Package 'starling'

July 10, 2026

Title Record Linkage for Public Health Surveillance

Version 1.1.0

Description Record linkage for public health surveillance datasets using either the Fellegi-Sunter probabilistic framework (via reclin2) or deterministic exact-key matching. Provides pre-linkage data quality auditing (preflight()), Medicare number checksum validation (check_medicare()), blocking variable construction (flock()), and the murmuration() linkage engine. murmuration() expresses linkage as a single cohort-to-event concept (linking a linelist of people to a dated stream of vaccination, hospitalisation, or case events within a before/during/after time window), with the historical four-code taxonomy (case-to-hospitalisation, vaccination-to-case, vaccination-to-hospitalisation, vaccination-to-event) retained as deprecated aliases. Also provides linkage weight distribution visualisation (murmuration_plot()) and threshold sensitivity analysis (perch()) with annotated AIHW, WA Data Linkage Unit, and PHRN reference benchmarks. perch() can be called standalone or triggered automatically mid-linkage via murmuration(perch_before_linking = TRUE). Includes synthetic datasets (cases_notifiable, vax_air) and three vignettes demonstrating the complete linkage workflow. Part of the aviary public health surveillance ecosystem.

License MIT + file LICENSE

Encoding UTF-8

Depends R (>= 4.1)

Imports reclin2, dplyr, stringr, lubridate, rlang, ggplot2

Suggests patchwork, plotly, phonics, mudnester, testthat (>= 3.0.0), knitr, rmarkdown

VignetteBuilder knitr

URL https://github.com/nrsmoll/starling

BugReports https://github.com/nrsmoll/starling/issues

Config/roxygen2/version 8.0.0

NeedsCompilation no

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

Maintainer Nicolas Smoll <nicolas.smoll@health.qld.gov.au>

Repository CRAN

Date/Publication 2026-07-10 07:30:02 UTC

Contents

`cases_notifiable` *Synthetic Notifiable Disease Case Linelist*

Description

Bird note: Like the individual birds that form a murmuration, each row in this dataset is one record that must find its match in the vaccination register. The dataset is designed so that some birds have a clear pair and some do not — mirroring real-world linkage where not every notified case appears in the AIR.

A synthetic EDIS/case notification linelist for use in starling vignettes, documentation examples, and tests. No real person data. All names, dates of birth, Medicare numbers, and postcodes are randomly generated. 300 records; approximately 200 have a true match in `vax_air`.

The dataset deliberately includes data quality challenges that mirror real-world linkage scenarios: approximately 5 name typo (adjacent characters swapped), and approximately 10 numbers have one digit corrupted (checksum fails). These are designed to demonstrate `check_medicare`, `preflight`, `flock`, `murmuration`, and `perch`.

Usage

```
data(cases_notifiable)
```

Format

A data frame with 300 rows and 10 columns:

id_var character. Unique case identifier (`"EDIS_00001"`, etc.).

true_link_id character. The matching `id_var` from `vax_air` for records with a true match; `NA` for the ~100 unmatched cases. Used in vignettes for post-hoc recall and precision calculation. Never passed to `murmuration()`.

lettername1 character. First name (standardised form expected by `murmuration()`).

lettername2 character. Surname.

dob Date. Date of birth.

gender character. `"Male"` or `"Female"`.

postcode character. Sunshine Coast postcode (4550–4562).

medicare10 character. 10-digit Medicare number. ~10% have one digit deliberately corrupted so `check_medicare` flags them as invalid (flag = `0L`).

onset_date Date. Date of symptom onset (2024 calendar year).

pathogen character. Notified pathogen: one of Influenza A, Influenza B, COVID-19, RSV, Pertussis, Salmonellosis, Campylobacteriosis.

Source

Generated by `data-raw/starling_synthetic_data.R`. Run `source("data-raw/starling_synthetic_data.R")` from the package root to rebuild.

See Also

`vax_air` for the paired vaccination register dataset. `vignette("linked-cohort", package = "starling")` for a worked end-to-end linkage example using both datasets.

Examples

```
## Not run:
data(cases_notifiable)
head(cases_notifiable)

# Check Medicare validity before linkage
cases_checked <- check_medicare(cases_notifiable)
table(cases_checked$medicare_valid, useNA = "always")

# Pre-linkage quality audit
data(vax_air)
preflight(cases_notifiable, vax_air,
          linkage_vars = c("lettername1", "lettername2", "dob", "medicare10"),
          medicare_col = "medicare10")

## End(Not run)
```

`check_medicare` *Validate Australian Medicare Numbers Using the Modulus 10 Checksum*

Description

Bird note: Starlings are famously alert to impostors – a murmuration will immediately eject a bird that does not move quite right. `check_medicare()` plays the same sentinel role before linkage: it identifies impostors hiding in your Medicare number field before they corrupt the match scores. A single transposed digit in a Medicare number silently destroys a linkage pair; catching it here costs nothing compared to auditing a linked dataset afterwards.

Validates Australian Medicare numbers using the official Services Australia check-digit algorithm: a Modulus 10 weighted checksum on digits 1-8, verified against the 9th digit. The 10-digit Medicare card number has the structure XXXXXXXXX C I, where digits 1-8 form the individual identifier, digit 9 (C) is the check digit derived from those 8 digits, and digit 10 (I) is the Individual Reference Number (IRN) identifying family members on the same card. Only the check digit (position 9) is validated here; the IRN is not part of the checksum algorithm.

Usage

```
check_medicare(
  data,
  medicare_col = "medicare10",
  output_col = "medicare_valid",
  verbose = TRUE,
  keep_digits = 10L
)
```

Arguments

`data`	A data frame containing a Medicare number column.
`medicare_col`	Character. Name of the column containing Medicare numbers. Values may include spaces or hyphens (stripped before validation). Default `"medicare10"` (the standardised name from `mudnester::clean_the_nest()`).
`output_col`	Character. Name of the new validation flag column added to `data`. Default `"medicare_valid"`. Values: `1L` for a valid checksum, `0L` for an invalid checksum, `NA` for a missing or non-numeric entry.
`verbose`	Logical. If `TRUE` (default), prints a summary report to the console showing total records, valid count and percentage, invalid count, missing count, and a warning if the valid rate falls below 95%.
`keep_digits`	Integer. Either `9` or `10` (default). Length of a valid Medicare number string after stripping spaces and hyphens. Use `10` for the full card number including IRN; use `9` for numbers stored without the IRN.

Details

The Modulus 10 check-digit algorithm

The 9th digit of an Australian Medicare number is computed from the first 8 digits using positional weights 1, 3, 7, 9, 1, 3, 7, 9 (repeating 1-3-7-9). The check digit C is:

C = (d1*1 + d2*3 + d3*7 + d4*9 + d5*1 + d6*3 + d7*7 + d8*9) mod 10

If the computed value equals position 9 of the submitted number, the number passes validation.

Interpreting the output flag

A flag of NA means the field was missing, blank, or non-numeric – these records cannot be linked on Medicare number but may still link via other variables. A flag of 0L means the number is present and numeric but the checksum fails – at least one digit is wrong and the number should not be used as a linkage variable. A flag of 1L means the checksum passes – the number is internally consistent, though it may still be wrong (e.g. a different person's valid card number).

A valid-checksum rate below 95% usually signals a systematic data entry or export issue and should be investigated before linkage.

Value

The input data frame with two new columns appended: medicare_clean (the Medicare number with spaces, hyphens, and dots stripped; NA for missing or empty entries) and the column named by output_col (integer flag: 1L valid, 0L invalid, NA missing or non-numeric).

References

Services Australia (2024). Medicare card number format and check digit. https://www.servicesaustralia.gov.au/

See Also

preflight for a full pre-linkage audit. flock for blocking variable construction. murmuration for the linkage step. murmuration_plot for threshold visualisation.

Examples

```
## Not run:
# Basic usage
cases_checked <- check_medicare(cases_clean)

# Custom column name
cases_checked <- check_medicare(cases_clean,
  medicare_col = "medicare_number",
  output_col   = "mcn_valid")

# Suppress console report
cases_checked <- check_medicare(cases_clean, verbose = FALSE)

# Exclude invalid Medicare numbers before linkage
cases_clean$medicare10 <- ifelse(
  cases_checked$medicare_valid == 1L,
  cases_clean$medicare10,
  NA_character_
)

# Validate 9-digit numbers (no IRN appended)
cases_checked <- check_medicare(cases_clean,
  medicare_col = "medicare9",
  keep_digits  = 9L)
```

```
## End(Not run)
```

`flock` *Create Blocking Variables for Probabilistic Record Linkage*

Description

Bird note: Before a murmuration forms, starlings first gather into loose flocks — smaller, manageable sub-groups that share a roost site or feeding ground. `flock()` plays that preparatory role: it partitions records into candidate sub-groups (blocks) so that `murmuration()` only compares records within the same block, rather than every record against every other. Good blocking dramatically reduces computation without sacrificing linkage quality, provided the blocking variable is near-complete and consistent across both datasets.

Generates one or more blocking variables from demographic fields already standardised by `clean_the_nest()`. Blocking variables partition the record space so that `murmuration()` only compares candidate pairs within the same block. Three strategies are offered:

- **Single field** — use one column directly (e.g. `"gender"`, `"postcode"`).
- **Phonetic** — Soundex or Double Metaphone encoding of a name field, so that near-spelling-variants of the same name block together.
- **Composite** — concatenation of two or more fields into one blocking key (e.g. first letter of `lettername2` + birth year).

All three strategies can be generated in a single `flock()` call and appended as new columns to the data frame, ready to pass to `murmuration()`'s `blocking_var` argument.

Usage

```
flock(
  data,
  block1_vars = NULL,
  block2_vars = NULL,
  block3_vars = NULL,
  phonetic_vars = NULL,
  phonetic_method = "soundex",
  birth_year_col = NULL,
  postcode_col = NULL
)
```

Arguments

`data`	A data frame that has been processed by `clean_the_nest()`. Must contain the columns referenced by `block1_vars` / `block2_vars` / `block3_vars`.
`block1_vars`	Character vector of one or more column names to combine into the primary blocking variable (`block1`). Single element = direct use of that column. Multiple elements = concatenated composite key.

`block2_vars` Optional character vector for a second blocking variable (`block2`). Use a second, independent variable to allow `murmuration()` to be run twice with different blocks and results merged — the standard multi-pass blocking strategy to improve recall. `NULL` (default) suppresses `block2`.

`block3_vars` Optional character vector for a third blocking variable (`block3`). `NULL` (default) suppresses `block3`.

`phonetic_vars` Optional character vector of name columns to encode phonetically. Each column `x` generates a new column `x_soundex`. These can then be referenced in `block1_vars` / `block2_vars` / `block3_vars`. Requires the **phonics** package.

`phonetic_method`
One of `"soundex"` (default) or `"metaphone"`. Applied to all columns in `phonetic_vars`.

`birth_year_col` Optional name of a `Date` column from which birth year is extracted. If supplied, a `birth_year` integer column is added, which can be referenced in composite blocking keys.

`postcode_col` Optional name of a postcode column. If supplied, the first 3 digits are extracted as `postcode3` (broad geographic blocking).

Details

Blocking strategy guidance

The goal of blocking is to be broad enough to keep true matches in the same block, and narrow enough to avoid exploding the number of candidate pairs. Practical guidance for Australian surveillance linkage:

Block	Recommended variable	Notes
Primary	`"gender"`	Near-universal; halves the candidate space immediately
Secondary	`"postcode"` or `"postcode3"`	Good geographic signal; use `postcode3` if postcode completeness is low
Tertiary	Soundex of `lettername2`	Catches surname spelling variants; do not use as the sole block

Multi-pass blocking (running `murmuration()` separately with `block1` and `block2` then unioning results) substantially improves recall at modest computational cost. This is the recommended approach for large datasets (> 100 000 records in either dataset).

Why not block on Medicare number?

Medicare number is an excellent *comparison* variable but a poor blocking variable: missing or transcription-error values will split a true pair across blocks, causing them to never be compared. Use Medicare as a `compare_var` in `murmuration()`, and as a *component* of a composite block only when completeness is verified (e.g. > 95% non-missing in both datasets).

Value

The input data frame with new columns appended:

`block1` Primary blocking key (always present).

`block2` Secondary blocking key (if `block2_vars` supplied).

`block3` Tertiary blocking key (if `block3_vars` supplied).

<col>_soundex **or** <col>_metaphone Phonetic encoding columns (if phonetic_vars supplied).

birth_year Integer birth year (if birth_year_col supplied).

postcode3 First 3 digits of postcode (if postcode_col supplied).

Original columns are always preserved.

See Also

murmuration, mudnester::clean_the_nest()

Examples

```
## Not run:
# Single-field primary block (gender) + postcode secondary block
cases_blocked <- flock(cases_clean,
  block1_vars = "gender",
  block2_vars = "postcode")

# Composite primary block: first letter of surname + gender
cases_blocked <- flock(cases_clean,
  block1_vars  = c("lettername2_initial", "gender"),
  block2_vars  = "gender",
  phonetic_vars = "lettername2")

# Full three-pass setup
cases_blocked <- flock(cases_clean,
  block1_vars   = "gender",
  block2_vars   = "postcode3",
  block3_vars   = "birth_year",
  phonetic_vars = "lettername2",
  birth_year_col = "dob",
  postcode_col  = "postcode")

# Then pass to murmuration() separately for each block:
linked1 <- murmuration(cases_blocked, vax_blocked,
  blocking_var = "block1", ...)
linked2 <- murmuration(cases_blocked, vax_blocked,
  blocking_var = "block2", ...)
linked_all <- dplyr::bind_rows(linked1, linked2) |>
  dplyr::distinct(id_var.x, .keep_all = TRUE)

## End(Not run)
```

---

murmuration *Links case, hospital, outbreak, event linelists, vaccination datasets*

Description

Bird note: A murmuration is one of nature's most arresting phenomena: tens of thousands of starlings moving as a single fluid shape across the sky, with no conductor and no central instruction — each bird responding only to its nearest neighbours until a coherent, shifting pattern emerges from the whole flock. `murmuration()` works the same way: it takes two separate, uncoordinated datasets and lets pairwise local comparisons between records resolve, through the Fellegi-Sunter EM algorithm, into one coherent linked result. No single record "knows" about the others — the shape emerges from the scoring.

Probabilistic record linkage of two surveillance datasets using the Fellegi-Sunter framework. Links diagnostic case linelists, hospital admission records, outbreak linelists, and event manifests to vaccination history (Australian Immunisation Register) or to each other. Wraps the full **reclin2** pipeline — blocking, comparison, EM scoring, threshold selection, and post-linkage window filtering — in a single practitioner- facing call.

Usage

```
murmuration(
  df1,
  df2,
  linkage_type = NULL,
  event_type = NULL,
  method = "probabilistic",
  event_date = NULL,
  id_var,
  blocking_var,
  compare_vars,
  threshold_value = 17,
  perch_before_linking = FALSE,
  vax_window = list(days_before = 14, days_after = 0, lookback_days = Inf),
  cohort_window = NULL,
  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_the_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 event dates (e.g., onset_date).
`df2`	This is a dataframe object, cleaned using clean_the_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` The linkage to perform. As of starling 1.1.0 the preferred value is `"cohort2event"` — a single, explicit linkage concept: link a base linelist of people (a "cohort" — cases, hospitalisations, a birth cohort, an outbreak linelist, a flight manifest) to a stream of dated events (a "case" diagnosis, a "hospitalisation", or a "vaccination"), keeping events that fall in a time window relative to an anchor. Pair it with `event_type` to name the event stream. The window may sit *before*, *during*, or *after* the cohort's follow-up, governed by `vax_window` (event-relative) or `cohort_window` (calendar).

Legacy codes (deprecated, still functional): the historical `"c2h"`/`"v2c"`/`"v2h"`/`"v2e"` tokens still work and route identically, but emit a deprecation warning nudging to the collapsed form: `"c2h"` (cases->hospitalisations) becomes `event_type = "hospitalisation"`; `"v2c"`/`"v2h"` (->vaccination history) become `event_type = "vaccination"`; `"v2e"` (fixed-date event) becomes `event_type = "vaccination"` with a scalar `Date` passed to `event_date`. If `NULL` (default), `"c2h"` is assumed for backwards compatibility. When linking to a vaccination dataset, use a single row per person (run the vaccination data through `clean_the_nest()` with `lie_nest_flat = TRUE`).

`event_type` Required when `linkage_type = "cohort2event"`: names the event stream in df2. One of `"vaccination"`, `"hospitalisation"` (`"hospitalization"` also accepted), or `"case"`. Ignored (with the routing inferred from the code) when a legacy `linkage_type` is supplied.

`method` Linkage engine. `"probabilistic"` (default) uses the Fellegi-Sunter EM pipeline (**reclin2**) with fuzzy field comparison and a score threshold — appropriate when identifiers are imperfect and you need weighted, partial-agreement matching. `"deterministic"` skips EM scoring entirely and links only records whose `compare_vars` match *exactly* (all fields, no partial credit). Deterministic linkage is the right choice when you have a clean, reliable composite key (e.g. full name + date of birth, as in Smoll et al. 2023) and want reproducible, explainable, threshold-free links. Under `"deterministic"`: `weights` is `NA` (no composite score exists), `threshold_value`/`perch_before_linking` are ignored, and a non-unique key in df2 emits a message rather than silently resolving. All linkage types and both window mechanisms work under either method.

`event_date` The anchor date used to define the linkage window and determine valid vaccinations or related admissions.

- For `"c2h"`, `"v2c"`, and `"v2h"`: a character string naming any date column present in df1. Any date column produced by `mudnester::clean_the_nest()` is valid here — not just `"onset_date"` or `"admission_date"`. In birth-cohort or coverage studies, pass `"cohort_entry_date"` to anchor the vaccination window from the start of follow-up, or `"cohort_exit_date"` to bound from the end. Examples: `"onset_date"`, `"admission_date"`, `"cohort_entry_date"`, `"cohort_exit_date"`.
- For `"v2e"`: a `Date` object (e.g., `as.Date("2024-12-15")`) representing a fixed event date shared across all records (e.g. a flight, an outbreak event).

This parameter is critical for determining valid vaccinations (must occur before `event_date - days_allowed_before_event`) and for `"c2h"` linkages, determining disease-related admissions.

`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. "block2"). Choice of blocking variable. Use `flock()` to automatically construct up to three blocking variables (coarse, medium, fine) from standardised linkage fields — it produces columns named `block1`, `block2`, `block3` by default. You can also create your own. Choose a broader/looser variable (e.g. gender) for small datasets or low data quality, and a finer variable (e.g. postcode + DOB year) for large, high-quality datasets.

`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. 17), default is 17. This is the Fellegi-Sunter log-likelihood ratio score above which a pair is classified as a match. The score is **dataset-specific** — there is no universal correct value. Use `murmuration_plot()` to inspect your score distribution and select a threshold that sits in the natural valley between the match and non-match peaks. Higher values increase specificity (fewer false links); lower values increase sensitivity (fewer missed true matches). Typical working ranges against SCPHU datasets with 4-5 comparison variables: conservative ~20-25; balanced ~15-20; sensitive ~10-14.

`perch_before_linking`

Logical. If `TRUE`, calls `perch` on the scored pairs immediately after the EM algorithm fits and before the threshold is applied, printing a sensitivity table showing match counts and clerical burden across the score range. In interactive sessions the plot is also displayed. In non-interactive sessions (e.g. Quarto render, batch jobs) only the table is printed and execution continues automatically. Default `FALSE`.

`vax_window` A named list defining the vaccination validity window relative to `event_date`. Used for all vaccination linkage types (`"v2c"`, `"v2h"`, `"v2e"`). Contains up to three elements:

`days_before` Minimum days *before* `event_date` a vaccine must have been administered to count as prior valid exposure. Vaccines administered within this buffer (i.e. fewer than `days_before` days before the event) are excluded because they may not yet have conferred protection. Default `14`. Set to `0` if you want all prior vaccinations regardless of recency (e.g. birth-cohort studies where protection is assumed immediate).

`days_after` Maximum days *after* `event_date` to retain a vaccination record. Default `0` (no post-event vaccinations retained). Set to a positive value to capture post-event vaccination – useful for studying vaccination uptake following a diagnosis or intervention (e.g. 365 to capture all vaccinations in the year after diagnosis). Post-event vaccinations are flagged with a new `vax_timing` column (`"pre_event"` or `"post_event"`) so they can be analysed separately downstream.

`lookback_days` Maximum days *before* `event_date` to look back for valid vaccinations. Default `Inf` (no upper lookback limit – any prior vaccination regardless of age counts). Set to `365` to restrict to vaccinations in the past year

only (appropriate for annual influenza VE studies where a vaccine from five years ago is not the relevant exposure).

Example: `vax_window = list(days_before = 14, days_after = 0, lookback_days = 365)` retains only vaccinations that occurred 14 or more days before the event and within the past year, excluding both too-recent and too-old doses.

Overridden by `cohort_window` **when both are supplied.** For `"c2h"` linkages, use `days_allowed_before_event` and `days_allowed_after_event` instead (those govern the admission window, not vaccination).

`cohort_window` An optional named list defining a *calendar-based* observation window that **replaces** `vax_window` when supplied. Use this for cohort studies, coverage analyses, or any design where the observation window is defined at the study level rather than relative to individual event dates:

`entry` A `Date` object (or character coercible to Date via `as.Date()`) giving the start of the observation window. Vaccinations before this date are excluded. Example: `as.Date("2024-01-01")`.

`exit` A `Date` object giving the end of the observation window. Vaccinations after this date are excluded. Example: `as.Date("2024-12-31")`.

When `cohort_window` is supplied, `vax_window` is ignored entirely and a message is emitted confirming which window is active. Useful when the event date is unknown, unreliable, or irrelevant to the vaccination exposure definition (e.g. a prevalence survey, a screening programme evaluation, or a birth-cohort study anchored to a fixed follow-up period rather than individual birth dates). Can be used with any `linkage_type`.

`days_allowed_before_event`
Numeric. For `"c2h"` linkages only: the lower bound of the disease-related admission window – how many days before `event_date` an admission may still be considered disease-related. Default `7`. Not used for vaccination linkage types; use `vax_window$days_before` for those.

`days_allowed_after_event`
Numeric. For `"c2h"` linkages only: the upper bound of the disease-related admission window – how many days after `event_date` an admission may still be considered disease-related. Default `14`. Not used for vaccination linkage types; use `vax_window$days_after` for those.

`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_the_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 event_date 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

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()`. For example, if both datasets have a date of death, choose the dataset with the highest confidence and drop the date of death from the other before linking. If linking to a hospitalisation dataset, the difference between `event_date` and `admission_date` is used to identify disease-related admissions; unrelated hospitalisations can be filtered separately using ICD-10 codes or AR-DRG codes prior to linkage.

The recommended pre-linkage workflow is:

1. `mudnester::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. `linkage_quality()` to run a structured battery of pre-linkage quality checks across both datasets (completeness, duplicates, Medicare validity, date plausibility, gender and DOB year distribution).
3. `check_medicare` to validate Medicare check digits and flag invalid numbers before they enter comparison scoring.
4. `flock()` to create up to three blocking variables at different specificity levels (coarse, medium, fine) — pass one to the `blocking_var` argument below.
5. `murmuration` with linkage_type="v2c" to link cases to vaccination data.
6. `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.
7. `murmuration` with linkage_type="v2e" to link event linelists (flight manifests, outbreak investigations) to vaccination history.
8. `murmuration_plot()` to visualise the full linkage score distribution and confirm the threshold before accepting the linked dataset.
9. `mudnester::preening()` to prettify the dataframe prepping it for exploration, analysis and presentation. Great to use with `gtsummary::tbl_summary()`.

On threshold selection The `threshold_value` is a Fellegi-Sunter log-likelihood ratio score and is **dataset-specific** — there is no universal correct value. The default of 17 was chosen empirically against SCPHU datasets with 4-5 comparison variables (names, DOB, Medicare, gender). Use `flock_plot()` to inspect the bimodal score distribution from your specific dataset and identify the natural valley between the match and non-match peaks. Typical working ranges: conservative (high specificity) ~20-25; balanced (default) ~15-20; sensitive (high recall) ~10-14.

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

```
## Not run:
# Example 1: Link cases to vaccination history (onset_date as anchor)
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",
  onset_date = "date_of_onset")

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")

df1 <- murmuration(dx_clean, vax_clean,
  linkage_type = "v2c",
  event_date = "onset_date",
  id_var = "identity",
  blocking_var = "gender",
  compare_vars = c("lettername1", "lettername2", "dob"),
  clean_eggs = FALSE)

# Example 2: Link hospitalization data to vaccination history (admission_date as anchor)
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 = "v2h",
  event_date = "admission_date",
  id_var = "patient_id",
  blocking_var = "gender",
  compare_vars = c("lettername1", "lettername2", "medicare10", "dob"),
  clean_eggs = FALSE,
  one_row_per_person = TRUE)

# Example 3: Link cases to hospitalisations (onset_date as anchor)
df3 <- murmuration(dx_clean, hosp_clean,
  linkage_type = "c2h",
  event_date = "onset_date",
  id_var = "identity",
  blocking_var = "postcode",
  compare_vars = c("lettername1", "lettername2", "dob", "medicare10"),
  days_allowed_before_event = 7,
  days_allowed_after_event = 30,
  clean_eggs = FALSE)

# Example 4: Birth-cohort study (e.g. RICOR nirsevimab) — cohort_window approach.
# Captures any vaccination administered during the defined follow-up period,
# regardless of individual event dates.
cohort_clean <- clean_the_nest(birth_cohort_data,
  data_type = "cases",
  id_var = "baby_id",
  lettername1 = "first_name",
  lettername2 = "last_name",
  dob = "babys_date_of_birth",
  gender = "sex")

df_cohort <- murmuration(cohort_clean, vax_clean,
  linkage_type  = "v2c",
  event_date    = "dob",
  id_var        = "baby_id",
  blocking_var  = "gender",
  compare_vars  = c("lettername1", "lettername2", "dob"),
  cohort_window = list(
    entry = as.Date("2023-01-01"),
    exit  = as.Date("2024-06-30")
  ),
  clean_eggs = FALSE)

# Example 4b: Annual influenza VE study — 1-year lookback, 14-day buffer.
# Only vaccinations in the 12 months before onset, and >= 14 days before onset,
# are retained as valid prior exposure.
```

```
df_flu_ve <- murmuration(cases_clean, vax_clean,
  linkage_type = "v2c",
  event_date   = "onset_date",
  id_var       = "identity",
  blocking_var = "gender",
  compare_vars = c("lettername1", "lettername2", "dob", "medicare10"),
  vax_window   = list(
    days_before   = 14,
    days_after    = 0,
    lookback_days = 365
  ),
  clean_eggs = FALSE)

# Example 4c: Post-diagnosis vaccination uptake study.
# Retains vaccinations up to 1 year AFTER diagnosis as well as valid prior doses.
# The vax_timing column flags each dose as "pre_event" or "post_event".
df_post_dx <- murmuration(cases_clean, vax_clean,
  linkage_type = "v2c",
  event_date   = "onset_date",
  id_var       = "identity",
  blocking_var = "gender",
  compare_vars = c("lettername1", "lettername2", "dob", "medicare10"),
  vax_window   = list(
    days_before   = 14,
    days_after    = 365,
    lookback_days = Inf
  ),
  clean_eggs = FALSE)

# Example 5: Link flight manifest to vaccination history (fixed event Date object)
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"),
  id_var = "passenger_id",
  blocking_var = "gender",
  compare_vars = c("lettername1", "lettername2", "dob"),
  days_allowed_before_event = 14,
  clean_eggs = FALSE)

# Example 6: Link outbreak linelist to vaccination history (fixed event Date object)
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"),
  id_var = "case_id",
  blocking_var = "postcode",
  compare_vars = c("lettername1", "lettername2", "dob", "medicare10"),
  days_allowed_before_event = 7,
  clean_eggs = FALSE)

## End(Not run)
```

`murmuration_plot` *Visualise Linkage Weight Distribution with Threshold Overlay*

Description

Bird note: When a murmuration peaks, a watcher on the ground sees the flock split momentarily into two layers — the dense, high-altitude core of confirmed companions and the looser, lower-flying fringe of uncertain travellers — before the whole shape resolves again. `murmuration_plot()` captures exactly that split: the linkage weight distribution separates into a high-score cluster of likely true matches at the top, a low-score cluster of likely non-matches at the bottom, and — critically — a gap or overlap zone in between that tells you whether your threshold is well-placed or needs adjustment.

Generates a jittered scatter plot of Fellegi-Sunter linkage weights from a scored pairs object (the output of `predict()` in the **reclin2** workflow, before `select_threshold()` is called). Weights are plotted on the y-axis with jitter on the x-axis for readability. A horizontal line marks the chosen threshold, with regions above and below shaded to distinguish likely matches from likely non-matches. Optionally overlays a density curve to highlight the bimodal (match/non-match) distribution.

Use this plot before finalising `threshold_value` in `murmuration` to confirm that the threshold sits in the natural valley between the two score clusters. If the valley is not visible (scores form a unimodal distribution), this is a signal that the linkage variables or blocking strategy needs revision.

Usage

```
murmuration_plot(
  pairs_pred,
  threshold = 17,
  weight_col = "weights",
  n_sample = 5000L,
  jitter_width = 0.3,
  point_alpha = 0.35,
```

```
  point_size = 1.2,
  show_density = TRUE,
  show_counts = TRUE,
  palette = "sch",
  title = NULL,
  interactive = FALSE
)
```

Arguments

pairs_pred	A pairs object with a `weights` column — the output of `reclin2::predict.problink_em(m, pairs = pairs, add = TRUE)`. May also be a plain data frame with a numeric `weights` column.
threshold	Numeric. The threshold value to display as a horizontal reference line. Default `17` (reasonable starting point for Australian surveillance linkage with Medicare + 2 names + DOB; see Details for guidance).
weight_col	Name of the column containing linkage weights. Default `"weights"`.
n_sample	Integer. Maximum number of points to plot (random sample drawn if the pairs object is larger than this). Plotting millions of points is slow and uninformative; the distribution shape is well-represented by 5 000–10 000 points. Default `5000`.
jitter_width	Numeric. Horizontal jitter width. Default `0.3`.
point_alpha	Numeric (0–1). Point transparency. Default `0.35`.
point_size	Numeric. Point size. Default `1.2`.
show_density	Logical. Overlay a right-margin density curve of the weight distribution? Default `TRUE`.
show_counts	Logical. Annotate the plot with the count of pairs above and below the threshold? Default `TRUE`.
palette	Character. Colour palette for match/non-match zones: `"sch"` (Sunshine Coast HHS greens — default), `"default"` (teal/coral), or `"grey"` (greyscale, publication-safe).
title	Optional plot title string. `NULL` uses a default title.
interactive	Logical. Return a **plotly** object (`TRUE`) or a **ggplot2** object (`FALSE`, default).

Details

Threshold guidance for Australian public health linkage

The Fellegi-Sunter weights produced by the EM algorithm are log-likelihood ratios, not fixed absolute scores, so the "right" threshold is dataset-specific. That said, common reference points for Australian population health work (Medicare, 2 names, DOB) are:

| Threshold | Practical meaning | |—|—| | 8–12 | High sensitivity, lower specificity — suitable when recall matters more than precision (e.g. small datasets, rare disease surveillance) | | 12–17 | Balanced — typical starting point for surveillance linkage with a complete variable set | | 17–22 | High specificity — suitable when false matches are especially costly (e.g. VE studies where false vaccination attribution biases the estimate) | | > 22 | Very conservative — consider adding clerical review for records in the 17–22 range |

The most principled approach is to look at the weight distribution (this plot) and choose the threshold at the **lowest density point** between the two modes. If the two modes overlap substantially, your variable set may lack discriminating power for this dataset — see `preflight` for diagnostic guidance.

Value

A **ggplot2** object (`interactive = FALSE`) or a **plotly** htmlwidget (`interactive = TRUE`).

See Also

`murmuration`, `preflight`

Examples

```
## Not run:
# Standard reclin2 workflow, with plot inserted before threshold selection
pairs <- reclin2::pair_blocking(df1, df2, blocking_var)
reclin2::compare_pairs(pairs, on = compare_vars,
  default_comparator = reclin2::jaro_winkler(0.9), inplace = TRUE)
m          <- reclin2::problink_em(reformulate(compare_vars), data = pairs)
pairs_pred <- predict(m, pairs = pairs, add = TRUE)

# Inspect the weight distribution before committing to a threshold
murmuration_plot(pairs_pred, threshold = 17)

# Try a different threshold, interactively
murmuration_plot(pairs_pred, threshold = 14, interactive = TRUE)

# Greyscale for a journal figure
murmuration_plot(pairs_pred, threshold = 17, palette = "grey")

## End(Not run)
```

---

perch *Threshold Sensitivity Analysis for Linkage Score Cutoff Selection*

Description

Bird note: A starling tests a perch before committing its weight to it — sitting briefly, sensing stability, probing whether this particular spot will hold before deciding to stay or move on. `perch()` does exactly that to a threshold: it tries every candidate value across your scored pairs and shows you precisely what the linkage looks like at each one — match count, link rate, clerical burden — so you can commit to the perch that holds, informed by the same benchmarks the AIHW, WA Data Linkage Unit, and PHRN use in practice.

Sweeps a range of candidate Fellegi-Sunter threshold values across a scored pairs object and returns a structured sensitivity table with one row per candidate cutoff. Each row reports the number and percentage of pairs that would be accepted or rejected at that threshold, the number falling in a

configurable clerical review zone around it, and (if `n_records_df1` is supplied) the estimated link rate for the primary dataset. Four canonical thresholds are highlighted in the printed table and the plot with annotations from the three key Australian and international linkage authorities — AIHW, WA Data Linkage Unit, and the Population Health Research Network (PHRN).

Call `perch()` after `murmuration_plot` confirms your weight distribution is bimodal, then pass the chosen value to `murmuration(threshold_value = ...)`.

Usage

```
perch(
  pairs_pred,
  weight_col = "weights",
  n_records_df1 = NULL,
  thresholds = seq(5, 30, by = 1),
  clerical_window = 6,
  highlight_thresholds = c(10, 15, 17, 20),
  report = TRUE,
  plot = TRUE,
  palette = "sch",
  interactive = FALSE
)
```

Arguments

`pairs_pred`	A pairs object (data frame) as returned by `reclin2::predict.problink_em(m, pairs = pairs, add = TRUE)`, containing a numeric linkage weight column.
`weight_col`	Character. Name of the numeric linkage weight column. Default `"weights"` (the column name `murmuration` uses internally).
`n_records_df1`	Integer or `NULL`. Number of records in the primary (`df1`) dataset passed to `murmuration`. When supplied, a `link_rate` column is added showing the proportion of df1 records that would receive at least one accepted match at each threshold. Default `NULL` (link rate column omitted).
`thresholds`	Numeric vector of candidate threshold values to evaluate. Default `seq(5, 30, by = 1)`, covering the full practical range for Australian population health linkage with a complete variable set. Narrow this for targeted comparisons (e.g. `seq(14, 22, by = 0.5)`).
`clerical_window`	Numeric. Width of the clerical review zone centred on each threshold value: the band `[threshold - clerical_window/2, threshold + clerical_window/2]`. Pairs in this band are those that would change classification if the threshold shifted by half the window width — i.e. the marginal cases a human reviewer would be asked to adjudicate in a two-threshold design. Default `6` (matching the AIHW/WADLU convention of a roughly 10-unit clerical zone centred at each candidate threshold).
`highlight_thresholds`	Numeric vector. Threshold values to mark with `[*]` in the printed table and with diamond symbols in the plot. Default `c(10, 15, 17, 20)` — the four canonical

reference points from Australian and international linkage guidance (see Details).

`report` Logical. If `TRUE` (default), prints a formatted sensitivity table with reference annotations to the console.

`plot` Logical. If `TRUE` (default), generates and returns a **ggplot2** figure (or **patchwork** composite if `n_records_df1` is supplied) showing match count and link rate against threshold with AIHW zone shading and reference markers. If `FALSE`, returns only the data frame invisibly.

`palette` Character. Colour palette: `"sch"` (SCH Evergreen, default), `"default"` (teal/coral), `"grey"` (greyscale, publication-safe).

`interactive` Logical. If `TRUE`, returns a **plotly** htmlwidget with hover tooltips. Requires **plotly**. Default `FALSE`.

Details

Interpreting the output

Use `perch()` together with `murmuration_plot`:

1. `murmuration_plot()` confirms the weight distribution is bimodal and shows you where the valley between the two clusters lies.
2. `perch()` quantifies the match count, link rate, and clerical burden at each candidate cutoff in that valley.
3. Choose the threshold that best balances precision and recall for your study design (see benchmarks below), then pass it to `murmuration(threshold_value = ...)`.

The `n_clerical` column is the key diagnostic for threshold placement. A large `n_clerical` relative to `n_above` means the threshold sits in a high-density region of the score distribution — a small shift would reclassify many pairs. You want to sit in the valley (low density), not on a slope (high density). If every threshold in your range has a large clerical burden, the two score modes overlap substantially — this is a signal that the variable set or blocking strategy needs revision before threshold selection will be meaningful.

Australian and international linkage benchmarks

The Fellegi-Sunter weights are log-likelihood ratios and are **dataset-specific** — no universal cutoff exists. These benchmarks from recognised Australian and international linkage authorities inform where to look:

| Threshold | Authority and basis | |—|—| | **10–20 (clerical zone)** | **AIHW and WA Data Linkage Unit (WADLU)**: the established practice is a two-threshold approach with a clerical review zone of roughly 10–20. Pairs above ~20 are auto-accepted as matches; pairs below ~10 are auto-rejected as non-matches; pairs in between are sent for human adjudication. When operating without a dedicated clerical reviewer (as in most routine surveillance settings), the single threshold should sit somewhere in this range, with the direction of error dictated by the study design. | | **~15–20** | **Population Health Research Network (PHRN)**: no universal cutoff is mandated, but the operational target is a false-match rate below 0.5 | **12–17** | **starling balanced default** — typical starting point for SCPHU routine surveillance linkage with a complete variable set. | | **17–22** | **High specificity** — appropriate when false matches carry high analytical cost, e.g. vaccine effectiveness studies where falsely attributing vaccination status to an unvaccinated person

biases the VE estimate downward. | | **8–12** | **High sensitivity** — suitable for small datasets or rare-disease surveillance where missing a true match is the dominant concern. |

Note on link rate

The `link_rate` column approximates the proportion of primary-dataset records (`df1`) that would receive at least one accepted match at a given threshold. It counts accepted pairs above threshold rather than unique linked records (one df1 record can appear in multiple pairs), so it is an upper bound on the true link rate. Nonetheless it is useful for spotting the point at which raising the threshold begins to meaningfully reduce coverage — a sharp inflection in the link rate curve often marks the valley between the two score modes.

Value

When `plot = FALSE`, returns the sensitivity data frame invisibly. When `plot = TRUE` (default), prints the data frame to the console (if `report = TRUE`) and returns the plot object. The data frame has one row per threshold value and the following columns:

`threshold` Numeric. The candidate cutoff.

`n_above` Integer. Pairs with weight $\geq$ threshold (accepted as matches at this cutoff).

`n_below` Integer. Pairs with weight $<$ threshold (rejected as non-matches).

`pct_above` Numeric. Percentage of all pairs above threshold.

`n_clerical` Integer. Pairs in the clerical review zone [`threshold - clerical_window/2`, `threshold + clerical_window/2`].

`pct_clerical` Numeric. Percentage in the clerical zone.

`link_rate` Numeric or `NA`. Proportion of `n_records_df1` records that would receive a match. Only populated when `n_records_df1` is supplied.

`reference` Character. Annotation for the four highlighted threshold values; `NA` for all others.

References

Australian Institute of Health and Welfare (2021). *Data linkage: cohort studies*. AIHW, Canberra. https://www.aihw.gov.au/reports/methods-data-development/data-linkage

Holman, C.D.J. et al. (1999). A population-based linkage study of 3.4 million records in Western Australia. *Australian and New Zealand Journal of Public Health*, 23(5): 453–459. doi:10.1111/j.1467-842X.1999.tb01297.x

Population Health Research Network (2023). *PHRN Linkage Guidelines*. PHRN, Perth. https://www.phrn.org.au/

Fellegi, I. and Sunter, A. (1969). A theory for record linkage. *Journal of the American Statistical Association*, 64(328): 1183–1210. doi:10.2307/2286061

See Also

`murmuration` for the linkage step. `murmuration_plot` for visual weight distribution inspection. `preflight` for pre-linkage data quality auditing. `flock` for blocking variable construction.

Examples

```
## Not run:
# Standard workflow: score pairs then perch to find the best threshold
library(reclin2)

pairs <- pair_blocking(df1, df2, "block2")
compare_pairs(pairs,
  on = c("lettername1", "lettername2", "dob", "medicare10"),
  default_comparator = jaro_winkler(0.9), inplace = TRUE)
m          <- problink_em(
  ~ lettername1 + lettername2 + dob + medicare10, data = pairs)
pairs_pred <- predict(m, pairs = pairs, add = TRUE)

# Step 1: inspect the distribution
murmuration_plot(pairs_pred, threshold = 17)

# Step 2: quantify the trade-off across the plausible range
results <- perch(pairs_pred, n_records_df1 = nrow(df1))

# Step 3: narrow to the AIHW clerical zone for fine-grained inspection
results <- perch(pairs_pred,
  thresholds    = seq(10, 20, by = 0.5),
  n_records_df1 = nrow(df1))

# Get only the data frame (no plot)
tbl <- perch(pairs_pred, plot = FALSE)

# Interactive dual-panel plot for exploratory use
perch(pairs_pred, n_records_df1 = nrow(df1), interactive = TRUE)

# Greyscale for publication figures
perch(pairs_pred, palette = "grey")

# Step 4: apply the chosen threshold
linked <- murmuration(df1, df2,
  linkage_type  = "v2c",
  event_date    = "onset_date",
  blocking_var  = "block2",
  compare_vars  = c("lettername1", "lettername2", "dob", "medicare10"),
  threshold_value = 18)

## End(Not run)
```

`preflight` *Pre-Linkage Data Quality Checks*

Description

Bird note: Before a long migration, birds enter a physiological state called *hyperphagia* — they systematically assess and load up on resources, checking body condition before committing to

the journey. `preflight()` is that pre-migration check for your data: a structured, systematic audit of both datasets before committing to probabilistic linkage with `murmuration()`. Skipping it is like a shorebird launching across the Pacific without first checking its fat reserves.

Runs a battery of pre-linkage data quality checks on two datasets and returns a structured report. Checks cover: completeness of linkage variables, date plausibility, uniqueness of identifier columns, consistency of factor levels across datasets (e.g. gender coding), Medicare validity (if present), name length and character distribution, duplicate record detection, and dataset overlap summaries. Optionally runs `check_medicare` if a Medicare column is identified.

The report is printed to the console and returned invisibly as a named list so it can be saved, embedded in a Quarto report, or passed to downstream checks.

Usage

```
preflight(
  data1,
  data2,
  linkage_vars = NULL,
  id_col1 = "id_var",
  id_col2 = "id_var",
  blocking_vars = NULL,
  date_cols = NULL,
  medicare_col = NULL,
  name_cols = NULL,
  min_name_length = 2L,
  verbose = TRUE
)
```

Arguments

`data1`	First data frame (typically the case/index dataset, `df1` in `murmuration()`).
`data2`	Second data frame (typically the vaccination/hospitalisation dataset, `df2` in `murmuration()`).
`linkage_vars`	Character vector of column names that will be used as `compare_vars` in `murmuration()`. These are the variables `preflight()` will assess most carefully.
`id_col1`	Name of the unique record identifier in `data1`. Default `"id_var"`.
`id_col2`	Name of the unique record identifier in `data2`. Default `"id_var"`.
`blocking_vars`	Character vector of planned blocking variable names to check for completeness. If `NULL` (default), blocking variables are not checked.
`date_cols`	Character vector of date column names to check for plausibility (no future dates, no dates before 1900). `NULL` (default) skips date checks.
`medicare_col`	Name of the Medicare number column if present and to be validated. `NULL` (default) skips Medicare validation. If supplied, `check_medicare` is run on both datasets and results summarised here.
`name_cols`	Character vector of name columns to check for short strings (potential initials or truncations), all-numeric entries, and excessive punctuation. Default checks `"lettername1"` and `"lettername2"` if present.

`min_name_length`
: Integer. Names shorter than this are flagged as potentially truncated or coded. Default `2`.

`verbose`
: Logical. Print the full report to console? Default `TRUE`.

Value

Invisibly, a named list with the following elements:

`completeness` A data frame: variable, dataset, n_present, n_missing, pct_missing for each linkage variable.

`duplicates` A data frame: dataset, n_records, n_unique_ids, n_duplicate_ids, pct_duplicate.

`blocking_completeness` Completeness of blocking variables, or `NULL` if not checked.

`date_checks` Date plausibility results, or `NULL` if not checked.

`medicare` Medicare validity results from `check_medicare()`, or `NULL` if not checked.

`name_checks` Name quality results, or `NULL` if not checked.

`factor_consistency` Comparison of factor levels for shared categorical columns.

`overlap_summary` Approximate overlap statistics for key demographic variables.

`flags` Character vector of warning-level flags raised during the checks. Empty if no issues found.

See Also

`check_medicare`, `murmuration`, `flock`

Examples

```
## Not run:
# Basic pre-flight check before v2c linkage
report <- preflight(
  data1         = cases_clean,
  data2         = vax_clean,
  linkage_vars  = c("lettername1", "lettername2", "dob", "medicare10"),
  id_col1       = "id_var",
  id_col2       = "id_var",
  blocking_vars = "gender",
  date_cols     = c("dob", "onset_date"),
  medicare_col  = "medicare"
)

# Access the completeness table programmatically
report$completeness

# Save the report to a CSV for documentation
write.csv(report$completeness, "preflight_completeness.csv", row.names = FALSE)

## End(Not run)
```

vax_air *Synthetic Australian Immunisation Register Extract*

Description

Bird note: This is the second flock — the AIR records whose individual birds must be matched to their counterparts in the case linelist. Some will pair cleanly; others will remain unpaired because the case was not vaccinated or vaccination was administered elsewhere.

A synthetic extract from the Australian Immunisation Register (AIR) for use in starling vignettes, documentation examples, and tests. No real person data. 400 records; approximately 200 are true matches to records in `cases_notifiable`. All Medicare numbers have valid checksums. Each person may have 1–4 vaccination doses stored in wide format (`vax_date_1`–`vax_date_4`, `vax_type_1`–`vax_type_4`).

Usage

```
data(vax_air)
```

Format

A data frame with 400 rows and 15 columns:

id_var character. Unique AIR identifier (`"AIR_00001"`, etc.).

lettername1 character. First name.

lettername2 character. Surname.

dob Date. Date of birth.

gender character. `"Male"` or `"Female"`.

postcode character. Sunshine Coast postcode (4550–4562).

medicare10 character. 10-digit Medicare number. All records have valid Modulus 10 checksums (`check_medicare` returns `1L` for every row).

vax_date_1 Date. Date of first vaccination dose. Never `NA`.

vax_type_1 character. Vaccine type for dose 1.

vax_date_2 Date. Date of second dose; `NA` if person received only one dose.

vax_type_2 character. Vaccine type for dose 2; `NA` if absent.

vax_date_3 Date. Date of third dose; `NA` if fewer than three doses.

vax_type_3 character. Vaccine type for dose 3; `NA` if absent.

vax_date_4 Date. Date of fourth dose; `NA` if fewer than four doses.

vax_type_4 character. Vaccine type for dose 4; `NA` if absent.

Source

Generated by `data-raw/starling_synthetic_data.R`. Run `source("data-raw/starling_synthetic_data.R")` from the package root to rebuild.

See Also

cases_notifiable for the paired case linelist. vignette("linked-cohort", package = "starling") for a worked end-to-end linkage example using both datasets.

Examples

```
## Not run:
data(vax_air)
head(vax_air)

# Confirm all Medicare numbers pass checksum
vax_checked <- check_medicare(vax_air)
mean(vax_checked$medicare_valid == 1L, na.rm = TRUE)  # should be ~1.0

# Distribution of dose counts
n_doses <- rowSums(!is.na(vax_air[, paste0("vax_date_", 1:4)]))
table(n_doses)

## End(Not run)
```

Index