Package: starling (via r-universe)

May 27, 2026

Type Package

Title Link Infectious Disease Cases to Vaccination and Hospitalization Records

Version 0.6.5

Description Facilitates probabilistic record linkage between infectious disease surveillance datasets (notifiable disease registers, outbreak line-lists), vaccination registries, and hospitalization records using methods based on Fellegi and Sunter (1969) <doi:10.1080/01621459.1969.10501049> and Sayers et al. (2016) <doi:10.1093/ije/dyv322>. The package provides core functions for data preparation, linkage, and analysis: clean_the_nest() standardizes variable names and formats across heterogeneous datasets; murmuration() performs machine learning-based record linkage using blocking variables and similarity metrics; molting() deidentifies datasets for secure sharing; homing() re-identifies previously deidentified datasets; plumage() identifies and categorizes comorbidities; and preening() creates analysis-ready variables including age categories and temporal groupings. Designed for epidemiological research linking acute and post-acute disease outcomes to vaccination status and healthcare utilization. Supports multiple linkage scenarios including case-to-vaccination, case-to-hospitalization, and event-based vaccination status determination (e.g., outbreak attendees, flight passengers, exposure site visitors).

License GPL (>= 3)

Depends R (>= 3.5.0)

Imports dplyr, lubridate, janitor, stringr, tidyr, reclin2, datawizard, digest, rlang, magrittr

Suggests testthat (>= 3.0.0), knitr, gtsummary, rmarkdown

Encoding UTF-8

LazyData true

RoxygenNote 7.3.2

VignetteBuilder knitr

Language en-US

NeedsCompilation no

Author Nicolas Smoll [aut, cre] (ORCID:
<https://orcid.org/0000-0002-6923-9701>)

Maintainer Nicolas Smoll <nrsmoll@gmail.com>

Config/pak/sysreqs libicu-dev

Repository https://nrsmoll.r-universe.dev

Date/Publication 2026-01-26 16:20:08 UTC

RemoteUrl https://github.com/cran/starling

RemoteRef HEAD

RemoteSha edbc6f4aae3c5ddf79247fbe0be265629b6d9e85

Contents

`clean_the_nest` *Clean datasets and establishes common variable name nomenclature*

Description

Cleans three dataset types and prepares them for data-linkage. This command is the first step in creating the datasets for analysis. Building a solid "nest" is akin to building a solid foundation for future work. Of note, Starlings are cavity nesters, meaning that they prefer to build their homes inside holes and crevices. This command is meant to work with diagnosis datasets (linelists like Notifiable Conditions registers) and, hospitalization datasets (administrative datasets), and vaccination datasets. This command is used to prepare datasets for linkage with `murmuration`. There are no mandatory variables to include. However, a dataset of infections would include at minimum an onset date (date of diagnosis), a dataset of admissions would include admission dates, and a dataset of vaccinations would include dates of vaccination and type of vaccines. All of the datasets should

include information that would allow for data-linkage, such as first name, last name, date of birth, address etc etc.

Classic workflow would be:

1. `clean_the_nest` to clean and prep data for linkage. Pay close attention to your linkage variables (letternames, date of birth, medicare number, gender and/or postcode), and ensure all dates are formatted as dates.
2. `murmuration` to link cases to vaccination data (named here "c2v").
3. `murmuration` to link c2v to hospitalization data (named here c2v2h). Of note, you can skip linking the vaccination dataset.
4. `preening` to prettify the dataframe prepping it for exploration, analysis and presentation. Great to use with `gtsummary::tbl_summary()`.

Usage

```
clean_the_nest(
  data,
  id_var = NULL,
  event_id_var = NULL,
  drop_eggs = FALSE,
  data_type = NULL,
  lie_nest_flat = FALSE,
  drop_the_na_vax = TRUE,
  keep_vars = NULL,
  diagnosis = NULL,
  lettername1 = NULL,
  lettername2 = NULL,
  dob = NULL,
  age = NULL,
  medicare = NULL,
  postcode = NULL,
  gender = NULL,
  fn = NULL,
  latitude = NULL,
  longitude = NULL,
  onset_date = NULL,
  vax_type = NULL,
  vax_date = NULL,
  lag = 0,
  admission_date = NULL,
  discharge_date = NULL,
  hospital = NULL,
  icd_code = NULL,
  diagnosis_description = NULL,
  drg = NULL,
  icu_date = NULL,
  icu_hours = NULL,
  dialysis = NULL,
```

```
  genomics = NULL,
  dod = NULL,
  died = NULL
)
```

Arguments

data	The dataset, which can be a case notifications dataset (infections), hospital admissions or vaccination dataset (must pre-specify if it is a vaccinations dataset). Make sure dates are in date format.
id_var	Any format as long as unique to individual. This is important This ID variable is critical. Must ensure for case data that it only has one row per person, or first infection only. Identifies the multiple rows associated with a person who has multiple vaccines, admissions or infections. Cannot have missing data, or the observation will be lost in the linking process.
event_id_var	Any format as long as unique for the whole dataset. This represents the ID of the vaccination event, or the hospitalization event, which MUST be distinct. A person (id_var) can have multiple events (event_id). Some datasets will surprise you with multiple entries for the same admission.
drop_eggs	This effectively drops the variables that are not being used. May turn this off if you need lots of extra information, but certainly good for the early stages of an analysis. Enables a lean dataset.
data_type	Three options: "vaccination", "hospital", or "cases". The key information required is that for linkage, and the vaccination events. No age or age categories will be calculated if it is a vaccination dataset.
lie_nest_flat	Takes a long vaccination dataset (like Australian Immunization Register; 1 or more rows per person) and turns it into a wide dataset - one row per person
drop_the_na_vax	Drops (removes) vaccines that are listed as having no names.
keep_vars	Vector list of variables. Variables in a vector list with quotation marks, as it will be used in a select statement.
diagnosis	Character format. The column with the infectious disease diagnosis listed. e.g. COVID-19, SARS-CoV-2, RSV, Influenza.
lettername1	Character format. First Name variable. If there is a second first name (some cases this might be a middle name), it will be removed during cleaning. All non-alphanumeric characters will be removed and everything becomes lower case.
lettername2	Character format. Last name variable. All non-alphanumeric characters will be removed and everything becomes lower case. Two part last names will be kept.
dob	Date format. The date of birth (make sure dates are in date format).
age	Numeric format. Include age only if it has been pre-specified in the dataset, and you don't want it re-calculated.
medicare	Numeric format. Medicare number. A medicare number with 9, 10 and 11 numbers will have been created. In Australia, the 10th number represents the card ID, and the 11th number represents the person ID. A family or individual will get a new card id (10th digit) every time their card expires.

postcode	Numeric format. Post code of person with no restriction on the number of digits.
gender	Character format. Pay close attention that your genders are in a similar format for data-linkage - "F", vs "0" vs "Female". This is left up to the user to clean.
fn	Character format. First Nations Status.
latitude	Numeric format. Latitude of address. Not explicitly required for linkage.
longitude	Numeric format. Longitude of address. Not explicitly required for linkage.
onset_date	Date format. Onset date of the illness. Commonly the date of diagnosis (date of the lab test or date of the first symptom). Must be in date format.
vax_type	Character format. Variable that indicates the vaccine type, brand, or antigen
vax_date	Date format. Variable that indicates the vaccination event date. Make sure is in date format, and arranged in order of dates you would like it to appear when it goes to wide format. For example, if it is not in order, vax_date_1 (an output variable) may be the latest vaccination date, instead of the first.
lag	Numeric format. Number of days to add to the vaccination event date. Useful to define when a person reaches peak immunity post-vaccination. For COVID-19 this is often thought to be 14 days. Default lag is zero days.
admission_date	Date format. Admission date variable. Typically, this should be later than the date of onset, but there are times when the disease is diagnosed in hospital.
discharge_date	Date format. Discharge date variable. This date should be later than the date of admission.
hospital	Hospital identifier. Typically name of the hospital.
icd_code	Character format. ICD code variable for the admission. No pre-specified format required.
diagnosis_description	Character format. Written description of the ICD code. For ease of understanding what the ICD codes mean, not a critical variable.
drg	Character format. Diagnostic related group variable for the admission. No pre-specified format required.
icu_date	Date format. ICU admission date preferably. Typically, this should be later than the date of onset and admission, but there are times when the disease is diagnosed in ICU.
icu_hours	ICU hours. Hours spent in ICU. Should be numeric.
dialysis	Dialysis indicator (0/1).
genomics	Character format. Genomics variable. Can be variant of SARS-CoV-2, or similarly the Hepatitis A.
dod	Date format. Variable representing date of death. Must only have one date of death chosen (in diagnosis dataset or hospitalization dataset, not both). If dod selected is from the hospitalization dataset, it will be deleted for persons without an admission.
died	Variable representing death, best use 0 and 1.

Value

The output is a dataframe that is cleaned and could be ready for machine learning data-linkage.

Examples

```
# Basic usage of clean_the_nest.
# Use this to set up for datalinkage using the murmuration command and then cleaning with preening
data(dx_data)
df_diag <- clean_the_nest(dx_data, drop_eggs=TRUE, data_type = "cases",
  id_var ="identity",
  diagnosis = "disease_name",
  lettername1 = "first_name",
  lettername2 = "surname",
  dob = "date_of_birth",
  medicare = "medicare_no",
  gender = "gender",
  postcode="postcode",
  fn="indigenous_status",
  onset_date = "diagnosis_date")

data(hosp_data)
df_hosp <- clean_the_nest(hosp_data, drop_eggs=TRUE,
  data_type = "hospital",
  id_var ="patient_id",
  lettername1 = "firstname",
  lettername2 = "last_name",
  dob = "birth_date",
  medicare = "medicare_number",
  gender = "sex",
  postcode="zip_codes",
  fn="cultural_heritage",
  icd_code = "icd_codes",
  admission_date = "date_of_admission",
  discharge_date = "date_of_discharge")

data(vax_data)
df_vax <- clean_the_nest(data = vax_data,
  data_type = "vaccination",
  lie_nest_flat=TRUE,
  id_var = "patient_id",
  lettername1="firstname",
  lettername2="last_name",
  dob="birth_date",
  medicare="medicare_number",
  gender = "gender",
  postcode = "postcode",
  vax_type = "vaccine_delivered",
  vax_date = "service_date")
```

`dx_data` *Example Diagnosis Dataset*

Description

A sample dataset containing diagnosis records for 10 individuals with respiratory illnesses (Influenza A/B, COVID-19, RSV).

Usage

```
dx_data
```

Format

A data frame with 10 rows and 10 variables:

identity Patient identifier

disease_name Name of diagnosed disease

first_name Patient's first name

surname Patient's surname

date_of_birth Patient's date of birth

medicare_no Medicare number

gender Gender (M/F)

postcode Postcode of residence

indigenous_status Indigenous status

diagnosis_date Date of diagnosis

Examples

```
data(dx_data)
head(dx_data)
```

`homing` *Homing: Relink De-identified Data Using Lookup Table*

Description

Like homing pigeons finding their way back, this function relinks de-identified data with original identifiers using the lookup table created by molting().

Usage

```
homing(
  deidentified_data,
  lookup_table,
  hash_col_name = "row_hash",
  keep_hash = TRUE
)
```

Arguments

deidentified_data	A de-identified data frame containing a hash column (typically the output from molting()$deidentified).
lookup_table	The lookup table data frame that maps anonymous hash values back to original identifiers. Created by molting(), it contains the hash column plus all removed identifier columns (names, dates of birth, medical record numbers, etc.). This serves as the secure "key" for relinking de-identified data back to real identities. Each row maps one hash to one set of identifiers. Typically obtained as molting()$lookup. Example structure for a dataset that had patient_name, dob, and mrn removed: row_hash | patient_name | dob | mrn.
hash_col_name	The name of the hash column used for linking. Must exist in both deidentified_data and lookup_table. Defaults to "row_hash".
keep_hash	Logical. If TRUE (default), keeps the hash column in the relinked data. If FALSE, removes it after relinking.

Value

A data frame with the original identifiers merged back in.

Examples

```
## Not run:
# First, de-identify the data
result <- molting(patient_data)

# Later, relink when needed
original_data <- homing(
  result$deidentified,
  result$lookup
)

## End(Not run)
```

---

hosp_data *Example Hospitalization Dataset*

Description

A sample dataset containing hospitalization records for 10 individuals, with 3 overlapping with dx_data.

Usage

```
hosp_data
```

Format

A data frame with 10 rows and 11 variables:

patient_id Patient identifier
firstname Patient's first name
last_name Patient's last name
birth_date Patient's date of birth
medicare_number Medicare number with check digit
sex Sex (1=Male, 2=Female)
zip_codes Postcode
cultural_heritage Indigenous status
icd_codes ICD-10 diagnosis code
date_of_admission Hospital admission date
date_of_discharge Hospital discharge date

Examples

```
data(hosp_data)
head(hosp_data)
```

---

linelist_data *Example Outbreak Linelist Dataset*

Description

A sample dataset containing measles outbreak cases associated with a music festival on 2024-06-01.

Usage

```
linelist_data
```

Format

A data frame with 10 rows and 10 variables:

case_id Case identifier
first_name Patient's first name
surname Patient's surname
date_of_birth Patient's date of birth
medicare_no Medicare number
gender Gender (M/F)
postcode Postcode of residence
onset_date Date of symptom onset
notification_date Date case was notified
case_classification Case classification (Confirmed/Probable)

Examples

```
data(linelist_data)
head(linelist_data)
```

`manifest_data` *Example Flight Manifest Dataset*

Description

A sample dataset containing passenger manifest for flight QF123 on 2024-03-15, used to demonstrate event-based vaccination linkage.

Usage

```
manifest_data
```

Format

A data frame with 8 rows and 8 variables:

passenger_id Passenger identifier

first_name Passenger's first name

surname Passenger's surname

date_of_birth Passenger's date of birth

gender Gender (M/F)

seat_number Seat assignment

flight_number Flight number

flight_date Date of flight

Examples

```
data(manifest_data)
head(manifest_data)
```

molting *Molt: De-identify a Dataset with Hash-based Relinking*

Description

Like a bird molting its feathers for new plumage, this function removes identifiable information and replaces it with a unique hash for each row. It returns both the de-identified dataset and a lookup table for relinking. Age category variables (age2cat, age3cat, etc.) are automatically retained.

Usage

```
molting(
  data,
  id_cols = NULL,
  pii_patterns = NULL,
  additional_pii_cols = NULL,
  hash_method = "sha256",
  hash_col_name = "row_hash",
  return_lookup = TRUE,
  seed = NULL
)
```

Arguments

data	A data frame to be de-identified.
id_cols	An optional character vector of column names to use for creating the hash. If NULL (the default), the function will use the PII columns it automatically detects.
pii_patterns	An optional character vector of regular expression patterns used to detect PII columns for removal. The default list includes common identifiers.
additional_pii_cols	An optional character vector of specific column names to remove as PII, in addition to those detected by pattern matching. Useful for adding dataset-specific identifiers without modifying patterns.
hash_method	The hashing algorithm to use. Options include "sha256" (default), "md5", "sha1", "sha512", "crc32", "xxhash32", "xxhash64", "murmur32", "spookyhash", or "blake3". See ?digest::digest for details.
hash_col_name	A string for the name of the new hash column. Defaults to "row_hash".
return_lookup	Logical. If TRUE (default), returns a list containing both the de-identified data and a lookup table. If FALSE, returns only the de-identified data frame.
seed	An optional integer seed for reproducible hashing with certain algorithms. Defaults to NULL.

Details

The function identifies PII columns based on pattern matching, creates a unique hash for each row based on the concatenated identifier values, and returns both a de-identified dataset and a secure lookup table.

Age category variables (variables matching the pattern "age\d+cat" such as age2cat, age5cat, age10cat, etc.) are automatically retained in the de-identified dataset as they are not considered directly identifying.

Security Note: The lookup table contains sensitive information and should be stored securely with appropriate access controls. Consider encrypting this file if storing to disk.

Value

If return_lookup = TRUE (default), a list with two elements:

- `deidentified`: The de-identified data frame with hash column
- `lookup`: A data frame containing only the identifier columns and the hash for relinking

If return_lookup = FALSE, returns only the de-identified data frame.

Examples

```
# Create sample data
patient_data <- data.frame(
  patient_name = c("John Doe", "Jane Smith"),
  dob = as.Date(c("1980-01-01", "1975-05-15")),
  mrn = c("12345", "67890"),
  age5cat = factor(c("18-64", "18-64")),
  diagnosis = c("Condition A", "Condition B"),
  lab_value = c(120, 95)
)

# Basic de-identification (age categories automatically retained)
result <- suppressMessages(molting(patient_data))
names(result$deidentified)  # Check column names
head(result$deidentified, 2)  # View de-identified data

# Use different hash method
result_md5 <- suppressMessages(
  molting(patient_data, hash_method = "md5")
)

# Return only de-identified data (no lookup table)
deidentified_only <- suppressMessages(
  molting(patient_data, return_lookup = FALSE)
)

# Add specific columns to PII removal
result_custom <- suppressMessages(
  molting(patient_data, additional_pii_cols = c("study_id"))
)
```

```
# Specify custom identifier columns for hashing
result_ids <- suppressMessages(
  molting(patient_data, id_cols = c("mrn", "dob"))
)
```

`murmuration` *Links case, hospital or vaccination datasets*

Description

Machine learning data linkage. The murmuration command will link diagnostic registry data (cases or linelist) to hospitalization and immunization records (e.g. Australian Immunization Register).

Usage

```
murmuration(
  df1,
  df2,
  linkage_type = "c2h",
  onset_date = NULL,
  event_date = NULL,
  id_var = id_var,
  blocking_var,
  compare_vars,
  threshold_value = 12,
  days_allowed_before_event = 7,
  days_allowed_after_event = 14,
  one_row_per_person = TRUE,
  clean_eggs = TRUE,
  days_between_onset_death = 30,
  last_follow_up = NULL
)
```

Arguments

`df1`	This is a dataframe object, cleaned using clean_build_nest, and would often represent the base, or "x" dataset (when doing left joins). Typically this would be a dataset of cases, have enough data to create linkages, and have onset dates.
`df2`	This is a dataframe object, cleaned using clean_build_nest, and would often represent the admissions or vaccination dataset ("y" dataset when doing left joins). Typically this would have enough data to create linkages, and include either admission data or vaccination event data (e.g. Australian Immunization Register).
`linkage_type`	Parameter name. Either "c2h", for linkage of cases to hospital admissions data (default). "v2c" for linkage of cases to vaccination datasets. "v2h" for linkage of hospitalizations to vaccination history (e.g. building a dataset for test-negative

case-control studies). Use "v2h" if you want to link a "v2c" dataset to a hospitalization dataset. "v2e" for linkage of event participants (flight manifest, outbreak linelist) to vaccination history to determine vaccination status at time of event. If using linking to a vaccination dataset, must use single row per person dataset. If you have multiple vaccines per person, run it through the clean_the_nest command with "lie_nest_flat" option set to TRUE.

`onset_date` Variable name for onset date (used in c2h and v2c linkage types). Should be present in df1.

`event_date` A date object (e.g., ymd('2024-12-15')) representing when the event occurred. Required for v2e linkage type. All valid vaccinations must occur before this date.

`id_var` Variable name (e.g. "id")This is critical for data-linkage and the base dataset is the dataset you would left join onto (e.g. the "x" dataset). Cannot have missing data, or the observation will be lost in the linking process.

`blocking_var` Variable name (e.g. "block1"). Choice of blocking variable. You can create your own. Up to three blocking vars are created in the past

`compare_vars` Vector of variables. Used to compare variables between each dataset and calculate the string score differences. Typically names, dates of births and medicare/social security numbers.

`threshold_value`
Numeric (e.g. "12"), default is 12. This represents the threshold above which you decide that the linkage is true or false. The higher the number, the higher the specificity of your linkages (compare_vars match more exactly). The lower the threshold, the more sensitive you are to selecting matches, at the expense of specificity. Default is 12, and arbitrarily chosen.

`days_allowed_before_event`
Numeric (e.g. "7"). How much time you choose to allow prior to the onset_date of a disease-related admission for a c2h dataset (see linkage type). For c2h linkages, this represents the lower limit of the window for disease related admissions. For v2h datasets this represents the minimum time between latest vaccination date and admission date to be considered a valid vaccination dose. For a v2c datasets this represents the minimum time between latest vaccination date and onset_date to be considered a valid vaccination dose. For v2e datasets this represents the minimum time between latest vaccination date and event_date to be considered a valid vaccination dose. For example, if you choose seven days, then you are allowing for an admission to occur up to seven days prior to the diagnosis, which means the disease was diagnosed while an inpatient.

`days_allowed_after_event`
Numeric (e.g. "30"). How much time you choose to allow after the onset of a disease related admission. Upper limit of window for disease related admissions. For example, if you choose 30 days, then you are allowing for a disease-related admission to occur up to 30 days after the diagnosis, which means the disease was diagnosed very close to or prior to the admission.

`one_row_per_person`
Logical (TRUE or FALSE) with the default being TRUE. It will take multiple admissions per person, and create a series of variables prefixed with "first_",

such as "first_admission_date", and put into a single row all admission events, and create a series of variables suffixed with "s", such as "admission_dates". Will work with single admissions per person.

`clean_eggs` Logical (TRUE or FALSE) with the default being TRUE. Drops all the .y variables that are duplicates of the second dataset (df2), and keeps the variables and removes the .x from df1. If you leave this on, many, if not most variables will have ".x" or ".y" attached to them (e.g. gender) and thus keep this as TRUE for default, and FALSE if you want to check the linkages are true and working.

`days_between_onset_death`
Numeric (e.g. "30"). If you have put a date of death into the clean_build_nest command (which will rename it to dod), then the command will find disease related dates of death. This is chosen number of days between onset and death for a disease-related death. Often this may be 30 days for SARS-CoV-2 or can be much longer for HIV. If you don't want an upper limit, use "9999".

`last_follow_up` represents a date (input as ymd(2024-11-22)) that represents last follow-up. This could be the latest admission date of a dataset. Used for calculating survival time.

Details

A murmuration is a shape-shifting flock of thousands of starlings all flying in synch with each other. Murmuration means that each bird must be linked (through observation of their movements) to approximately seen other birds to achieve the beautiful sky art that moves through the sky. Make sure that you do not have the same variables (other than linkage variables e.g. letternames, DOB, gender) in both datasets. Always make sure your date columns are properly formatted using as_date, or as.Date. For example, if both datasets have date of death, choose the dataset with the highest confidence, and drop out the date of death from the other dataset. If the dataset is being linked to a hospitalization dataset, the difference in time between onset_date and admission_date will be used to identify related hospitalizations. The user can filter out unrelated hospitalizations using diagnostic-related groups or ICD-10 codes separately, prior to linkage. Classic workflow would be:

1. `clean_the_nest` to clean and prep data for linkage. Pay close attention to your linkage variables (letternames, date of birth, medicare number, gender and/or postcode), and ensure all dates are formatted as dates.
2. `murmuration` with linkage_type="v2c" to link cases to vaccination data.
3. `murmuration` with linkage_type="v2h" to link a v2c dataset to hospitalization data. Or skip linking to case data, and just build a v2h dataset for test-negative case-control studies.
4. `murmuration` with linkage_type="v2e" to link event linelists (flight manifests, outbreak investigations) to vaccination data.
5. `preening` to prettify the dataframe prepping it for exploration, analysis and presentation. Great to use with `gtsummary::tbl_summary()`.

Value

A linked dataset with some new variables.

Note

Ensure there are no missing vaccination dates in vaccination dataset prior to murmuration. Murmuration requires complete vaccination data (equal date and type columns per observation) to achieve correct matching of vaccination columns. If there are too few variables to match on, then matching will not work well. For example, if you have first name, last name and date of birth, and a very large dataset (Immunization Register), then the scoring will not differentiate true from false matches. Consider deterministic linkage when there is a paucity of information to use to derive linkage scores.

Examples

```
# Example 1: Link cases to vaccination history
# First, clean the datasets to standardize column names
dx_clean <- clean_the_nest(dx_data,
  data_type = "cases",
  id_var = "identity",
  lettername1 = "first_name",
  lettername2 = "surname",
  dob = "date_of_birth",
  gender = "gender",
  postcode = "postcode",
  medicare = "medicare_no",
  diagnosis = "disease_name")

vax_clean <- clean_the_nest(vax_data,
  data_type = "vaccination",
  id_var = "patient_id",
  lettername1 = "firstname",
  lettername2 = "last_name",
  dob = "birth_date",
  gender = "gender",
  postcode = "postcode",
  medicare = "medicare_number",
  vax_type = "vaccine_delivered",
  vax_date = "service_date")

# Now link cases to vaccination history
df1 <- murmuration(dx_clean, vax_clean,
  linkage_type = "v2c",
  blocking_var = "gender",
  compare_vars = c("lettername1", "lettername2", "dob"),
  clean_eggs = FALSE)

# Example 2: Link hospitalization data to vaccination history
hosp_clean <- clean_the_nest(hosp_data,
  data_type = "hospital",
  id_var = "patient_id",
  lettername1 = "firstname",
  lettername2 = "last_name",
  dob = "birth_date",
  gender = "sex",
```

```
  postcode = "zip_codes",
  medicare = "medicare_number",
  admission_date = "date_of_admission",
  discharge_date = "date_of_discharge")

df2 <- murmuration(hosp_clean, vax_clean,
  linkage_type = "v2c",
  blocking_var = "gender",
  compare_vars = c("lettername1", "lettername2", "medicare10", "dob"),
  clean_eggs = FALSE,
  one_row_per_person = TRUE)

# Example 3: Link flight manifest to vaccination history
manifest_clean <- clean_the_nest(manifest_data,
  data_type = "cases",
  id_var = "passenger_id",
  lettername1 = "first_name",
  lettername2 = "surname",
  dob = "date_of_birth",
  gender = "gender")

df_flight <- murmuration(manifest_clean, vax_clean,
  linkage_type = "v2e",
  event_date = as.Date("2024-03-15"),
  blocking_var = "gender",
  compare_vars = c("lettername1", "lettername2", "dob"),
  days_allowed_before_event = 14,
  clean_eggs = FALSE)

# Example 4: Link outbreak linelist to vaccination history
linelist_clean <- clean_the_nest(linelist_data,
  data_type = "cases",
  id_var = "case_id",
  lettername1 = "first_name",
  lettername2 = "surname",
  dob = "date_of_birth",
  gender = "gender",
  postcode = "postcode",
  medicare = "medicare_no",
  onset_date = "onset_date")

df_outbreak <- murmuration(linelist_clean, vax_clean,
  linkage_type = "v2e",
  event_date = as.Date("2024-06-01"),
  blocking_var = "postcode",
  compare_vars = c("lettername1", "lettername2", "dob", "medicare10"),
  days_allowed_before_event = 7,
  clean_eggs = FALSE)
```

---

plumage *Identify Chronic Conditions Using ICD-10-AM U-Codes*

Description

Analyzes a hospitalization dataset to identify chronic conditions based on ICD-10-AM U-codes. Like identifying a bird by its distinctive plumage (feathers), this function identifies patients by their chronic condition patterns. Creates binary indicators for each condition and calculates total condition counts by category.

Usage

```
plumage(df, icd_column, prefix = NULL, decimal = TRUE, drop_eggs = FALSE)
```

Arguments

df	A data frame containing hospitalization records
icd_column	Character string specifying the name of the column containing ICD-10-AM codes
prefix	Optional character string to prefix all output column names (default: NULL)
decimal	Logical indicating whether to match U-codes with decimal points (TRUE, default) or without decimal points (FALSE). When TRUE, matches "U78.1" format; when FALSE, matches "U781" format.
drop_eggs	Logical indicating whether to drop individual condition columns and retain only summary columns. Default is FALSE.

Details

This function identifies chronic conditions from ICD-10-AM U-codes (Australian modification codes for chronic conditions). The function recognizes the following conditions:

Metabolic/Endocrine: * U78.1: Obesity * U78.2: Cystic fibrosis

Mental Health: * U79.1: Dementia * U79.2: Schizophrenia * U79.3: Depression * U79.4: Intellectual/developmental disability

Neurological: * U80.1: Parkinson's disease * U80.2: Multiple sclerosis * U80.3: Epilepsy * U80.4: Cerebral palsy * U80.5: Paralysis

Cardiovascular: * U82.1: Ischaemic heart disease * U82.2: Heart failure * U82.3: Hypertension

Respiratory: * U83.1: Emphysema * U83.2: COPD * U83.3: Asthma * U83.4: Bronchiectasis * U83.5: Respiratory failure

Gastrointestinal: * U84.1: Crohn's disease * U84.2: Ulcerative colitis * U84.3: Liver failure

Musculoskeletal: * U86.1: Rheumatoid arthritis * U86.2: Osteoarthritis * U86.3: Systemic lupus erythematosus * U86.4: Osteoporosis

Renal: * U87.1: Chronic kidney disease

Congenital: * U88.1: Spina bifida * U88.2: Down syndrome

The function searches for these codes within the specified ICD column and creates binary indicators for each condition. It also calculates summary measures including total conditions overall and by disease category.

Note: Cystic fibrosis (U78.2) is counted in both metabolic and respiratory categories.

Value

Returns the input data frame with additional columns: * Binary indicators (0/1) for each chronic condition, optionally prefixed (unless drop_eggs = TRUE) * total_conditions: Sum of all identified conditions * total_metabolic_conditions: Sum of metabolic/endocrine conditions * total_mental_health_conditions: Sum of mental health conditions * total_neurological_conditions: Sum of neurological conditions * total_cardiovascular_conditions: Sum of cardiovascular conditions * total_respiratory_conditions: Sum of respiratory conditions * total_gastrointestinal_conditions: Sum of gastrointestinal conditions * total_musculoskeletal_conditions: Sum of musculoskeletal conditions * total_renal_conditions: Sum of renal conditions * total_congenital_conditions: Sum of congenital conditions * conditions_category: Factor with levels "0", "1", "2", "3+" based on total_conditions

Examples

```
# Create sample hospitalization data
hospital_data <- data.frame(
  patient_id = 1:4,
  icd_codes = c(
    "K29.70",
    "U78.1, U83.2, U82.3",
    "U79.3, U83.3",
    "U80.1, U86.2"
  )
)

# Identify chronic conditions with decimal format (default)
results1 <- plumage(hospital_data, "icd_codes")

# View category summaries
results1[, c("patient_id", "total_conditions",
             "total_cardiovascular_conditions",
             "total_respiratory_conditions")]

# Identify chronic conditions without decimal format
results2 <- plumage(hospital_data, "icd_codes", decimal = FALSE)

# Identify chronic conditions with prefix
results3 <- plumage(hospital_data, "icd_codes", prefix = "chronic_")

# Keep only summary columns, drop individual conditions
results4 <- plumage(hospital_data, "icd_codes", drop_eggs = TRUE)
```

---

preening *Prettification of infectious diseases datasets*

Description

Prettifies your dataset in preparation for data exploration and presenting tables. Adds variable labels and creates a series of age and time categories for analysis. Just list the dataframe and it let it clean

your variables and create exploratory variables. Use it as late in the workflow as possible, but, can be used at anytime.

Classic workflow would be:

1. `clean_the_nest` to clean and prep data for linkage. Pay close attention to your linkage variables (letternames, date of birth, medicare number, gender and/or postcode), and ensure all dates are formatted as dates.
2. `murmuration` to link cases to vaccination data (named here "c2v").
3. `murmuration` to link c2v to hospitalization data (named here c2v2h). Of note, you can skip linking the vaccination dataset.
4. `preening` to prettify the dataframe prepping it for exploration, analysis and presentation. Great to use with `gtsummary::tbl_summary()`.

Usage

```
preening(
  df,
  create_age_categories = TRUE,
  create_temporal_vars = TRUE,
  calculate_age = TRUE,
  age_reference_date = NULL
)
```

Arguments

`df`	The dataset as a dataframe, which can be a case notifications dataset (infections), hospital admissions or vaccination dataset.
`create_age_categories`	Logical. If TRUE (default), creates 21 standardized age category variables. Requires an 'age' variable in the dataset.
`create_temporal_vars`	Logical. If TRUE (default), creates temporal variables (ISO weeks, quarters, months) for date columns.
`calculate_age`	Logical. If TRUE (default), attempts to calculate age from dob if age variable is missing.
`age_reference_date`	Character. Column name to use as reference date for age calculation if age is missing. If NULL (default), uses first available from: event_date, onset_date, admission_date, first_vax_date, last_vax_date, vax_date_*.

Details

This function enhances infectious disease datasets by:

- Adding descriptive variable labels for cleaner tables and graphics
- Creating comprehensive temporal variables (ISO weeks, quarters, months) from date fields
- Generating 21 standardized age category variables for flexible analysis

- Calculating age from date of birth if not already present
- Adding useful derived variables for epidemiological analysis

IMPORTANT - Date Format Requirements:

All date columns MUST be in R's Date format before using this function. The function expects dates to already be properly formatted and will error with a clear message if they are not.

Common date conversions:

- From character: data$dob <- as.Date(data$dob, format = "%Y-%m-%d")
- From character (alternative): data$dob <- lubridate::ymd(data$dob)
- From Excel dates: data$dob <- as.Date(data$dob, origin = "1899-12-30")
- Always check: class(data$dob) should return "Date"

If you receive an error like "column must be in Date format", convert your date columns first, then run preening().

Age Categorization: If create_age_categories = TRUE and an 'age' variable exists (or can be calculated), the function creates 21 standardized age category variables with nomenclature age[x]cat where x indicates the number of categories:

age2cat 2 categories: Pediatric vs Adult (<18, 18+)

age3cat 3 categories: Child, Adult, Older Adult (<18, 18-64, 65+)

age4cat 4 categories: Infant/Child, Young Adult, Adult, Older Adult (<5, 5-17, 18-64, 65+)

age5cat 5 categories: Standard public health categories (0-4, 5-17, 18-64, 65-74, 75+)

age6cat 6 categories: Granular infant categories (<1, 1-4, 5-17, 18-64, 65-74, 75+)

age7cat 7 categories: Fine pediatric cuts (<1, 1, 2-4, 5-11, 12-17, 18-64, 65+)

age8cat 8 categories: Infant subcategories (<3mo, 3-5mo, 6-11mo, 1-4, 5-17, 18-64, 65-74, 75+)

age9cat 9 categories: Monthly infant categories (<1mo, 1mo, 2-5mo, 6-11mo, 1-4, 5-17, 18-64, 65-74, 75+)

age10cat 10 categories: Decade bands (0-4, 5-9, 10-19, 20-29, 30-39, 40-49, 50-59, 60-69, 70-79, 80+)

age11cat 11 categories: Fine older adult categories (0-4, 5-17, 18-29, 30-39, 40-49, 50-59, 60-69, 70-79, 80-89, 90-99, 100+)

age12cat 12 categories: Detailed pediatric + adult decades (<1, 1-4, 5-9, 10-14, 15-19, 20-29, 30-39, 40-49, 50-59, 60-69, 70-79, 80+)

age13cat 13 categories: Very fine infant + standard adult (<1mo, 1mo, 2mo, 3-5mo, 6-11mo, 1, 2-4, 5-11, 12-17, 18-39, 40-64, 65-79, 80+)

age14cat 14 categories: ABS-like with fine elderly (0-4, 5-9, 10-14, 15-19, 20-24, 25-29, 30-39, 40-49, 50-59, 60-69, 70-79, 80-84, 85-89, 90+)

age15cat 15 categories: Vaccine schedule aligned (<2mo, 2-3mo, 4-5mo, 6-11mo, 1, 2-3, 4, 5-11, 12-17, 18-49, 50-64, 65-74, 75-84, 85-94, 95+)

age16cat 16 categories: Granular pediatric + 10-year adult bands (<1, 1, 2, 3, 4, 5-9, 10-14, 15-19, 20-29, 30-39, 40-49, 50-59, 60-69, 70-79, 80-89, 90+)

age17cat 17 categories: WHO/UNICEF standard with extensions (<1mo, 1-5mo, 6-11mo, 1, 2-4, 5-9, 10-14, 15-19, 20-24, 25-29, 30-39, 40-49, 50-59, 60-69, 70-79, 80-89, 90+)

age18cat 18 categories: Standard 5-year bands (census/ABS style) (0-4, 5-9, 10-14, 15-19, 20-24, 25-29, 30-34, 35-39, 40-44, 45-49, 50-54, 55-59, 60-64, 65-69, 70-74, 75-79, 80-84, 85+)

age19cat 19 categories: Extended 5-year bands with fine elderly (0-4, 5-9, ..., 80-84, 85-89, 90+)

age20cat 20 categories: Monthly up to 12 months + standard thereafter (<1mo, 1mo, 2mo, 3mo, 4mo, 5mo, 6mo, 7mo, 8mo, 9mo, 10mo, 11mo, 1-4, 5-17, 18-39, 40-64, 65-74, 75-84, 85-94, 95+)

age21cat 21 categories: Comprehensive life course categories (<1mo, 1-2mo, 3-5mo, 6-11mo, 1, 2-4, 5-9, 10-14, 15-19, 20-24, 25-29, 30-34, 35-39, 40-44, 45-49, 50-54, 55-59, 60-64, 65-74, 75-84, 85+)

Value

The output is a dataframe with variable labels (useful for making pretty tables and graphics), and creates several age categories and time categories (month-year, quarter-year etc.)

---

tweet *Examine and summarize variables in a dataset*

Description

Provides a comprehensive summary of variables in a dataset after cleaning with clean_the_nest. This function examines variables by type, providing appropriate statistics for numeric, date, factor, and character variables. For numeric variables, it shows min/max values, quartiles and missing data counts. For date variables, it displays the date range and percentage of non-missing values. For factor and character variables, it shows the number of unique levels, frequency of top levels, and missing data counts.

Usage

```
tweet(data, select_vars = NULL, top_n = 3, sort_by = "type")
```

Arguments

`data`	The dataset, typically output from clean_the_nest function
`select_vars`	Optional vector of variable names to examine. If NULL, all variables will be summarized.
`top_n`	Number of top categories to display for factor and character variables. Default is 3.
`sort_by`	How to sort variables in the output. Options are "name" (alphabetical) or "type" (grouped by data type). Default is "type".

Value

A data frame with one row per variable, containing variable name, type, missingness, and type-specific statistics.

Examples

```
# basic usage of tweet after clean_the_nest
data(dx_data)
df_diag <- clean_the_nest(dx_data, drop_eggs=TRUE, data_type = "cases",
  id_var ="identity",
  diagnosis = "disease_name",
  lettername1 = "first_name",
  lettername2 = "surname",
  dob = "date_of_birth",
  medicare = "medicare_no",
  gender = "gender",
  postcode="postcode",
  fn="indigenous_status",
  onset_date = "diagnosis_date")

# Examine all variables in the cleaned dataset
summary_df <- tweet(df_diag)

# Examine only specific variables
summary_df_subset <- tweet(df_diag, select_vars = c("age", "gender", "onset_date"))

# Show more categories for factor variables
summary_df_detailed <- tweet(df_diag, top_n = 5)
```

---

vax_data *Example Vaccination Dataset*

Description

A sample dataset containing vaccination records in long format for multiple individuals, with some having multiple vaccination entries.

Usage

```
vax_data
```

Format

A data frame with 15 rows and 9 variables:

patient_id Patient identifier
firstname Patient's first name
last_name Patient's last name
birth_date Patient's date of birth
medicare_number Medicare number
gender Gender (Male/Female)
postcode Postcode of residence
vaccine_delivered Type of vaccine administered
service_date Date of vaccination

Examples

```
data(vax_data)
head(vax_data)
```

Index