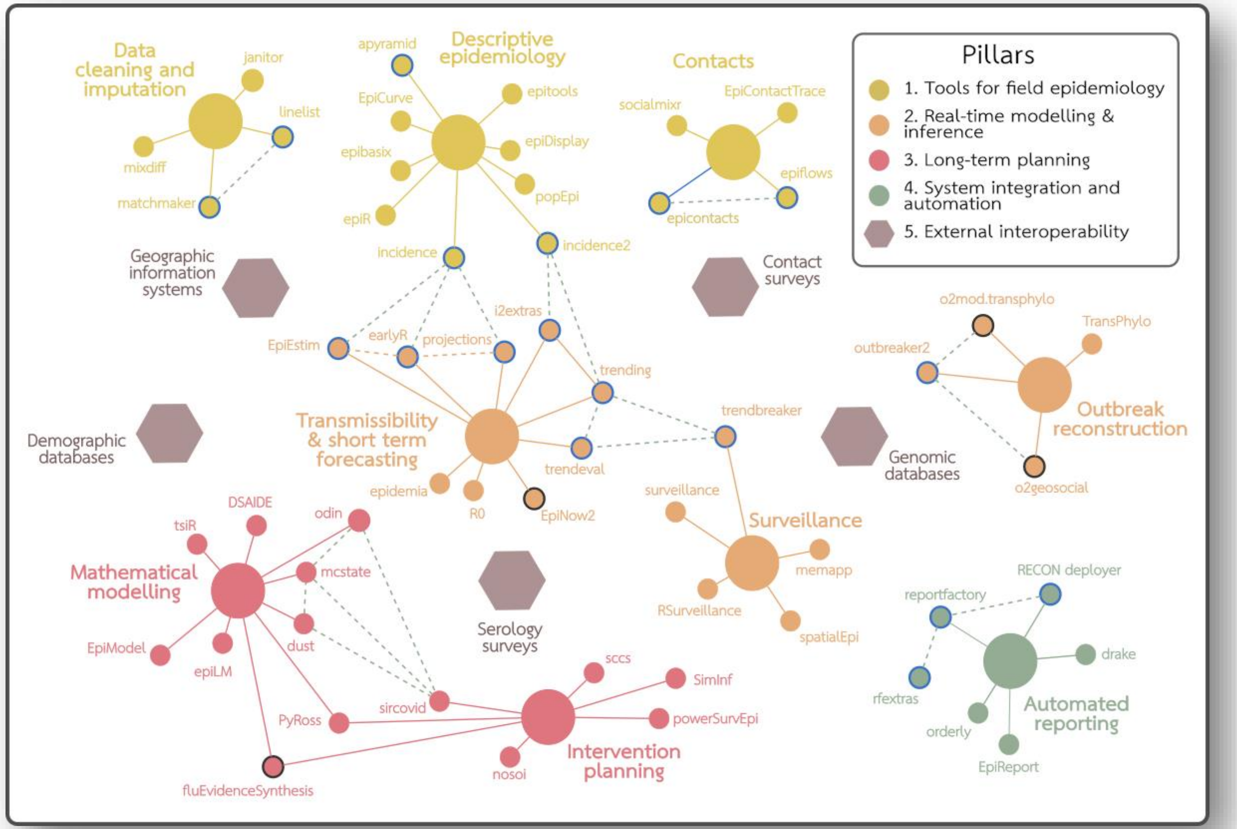
Rolling out a blueprint for software design: discussion on Epiverse's *linelist* package

London - 7 June 2022

Thibaut Jombart
Software Design and Implementation Lead, Data.org

Current ecosystem of R package for outbreak analytics

CURRENT LANDSCAPE



CHALLENGES

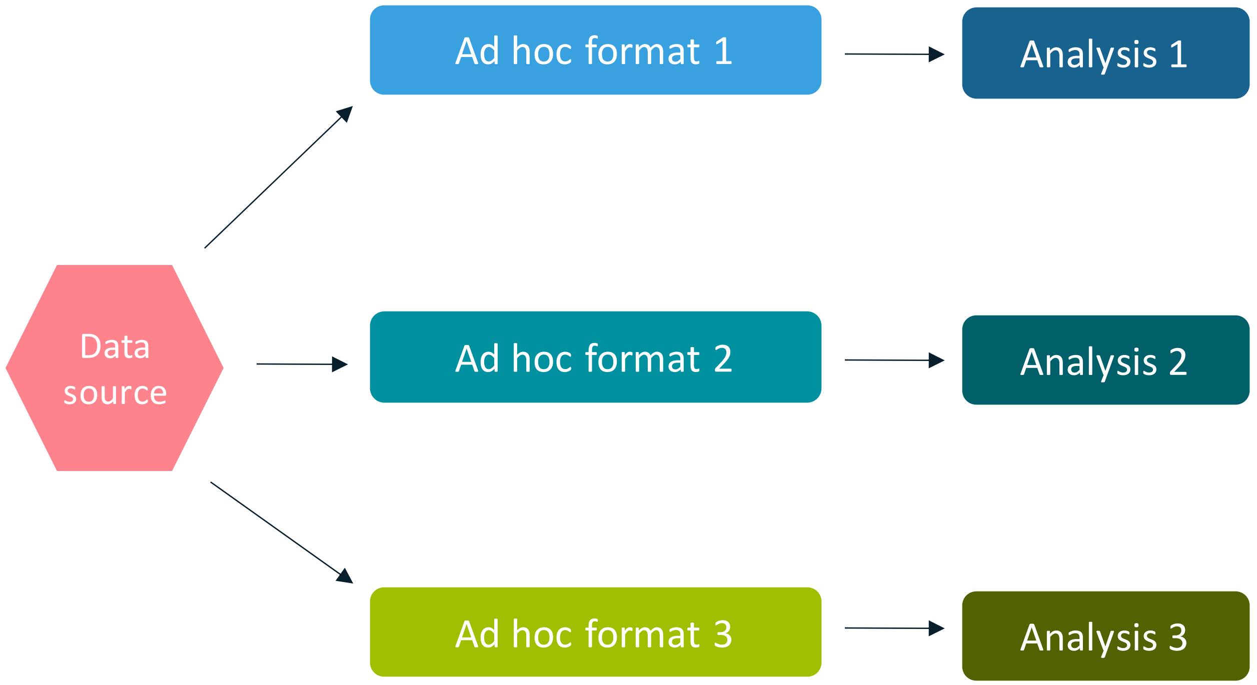
- Limited interoperability
- Lack of common data structures
- Data pipelines harder to build

data.org

Context

Ad hoc versus canonical data representations

CURRENTLY COMMON MODEL



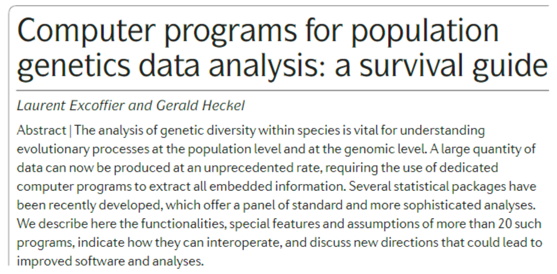
ISSUES

- Pre-processing overhead
  - Requires ad-hoc scripts
  - Sometimes more complex than analysis itself
  - May require dedicated import functions
  - Discourages multiple analyses and methods comparisons
- Prone to data discrepancies
  - Mistakes can be introduced during pre-processing
  - Hinders comparison of methods and results

data.org

Context

Why *ad hoc* formats are a problem: lessons from population genetics

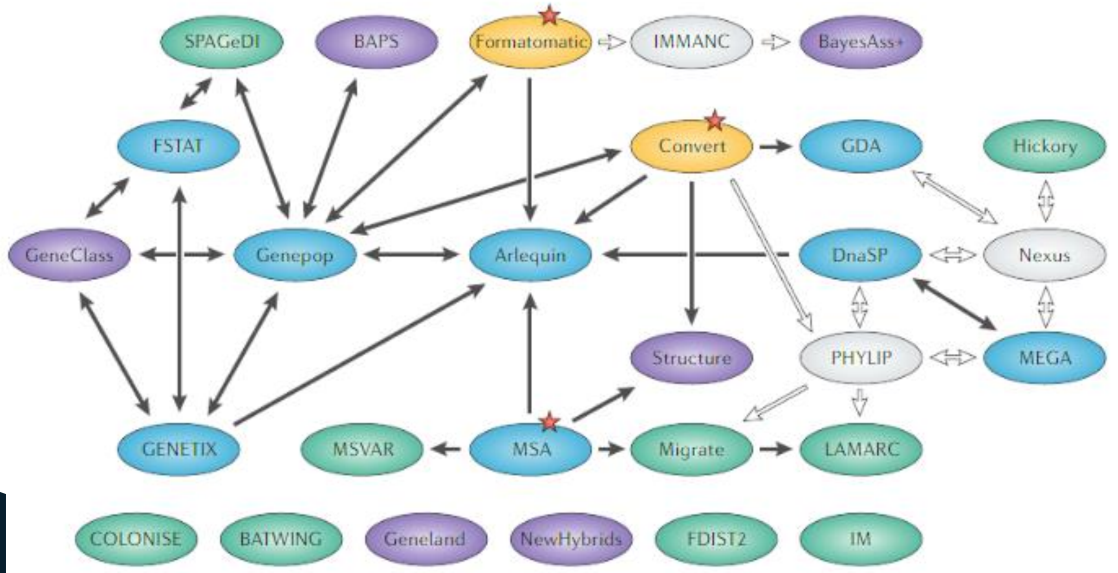


Computer programs for population genetics data analysis: a survival guide

Laurent Excoffier and Gerald Heckel

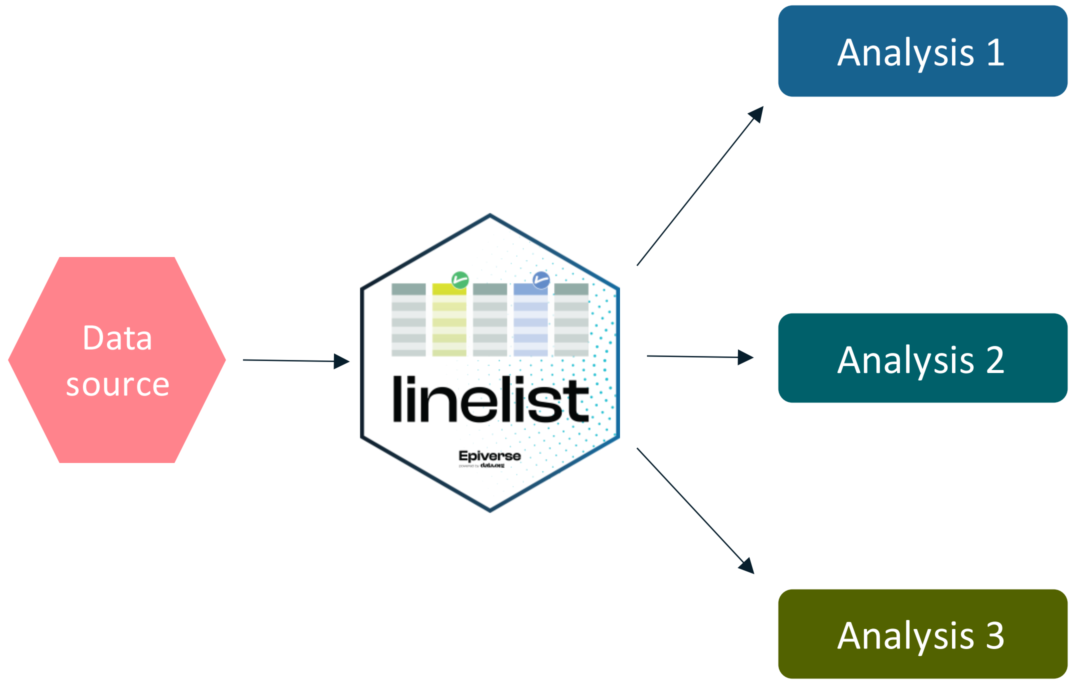
Abstract | The analysis of genetic diversity within species is vital for understanding evolutionary processes at the population level and at the genomic level. A large quantity of data can now be produced at an unprecedented rate, requiring the use of dedicated computer programs to extract all embedded information. Several statistical packages have been recently developed, which offer a panel of standard and more sophisticated analyses. We describe here the functionalities, special features and assumptions of more than 20 such programs, indicate how they can interoperate, and discuss new directions that could lead to improved software and analyses.

- Interoperability but…
  - Requires many conversion steps
  - Complicated data pathways
  - Prone to mistakes/errors
- No coding, testing, or documentation standards



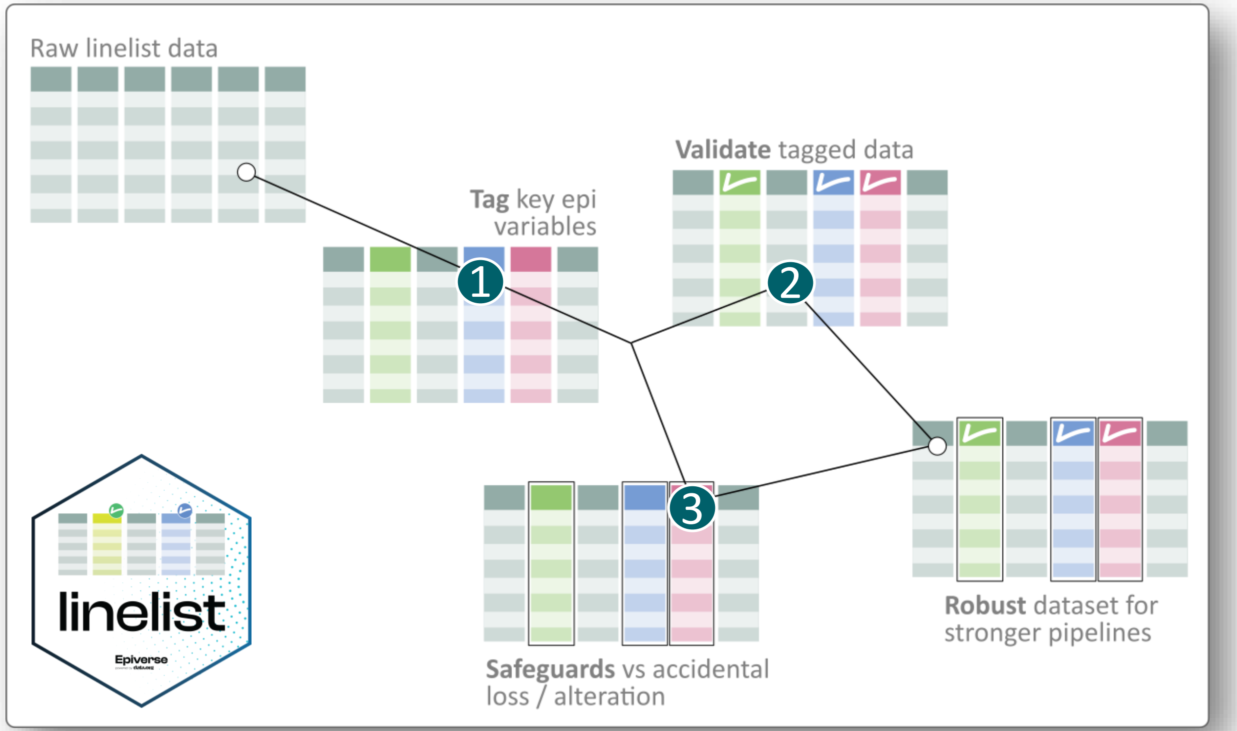
Excoffier & Heckel 2006 *Nature Reviews Genetics*

linelist: a package to handle case line list data



- Case line list data
  - Spreadsheet-like dataset
  - Rows = different patients
  - Columns = variables, including:
    - Dates of events: symptom onset, case reporting, …
    - Case data: epi case definition, disease outcome, symptoms
    - Patient info: age, gender, occupation, location, …
- Common issues
  - No standardized names for key epi variables
  - Data have the wrong type / class *e.g.:*
    - dates stored as `factors` rather than `Date`,
    - Epi case definition coded as `integer` rather explicit character strings
  - Key data may be altered or lost accidentally

linelist in a nutshell


Raw linelist data
Tag key epi variables
Validate tagged data
Safeguards vs accidental loss / alteration
Robust dataset for stronger pipelines
1
2
3
linelist
Epiverse

https://github.com/epiverse-trace/linelist

1 Tagging system

- `linelist`: S3 class extending `data.frame` or `tibble`
- Tags identify columns storing key epi variables in a dataset
- Tags are stored as named `list` in `attr(., "tags")`

2 Data validation

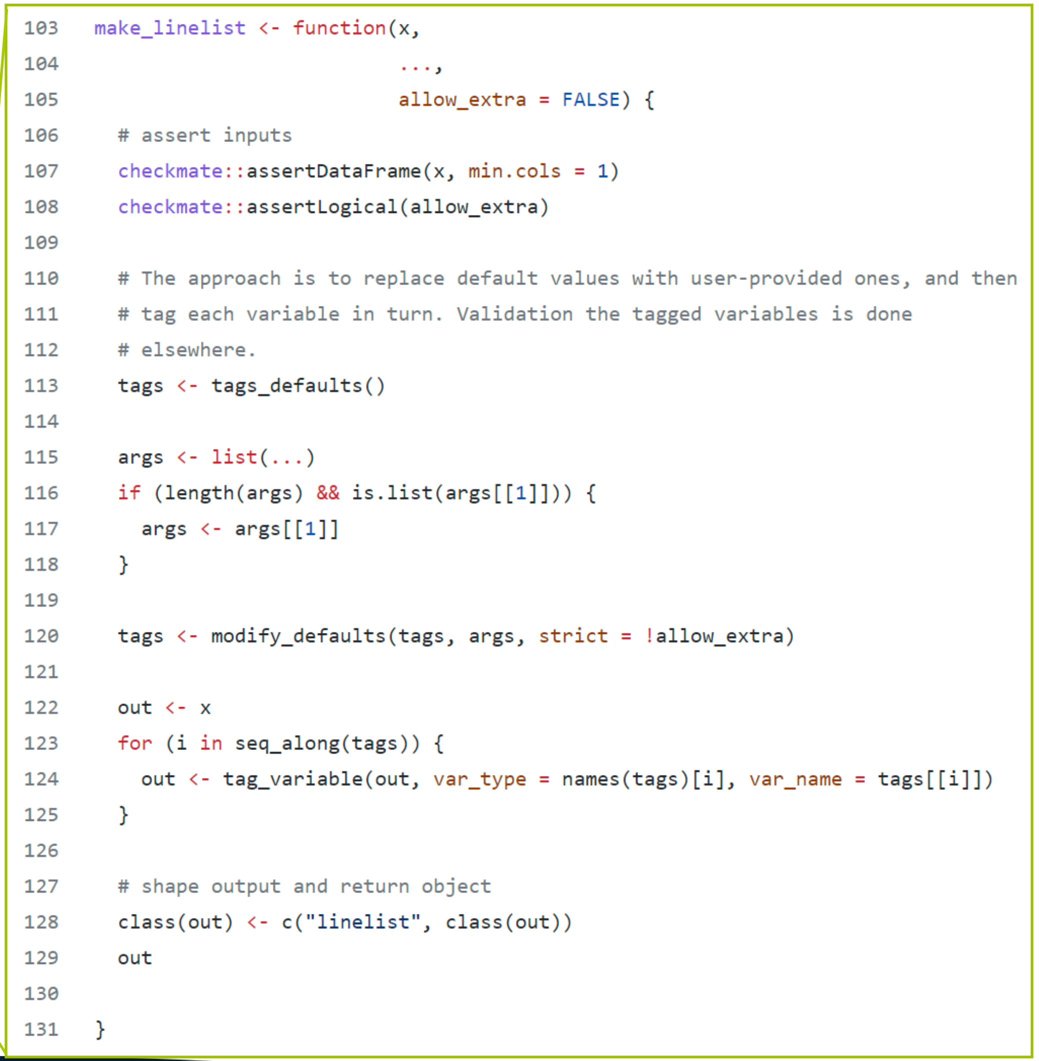
- Validate tagged variables against their expected types
- Easy to extend to more advanced validations (e.g. consistency in timing of events)

3 Safeguards

- Protections against alteration / loss of tagged variables
- Done through S3 dispatching with dedicated methods for 'dangerous' operations
- Implementation for non-generic more cumbersome

linelist

Coding practices

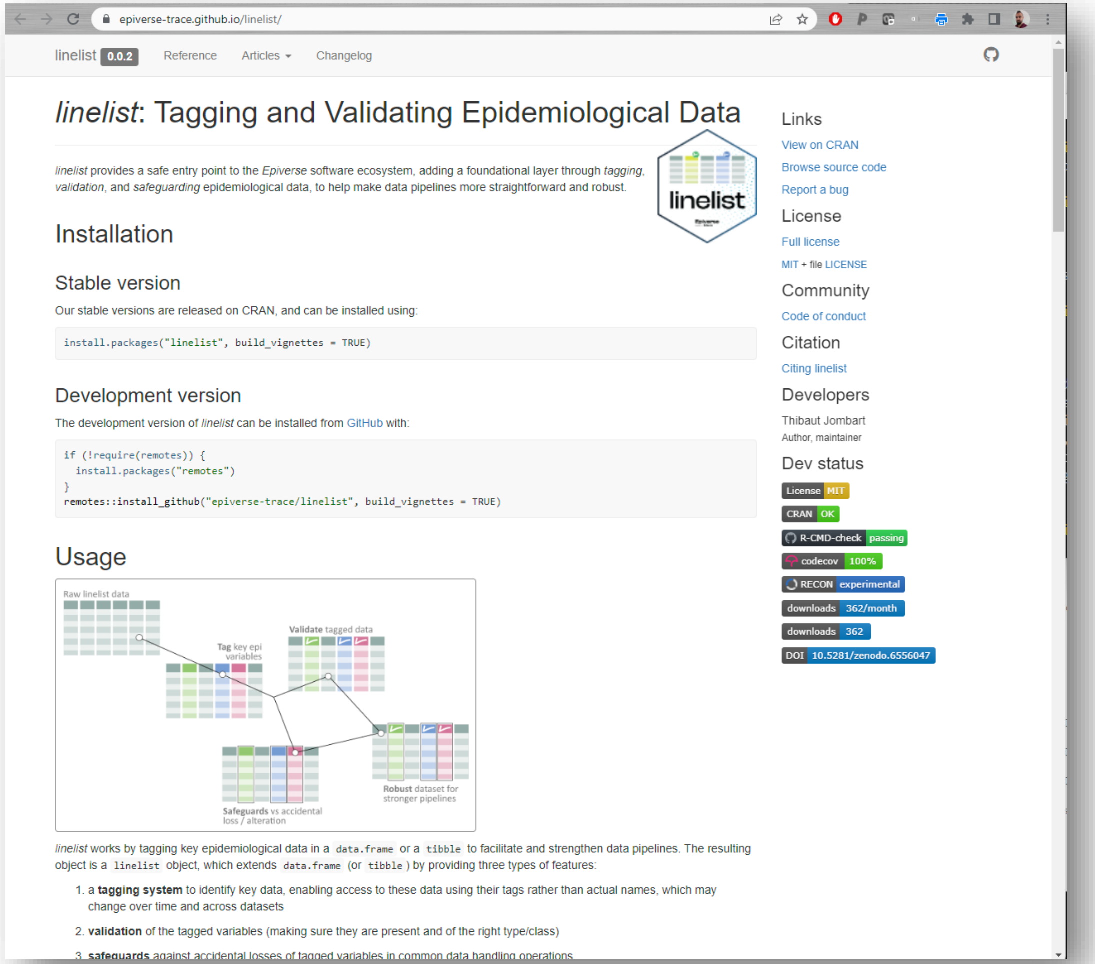


```r
make_linelist <- function(x,
                          ...,
                          allow_extra = FALSE) {
  # assert inputs
  checkmate::assertDataFrame(x, min.cols = 1)
  checkmate::assertLogical(allow_extra)

  # The approach is to replace default values with user-provided ones, and then
  # tag each variable in turn. Validation the tagged variables is done
  # elsewhere.
  tags <- tags_defaults()

  args <- list(...)
  if (length(args) && is.list(args[[1]])) {
    args <- args[[1]]
  }

  tags <- modify_defaults(tags, args, strict = !allow_extra)

  out <- x
  for (i in seq_along(tags)) {
    out <- tag_variable(out, var_type = names(tags)[i], var_name = tags[[i]])
  }

  # shape output and return object
  class(out) <- c("linelist", class(out))
  out

}
```

- Modular code, short functions
- Short argument list
- Assertion of inputs
- Comments
  - mark code structure
  - explain global strategy
  - avoid mere code reformulation
- Document as you code
- Test as you code (or before)

linelist

Documentation



epiverse-trace.github.io/linelist/

linelist 0.0.2 Reference Articles Changelog

linelist: Tagging and Validating Epidemiological Data

linelist provides a safe entry point to the *Epiverse* software ecosystem, adding a foundational layer through *tagging*, *validation*, and *safeguarding* epidemiological data, to help make data pipelines more straightforward and robust.

Installation

Stable version

Our stable versions are released on CRAN, and can be installed using:

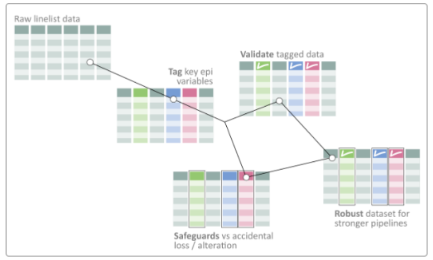
```
install.packages("linelist", build_vignettes = TRUE)
```

Development version

The development version of *linelist* can be installed from GitHub with:

```
if (!require(remotes)) {
  install.packages("remotes")
}
remotes::install_github("epiverse-trace/linelist", build_vignettes = TRUE)
```

Usage


Raw linelist data
Tag key epi variables
Validate tagged data
Safeguards vs accidental loss / alteration
Robust dataset for stronger pipelines

linelist works by tagging key epidemiological data in a `data.frame` or a `tibble` to facilitate and strengthen data pipelines. The resulting object is a `linelist` object, which extends `data.frame` (or `tibble`) by providing three types of features:

1. a **tagging system** to identify key data, enabling access to these data using their tags rather than actual names, which may change over time and across datasets
2. **validation** of the tagged variables (making sure they are present and of the right type/class)
3. **safeguards** against accidental losses of tagged variables in common data handling operations

Links
View on CRAN
Browse source code
Report a bug

License
Full license
MIT + file LICENSE

Community
Code of conduct

Citation
Citing linelist

Developers
Thibaut Jombart
Author, maintainer

Dev status
License MIT
CRAN OK
R-CMD-check passing
codecov 100%
RECON experimental
downloads 362/month
downloads 362
DOI 10.5281/zenodo.6556047

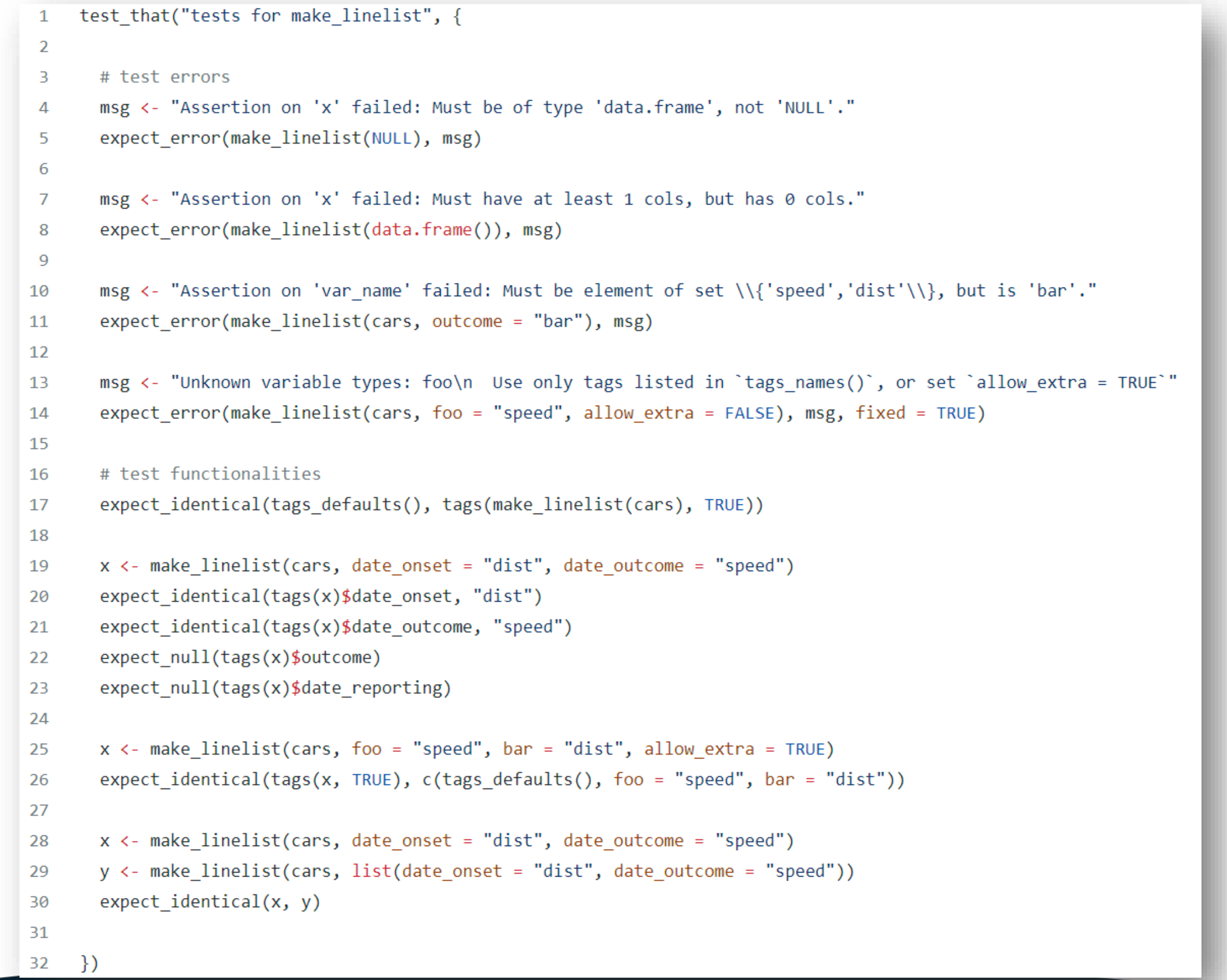
- All functions documented using roxygen2
- All exported functions have examples
- README.Rmd
- Vignette
- Website (pkgdown)
- Infographics
- Hex sticker!!
- Cheat-sheet underway

data.org

linelist

Testing and continuous integration

CRAN OK | R-CMD-check passing | codecov 100%



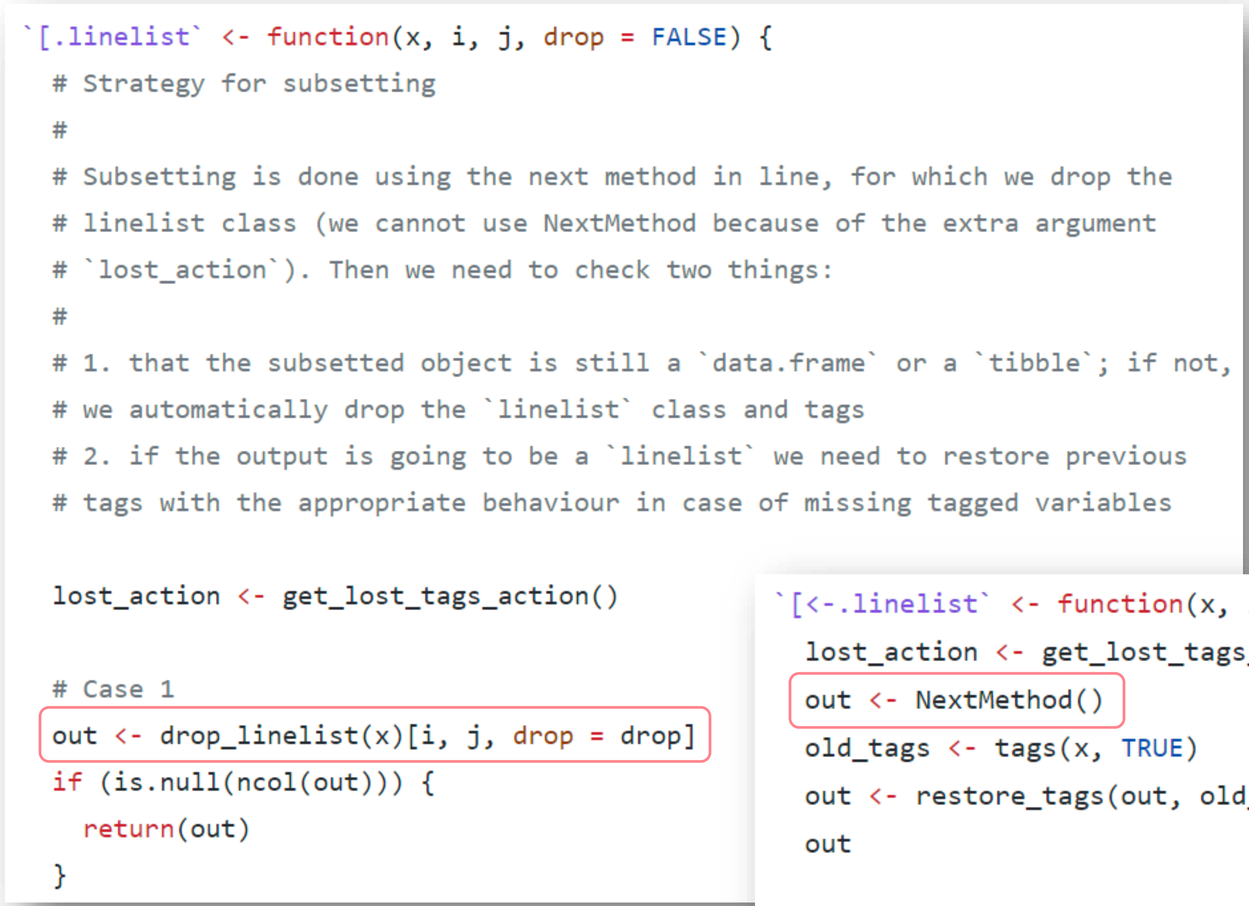
```r
test_that("tests for make_linelist", {

  # test errors
  msg <- "Assertion on 'x' failed: Must be of type 'data.frame', not 'NULL'."
  expect_error(make_linelist(NULL), msg)

  msg <- "Assertion on 'x' failed: Must have at least 1 cols, but has 0 cols."
  expect_error(make_linelist(data.frame()), msg)

  msg <- "Assertion on 'var_name' failed: Must be element of set \\{'speed','dist'\\}, but is 'bar'."
  expect_error(make_linelist(cars, outcome = "bar"), msg)

  msg <- "Unknown variable types: foo\n  Use only tags listed in `tags_names()`, or set `allow_extra = TRUE`"
  expect_error(make_linelist(cars, foo = "speed", allow_extra = FALSE), msg, fixed = TRUE)

  # test functionalities
  expect_identical(tags_defaults(), tags(make_linelist(cars), TRUE))

  x <- make_linelist(cars, date_onset = "dist", date_outcome = "speed")
  expect_identical(tags(x)$date_onset, "dist")
  expect_identical(tags(x)$date_outcome, "speed")
  expect_null(tags(x)$outcome)
  expect_null(tags(x)$date_reporting)

  x <- make_linelist(cars, foo = "speed", bar = "dist", allow_extra = TRUE)
  expect_identical(tags(x, TRUE), c(tags_defaults(), foo = "speed", bar = "dist"))

  x <- make_linelist(cars, date_onset = "dist", date_outcome = "speed")
  y <- make_linelist(cars, list(date_onset = "dist", date_outcome = "speed"))
  expect_identical(x, y)

})
```

- Full test coverage for all functions
- Each function has its own test file
- Testing guides coding
- ~ 50% tests on errors and warnings ; ~50% tests on functionalities
- Github actions for CI (`usethis`)

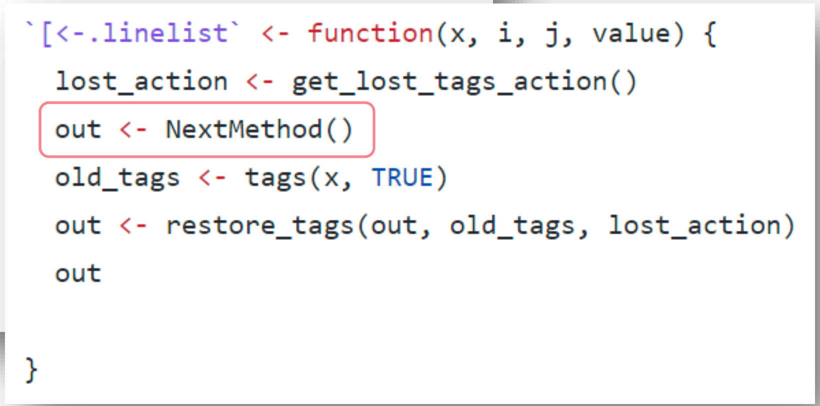
data.org

linelist

Code reviews, collaboration workflows, communities



```r
`[.linelist` <- function(x, i, j, drop = FALSE) {
  # Strategy for subsetting
  #
  # Subsetting is done using the next method in line, for which we drop the
  # linelist class (we cannot use NextMethod because of the extra argument
  # `lost_action`). Then we need to check two things:
  #
  # 1. that the subsetted object is still a `data.frame` or a `tibble`; if not,
  # we automatically drop the `linelist` class and tags
  # 2. if the output is going to be a `linelist` we need to restore previous
  # tags with the appropriate behaviour in case of missing tagged variables

  lost_action <- get_lost_tags_action()

  # Case 1
  out <- drop_linelist(x)[i, j, drop = drop]
  if (is.null(ncol(out))) {
    return(out)
  }
```



```r
`[<-.linelist` <- function(x, i, j, value) {
  lost_action <- get_lost_tags_action()
  out <- NextMethod()
  old_tags <- tags(x, TRUE)
  out <- restore_tags(out, old_tags, lost_action)
  out

}
```

Solo project is a bad example but…

- Code reviews
  - Useful, but non-trivial
  - Pair-programming better?
  - Part of Pull Requests, or separate exercise?
- Lean / Agile development
  - Important to incorporate user feedback early and often
  - Short cycles with regular production of MVPs better for user engagement?
- Dev and user community
  - How to engage end-users?
  - Metrics of community health or performance?

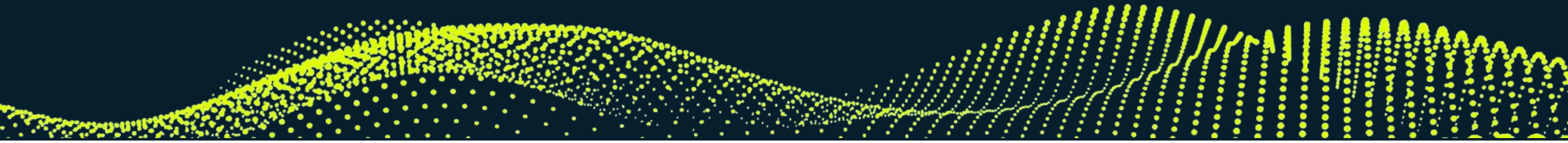
data.org

Blueprints: how many boxes to ticks?

- ❑ Coding standards
- ❑ Documentation standards
- ❑ Testing and continuous integration
- ❑ Collaboration framework
    - ❑ Code reviews
    - ❑ Pair programming
    - ❑ Agile/Lean methodologies
- ❑ Community engagement and monitoring
- ❑ …

data.org

THANK YOU



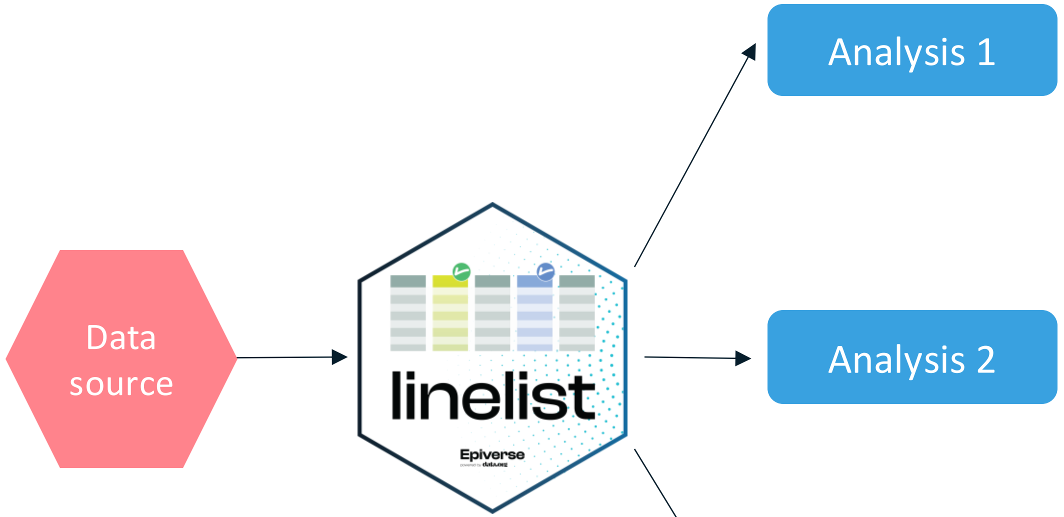
Senior RSE position for *Epiverse* open until Friday at data.org!

Contact:

Thibaut Jombart

thibaut@data.org

linelist: tagging, validating, and safeguarding key epi data



Linelist = S3 class extending `data.frame`

- Tagging system: identify key epi data
  - `set_tags()`: identify columns storing key epi variables (e.g. date of onset, disease outcome, age)
  - `select()`, `select_tags()`, `tags_df()`: access tagged variables
  - Tags are stored in the attributes of the object
- Data validation
  - Key epi variables have pre-defined acceptable types
  - `validate_linelist()`: check tags consistency and types
- Safeguards
  - Protections against alteration / loss of tagged variables
  - `lost_tags_action()`: defines behaviour to adopt
  - `rename(), names() <-`: update tags as needed
  - `select(), [, [[,` …: issues warnings/errors if variable lost

data.org

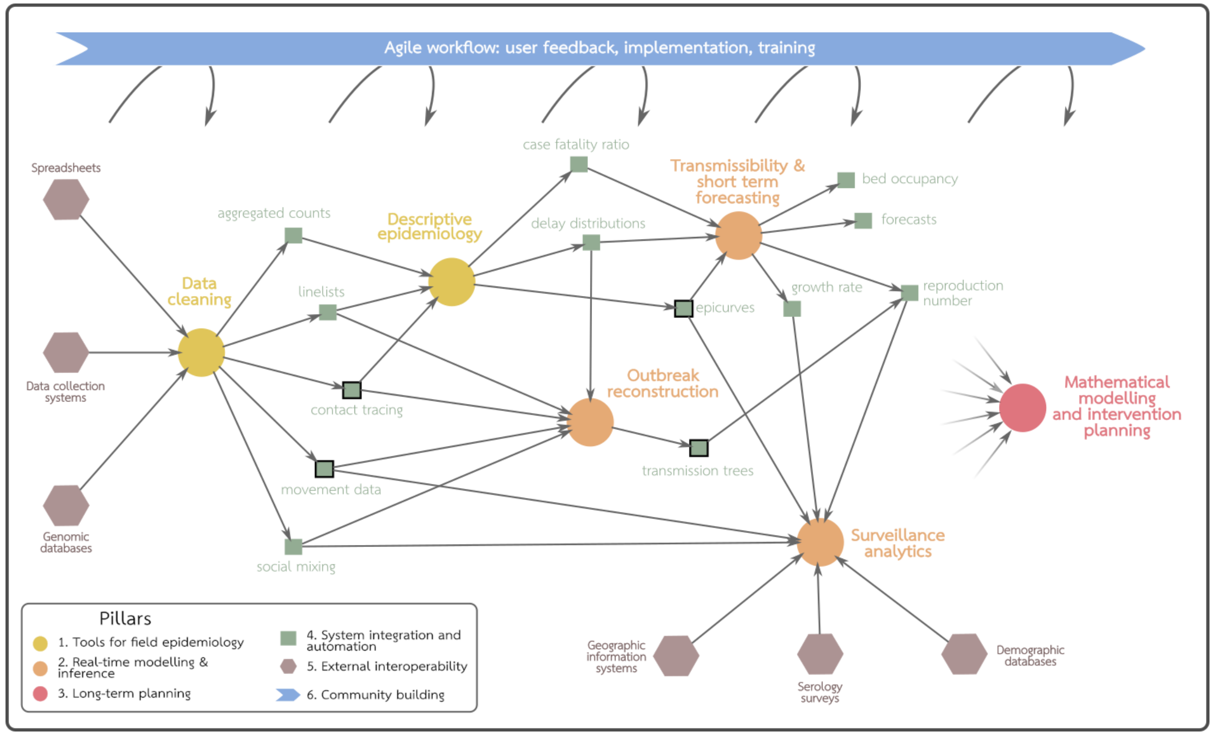
Context

Epiverse-TRACE: building a coherent ecosystem for outbreak analytics in R

- Simplest workflows rely on:

```
import_data() %>%
   preprocess_data() %>%
   use_analysis() %>%
   summarise_or_plot()
```

- Complications:
  - Lack of canonical representations
  - Multiple inputs (e.g. case counts + serial interval)
  - Multiple layers of analysis (e.g. estimate R -> make projections)



Agile workflow: user feedback, implementation, training
Spreadsheets
Data collection systems
Genomic databases
Data cleaning
aggregated counts
linelists
contact tracing
movement data
social mixing
Descriptive epidemiology
case fatality ratio
delay distributions
epicurves
Transmissibility & short term forecasting
bed occupancy
forecasts
growth rate
reproduction number
Outbreak reconstruction
transmission trees
Mathematical modelling and intervention planning
Surveillance analytics
Geographic information systems
Serology surveys
Demographic databases
Pillars
1. Tools for field epidemiology
2. Real-time modelling & inference
3. Long-term planning
4. System integration and automation
5. External interoperability
6. Community building