Package: epidm (via r-universe)

May 27, 2026

Version 1.0.6

Title UK Epidemiological Data Management

Description Contains utilities and functions for the cleaning, processing and management of patient level public health data for surveillance and analysis held by the UK Health Security Agency, UKHSA.

URL https://github.com/ukhsa-collaboration/epidm

BugReports https://github.com/ukhsa-collaboration/epidm/issues

License GPL (>= 3)

Depends R (>= 4.4.0)

Imports data.table, DBI, odbc, phonics, purrr, readr, stats, stringi, stringr, utils, lubridate, knitr, rmarkdown, ggplot2, DiagrammeR

Encoding UTF-8

LazyData true

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.3

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

Config/testthat/edition 3

VignetteBuilder knitr

Repository https://ukhsa-collaboration.r-universe.dev

Date/Publication 2026-02-19 10:39:04 UTC

RemoteUrl https://github.com/ukhsa-collaboration/epidm

RemoteRef HEAD

RemoteSha e0d0ac04a1ddd2f3569d6b634393e24e073ac555

Contents

`cip_spells` *Continuous Inpatient (CIP) Spells*

Description

[Stable]

Creates **Continuous Inpatient (CIP) spells** by combining one or more provider spells into a single uninterrupted period of inpatient care. CIP definitions follow the NHS Digital methodology: transfers between providers can be part of the same CIP spell where specific admission, discharge, and timing criteria are met. A CIP spell begins when a patient is admitted under consultant care and ends when they are discharged or die. http://content.digital.nhs.uk/media/11859/Provider-Spells-Methodology/pdf/Spells_Methodology.pdf

Where spells meet the CIP criteria, they are merged into a continuous spell. The output includes a CIP index and the derived start and end dates for the full CIP period.

Usage

```
cip_spells(
  x,
  group_vars,
  spell_start_date,
```

```
  admission_method,
  admission_source,
  spell_end_date,
  discharge_destination,
  patient_classification,
  .forceCopy = FALSE
)
```

Arguments

x	A data.frame or data.table; will be converted to a data.table (usually HES/SUS data)
group_vars	Character vector of variables used to group records (minimum: a patient identifier).
spell_start_date	Quoted column name containing the provider spell admission date.
admission_method	CDS admission method code.
admission_source	CDS admission source code.
spell_end_date	Quoted column name containing the provider spell discharge date.
discharge_destination	CDS discharge destination code.
patient_classification	CDS patient classification code.
.forceCopy	Logical (default FALSE). If FALSE, the input is converted to a data.table and modified by reference. If TRUE, the input must already be a data.table, and the function will create an explicit copy to avoid modifying the original object.

Value

A data.table containing the original data and three new variables:

cip_indx Unique identifier for the derived CIP spell.

cip_spell_start Start date for the continuous inpatient spell.

cip_spell_end End date for the continuous inpatient spell.

Examples

```
cip_test <- data.frame(
  id = c('465','465','465','465','8418','8418','8418',
         '8418','8418','8418','8418','8418','26443',
         '26443','26443','33299','33299','33299','33299',
         '33299','33299','33299','33299','33299','33299',
         '52635','52635','52635','52635','52635','52635',
         '52635','52635','52635','52635','52635','52635',
         '52635','52635','52635','52635','52635','52635',
         '52635','52635','52635','52635','52635','52635',
```

```
       '52635','52635','52635','78915','78915','78915'),
provider = c('X1T','X1T','X1T','X1T','KHA','KHA','KHA',
             'KHA','KHA','KHA','KHA','KHA','BX2','BX2',
             'BX2','PXH','PXH','PXH','PXH','PXH','PXH',
             'PXH','PXH','PXH','PXH','9HA','9HA','9HA',
             '9HA','9HA','9HA','9HA','9HA','9HA','9HA',
             '9HA','9HA','9HA','9HA','9HA','9HA','YYT',
             'YYT','YYT','YYT','YYT','YYT','YYT','YYT',
             'YYT','YYT','YYT','ABX','ABX','ABX'),
spell_start = as.Date(c(
  '2020-03-07','2020-03-07','2020-03-25','2020-04-03','2020-01-25',
  '2020-01-26','2020-07-14','2020-08-02','2020-08-12','2020-08-19',
  '2020-08-19','2020-11-19','2019-11-12','2020-04-17','2020-04-23',
  '2020-07-03','2020-01-17','2020-02-07','2020-03-20','2020-04-27',
  '2020-06-21','2020-07-02','2020-10-17','2020-11-27','2021-01-02',
  '2019-12-31','2020-01-02','2020-01-14','2020-01-16','2020-02-07',
  '2020-02-11','2020-02-14','2020-02-18','2020-02-21','2020-02-25',
  '2020-02-28','2020-03-09','2020-03-11','2020-03-12','2020-03-13',
  '2020-03-14','2020-02-04','2020-02-07','2020-02-11','2020-02-14',
  '2020-02-18','2020-02-21','2020-02-25','2020-02-28','2020-03-09',
  '2020-03-11','2020-03-12','2020-04-16','2020-04-24','2020-05-13')),
spell_end = as.Date(c(
  '2020-03-07','2020-03-25','2020-04-02','2020-04-27','2020-01-25',
  '2020-01-27','2020-07-17','2020-08-07','2020-08-14','2020-08-19',
  '2020-08-22','2020-12-16','2020-04-17','2020-04-23','2020-05-20',
  '2020-07-24','2020-01-28','2020-02-07','2020-03-23','2020-04-29',
  '2020-06-21','2020-07-03','2020-11-27','2021-01-02','2021-01-10',
  '2019-12-31','2020-01-11','2020-01-14','2020-02-04','2020-02-07',
  '2020-02-11','2020-02-14','2020-02-18','2020-02-21','2020-02-25',
  '2020-02-28','2020-03-09','2020-03-11','2020-03-12','2020-03-13',
  '2020-03-30','2020-02-07','2020-02-11','2020-02-14','2020-02-18',
  '2020-02-21','2020-02-25','2020-02-28','2020-03-09','2020-03-11',
  '2020-03-12','2020-03-13','2020-04-24','2020-05-13','2020-06-11')),
adm_meth = c('21','81','21','81','21','21','11','21','21','21','21',
             '21','21','81','21','81','21','21','21','21','21','21',
             '21','13','13','12','22','12','2D','13','13','13','13',
             '13','13','13','13','13','13','13','21','81','81','81',
             '81','81','13','81','81','13','13','13','21','11','81'),
adm_src = c('19','51','19','51','19','51','19','51','19','19','19',
            '51','19','51','19','51','19','19','19','19','19','19',
            '19','51','19','19','19','19','19','19','19','19','19',
            '19','19','19','51','51','51','51','19','51','51','51',
            '51','51','51','51','51','51','51','51','19','51','51'),
dis_meth = c('1','1','1','1','1','1','1','1','1','1','1','4','1','1',
             '4','1','1','1','1','1','1','1','8','1','4','1','1','1',
             '1','1','1','1','1','1','1','1','1','1','1','1','1','1',
             '1','1','1','1','1','1','1','1','1','1','1','1','1','2'),
dis_dest = c('51','51','51','54','51','19','19','19','19','51','19',
             '79','51','51','79','65','19','19','19','19','19','29',
             '98','51','79','19','19','19','51','19','19','19','51',
             '51','51','19','19','51','51','19','51','51','51','51',
             '51','51','51','51','51','51','51','51','29','54','19'),
patclass = c('1','1','1','1','1','1','1','1','1','1','1','1','1','1','1',
```

```
                '1','1','1','1','1','1','1','1','1','1','1','2','1','2',
                '1','2','2','2','2','2','2','2','2','2','2','2','1','1',
                '1','1','1','1','1','1','1','1','1','1','1','1','1')
  )

  cip_spells(x=cip_test,
    group_vars = c('id','provider'),
    patient_classification = 'patclass',
    spell_start_date = 'spell_start',
    admission_method = 'adm_meth',
    admission_source = 'adm_src',
    spell_end_date = 'spell_end',
    discharge_destination = 'dis_dest'
  )[]
```

`csv_from_zip` *Download a csv from a zip*

Description

[Stable] A convenience function to allow you to pull data from NHS, ONS and ODR assets

Usage

```
csv_from_zip(x)
```

Arguments

x	a zip file from the web

Value

a zip file for ingestion into your chosen readr

Examples

```
## Not run:
read.csv(csv_from_zip("https://files.digital.nhs.uk/assets/ods/current/pcodeall.zip"))

## End(Not run)
```

genus_gram_stain *Bacterial Genus Gram Stain Lookup Table*

Description

A reference table of bacterial gram stain results by genus to allow faster filtering of bacterial results. This dataset has been maintained manually against the PHE SGSS database. If there are organisms missing, please raise and issue or push request on the epidm GitHub

Usage

```
genus_gram_stain
```

Format

A data frame with four columns

organism_genus The bacterial genus

gram_stain A character string to indicate POSITIVE or NEGATIVE type

gram_positive A 0/1 flag to indicate if the genus is gram positive

gram_negative A 0/1 flag to indicate if the genus is gram negative

group_ecds_discharge_destination *A&E attendance discharge destination*

Description

In order to group A&E discharge destination from SNOWMED into human readable groups, a lookup table has been created. These work with Emergency Care Dataset (ECDS) data with the destination_code field to show where a patient goes after discharge from A&E.

Usage

```
group_ecds_discharge_destination
```

Format

code the ECDS destination_code

destination_code the destination grouping as a human readable string

group_inpatient_admission_method

Inpatient admission methods

Description

In order to group hospital inpatient admissions into human readable groups, a lookup table has been created. These work with Hospital Episode Statistics (HES) and Secondary Use Services (SUS) data with the admission_method fields.

Usage

```
group_inpatient_admission_method
```

Format

code the admission_method code

admission_method the admission_method grouping as a human readable string

group_inpatient_discharge_destination

Inpatient discharge destination

Description

In order to group hospital inpatient discharge destination into human readable groups, a lookup table has been created. These work with Hospital Episode Statistics (HES) and Secondary Use Services (SUS) data with the discharge_destination fields.

Usage

```
group_inpatient_discharge_destination
```

Format

code the discharge_destination code

discharge_destination the discharge_destination grouping as a human readable string

group_time *Grouping of intervals or events that occur close together in time*

Description

[Stable]

A utility function to group together observations that represent **overlapping date intervals** (e.g., hospital admission spells) or **events occurring within a defined time window** e.g., specimen dates grouped into infection episodes. The function supports both:

• **Interval-based grouping**: records have a start and an end date; any overlapping intervals are grouped together. • **Event-based grouping**: records have only a start date and are grouped using either a *static* or *rolling* time window.

The output provides a unique index per group and the minimum/maximum date that define the resulting aggregated episode or interval.

Usage

```
group_time(
  x,
  date_start,
  date_end,
  window,
  window_type = c("rolling", "static"),
  group_vars,
  indx_varname = "indx",
  min_varname = "date_min",
  max_varname = "date_max",
  .forceCopy = FALSE
)
```

Arguments

x	A data.frame or data.table containing date variables for grouping. Will be converted to a data.table internally.
date_start	Quoted column name giving the start date for each record.
date_end	column containing the end dates for the *interval*, quoted
window	an integer representing a time window in days which will be applied to the start date for grouping *events*
window_type	character, to determine if a 'rolling' or 'static' grouping method should be used when grouping *events*. A *'static'* window will identify the first event, and all records X days from that event will be attributed to the same episode. Eg. in a 14 day window, if first event is on 01 Mar, and events on day 7 Mar and 14 Mar will be grouped, but an event starting 15 Mar days after will start a new episode. A *'rolling'* window resets the day counter with each new event. Eg. Events on 01 Mar, 07 Mar, 14 Mar and 15 Mar are all included in a single episode, as will any additional events up until the 29 Mar (assuming a 14-day window).

group_vars	Character vector of quoted column names used to partition the data before grouping.
indx_varname	a character string to set variable name for the index column which provides a grouping key; default is indx
min_varname	a character string to set variable name for the time period minimum
max_varname	a character string set variable name for the time period maximum
.forceCopy	default FALSE; TRUE will force data.table to take a copy instead of editing the data without reference

indx**; renamed using** indx_varname an id field for the new aggregated events/intervals; note that where the date_start is NA, an indx value will also be NA

min_date**; renamed using** min_varname the start date for the aggregated events/intervals

max_date**; renamed using** max_varname the end date for the aggregated events/intervals

Details

How the function works:

The behaviour depends on whether date_end is supplied:

1. **Interval-based grouping (start + end dates)***:*
If both date_start and date_end are provided, the function identifies overlapping intervals within the same group_vars grouping. Any intervals that overlap are combined into a single episode.
This method is typically used for:

- Hospital spells (HES/SUS)
- Contact periods or inpatient stays

2. **Event-based grouping (single-date events)***:*
If only date_start is supplied, records are grouped using a **time window** defined by the window argument.
Two approaches are supported:

- window_type = "static" A fixed window is applied starting from the first event in the group. All events occurring within the window are grouped until a gap exceeds the threshold, at which point a new episode begins.
- window_type = "rolling" A dynamic window where each event extends the episode end point. An event is grouped as long as it occurs within window days of the most recent event in the same episode.

Handling of missing values:

Records missing date_start cannot be grouped and are returned with indx = NA. These rows are appended back to the final output.

Value

A data.table containing all original columns plus:

Workflow context

how `group_time()` might be used in a pipeline **1) SGSS specimen data – infection episode grouping (event-based)** After organism/specimen harmonisation (e.g., via `lookup_recode()`), `group_time()` groups specimen dates into infection episodes using a defined time window. This helps identify clusters of related positive tests for the same patient and organism.

2) HES/SUS inpatient data – continuous spell grouping (interval-based) When start and end dates of inpatient stays are available, `group_time()` collapses overlapping intervals into a single continuous hospital spell. This is used before linking SGSS infection episodes to inpatient activity.

3) Integration across datasets The outputs from `group_time()` are used downstream to determine whether infection events fall within or around periods of hospital care, enabling combined SGSS–HES/SUS–ECDS analyses.

Examples

```
episode_test <- structure(
  list(
    pat_id = c(1L, 1L, 1L, 1L, 2L, 2L, 2L,
               1L, 1L, 1L, 1L, 2L, 2L, 2L),
    species = c(rep("E. coli",7),rep("K. pneumonia",7)),
    spec_type = c(rep("Blood",7),rep("Blood",4),rep("Sputum",3)),
    sp_date = structure(c(18262, 18263, 18281, 18282, 18262, 18263, 18281,
                          18265, 18270, 18281, 18283, 18259, 18260, 18281),
                        class = "Date")
  ),
  row.names = c(NA, -14L), class = "data.frame")

group_time(x=episode_test,
           date_start='sp_date',
           window=14,
           window_type = 'static',
           indx_varname = 'static_indx',
           group_vars=c('pat_id','species','spec_type'))[]

spell_test <- data.frame(
  id = c(rep(99,6),rep(88,4),rep(3,3)),
  provider = c("YXZ",rep("ZXY",5),rep("XYZ",4),rep("YZX",3)),
  spell_start = as.Date(
    c(
      "2020-03-01",
      "2020-07-07",
      "2020-02-08",
      "2020-04-28",
      "2020-03-15",
      "2020-07-01",
      "2020-01-01",
      "2020-01-12",
      "2019-12-25",
      "2020-03-28",
      "2020-01-01",
      rep(NA,2)
    )
```

```
  ),
  spell_end = as.Date(
    c(
      "2020-03-10",
      "2020-07-26",
      "2020-05-22",
      "2020-04-30",
      "2020-05-20",
      "2020-07-08",
      "2020-01-23",
      "2020-03-30",
      "2020-01-02",
      "2020-04-20",
      "2020-01-01",
      rep(NA,2)
    )
  )
)

group_time(x = spell_test,
           date_start = 'spell_start',
           date_end = 'spell_end',
           group_vars = c('id','provider'),
           indx_varname = 'spell_id',
           min_varname = 'spell_min_date',
           max_varname = 'spell_max_date')[]
```

`hospital_in_out_dates` *Hospital IN/OUT dates*

Description

[Experimental]

Derives per-patient **hospital entry** (`hospital_in`) and **exit** (`hospital_out`) dates by reconciling A&E (ECDS) attendances and inpatient (HES/SUS) spells. Applies a simple ranking to determine the most relevant hospital period around an index event date (e.g., a specimen collection date).

"1" Current admissions take priority

"2" When conflicting on the same day, inpatient admissions take priority over A&E emergency care data

"3" Where a patient has a linked A&E admission to a hospital inpatient stay, the A&E admission date is used

"4" Where a patient has a positive test between two hospital stays the most recent completed hospital stay prior to the test is retained except if the time between these events is greater than 14 days, then the first admission following the test is retained

Usage

```
hospital_in_out_dates(
  data,
  person_id = "id",
 hospital = list(org_code = "organisation_code_of_provider", event_date = "ev_date",
    ae_arrive = "arrival_date", ae_depart = "departure_date", ae_discharge =
    "ecds_discharge", in_spell_start = "spell_start_date", in_spell_end =
    "spell_end_date", in_discharge = "discharge_destination")
)
```

Arguments

data	A linked table containing A&E and inpatient records (typically the output of `link_ae_inpatient()`), including person/event identifiers and date fields.
person_id	Quoted column name for the unique patient identifier.
hospital	A named **list** specifying column names (all quoted) for:
	`org_code` Organisation code (optional; used to scope grouping).
	`event_date` Index date to compare against (e.g., `specimen_date`).
	`ae_arrive` ECDS arrival date.
	`ae_depart` ECDS departure date.
	`ae_discharge` ECDS discharge status (use grouped values if available).
	`in_spell_start` Inpatient spell start date.
	`in_spell_end` Inpatient spell end date.
	`in_discharge` Inpatient discharge destination (grouped recommended).

Value

A `data.table` equal to `data` with additional columns:

`hospital_in` Derived hospital admission date for the relevant stay.

`hospital_out` Derived hospital discharge date for the relevant stay.

`hospital_event_rank` Rank of suitability of the hospital window for the given person/event (1 = most suitable).

Workflow context

Use `hospital_in_out_dates()` **after**:

- Linking A&E to inpatient spells (e.g., via `link_ae_inpatient()`),
- Constructing spells (e.g., `group_time()` or `cip_spells()`),
- Optional code standardisation (e.g., discharge groups via `lookup_recode()`)

See Also

epidm::lookup_recode()

epidm::group_time()

epidm::cip_spells()

Examples

```
## Not run:
hospital_in_out_dates(link,
person_id = 'id',
hospital = list(
  org_code = 'organisation_code_of_provider',
  event_date = 'ev_date',
  ae_arrive = 'arrival_date',
  ae_depart = 'departure_date',
  ae_discharge = 'ecds_discharge',
  in_spell_start = 'spell_start_date',
  in_spell_end = 'spell_end_date',
  in_discharge = 'discharge_destination'
))[]

## End(Not run)
```

---

inpatient_codes *Inpatient Codes cleanup*

Description

[Experimental]

When HES/SUS ICD/OPCS codes are provided in wide format you may want to clean them up into long for easier analysis. This function helps by reshaping long as a separate table. Ensuring they're separate allows you to retain source data, and aggregate appropriately later.

Usage

```
inpatient_codes(
  x,
  field_strings,
  patient_id_vars,
  type = c("icd9", "icd10", "opcs"),
  .forceCopy = FALSE
)
```

Arguments

x	a data.frame or data.table containing inpatient data
field_strings	a vector or string containing the regex for the the columns
patient_id_vars	a vector containing colnames used to identify a patient episode or spell
type	a string to denote if the codes are diagnostic or procedural
.forceCopy	Logical (default FALSE). If FALSE, the input is converted to a data.table and modified by reference. If TRUE, the input must already be a data.table, and the function will create an explicit copy to avoid modifying the original object.

Value

a separate table with codes and id in long form

Examples

```
# Example inpatient dataset
inpatient_test <- data.frame(
  id = c(1053L, 5487L, 8180L),
  spell_id = c("dwPDw", "iSpUq", "qpgk5"),
  primary_diagnosis_code = c("K602", "U071-", "I501"),
  procedure_code = c("H201", "H251", NA),
  procedure_date = as.Date(c("2023-01-01", "2023-01-04", NA))
)

# ICD-10 cleaning example
inpatient_codes(
  x = inpatient_test,
  field_strings = "diagnosis",
  patient_id_vars = c("id", "spell_id"),
  type = "icd10"
)

# OPCS cleaning example
inpatient_codes(
  x = inpatient_test,
  field_strings = c("procedure_code", "procedure_date"),
  patient_id_vars = c("id", "spell_id"),
  type = "opcs"
)
```

---

lab_data *Synthetic Lab Data for epidm*

Description

A dataset containing synthetic lab data for testing epidemiological data transformation functions.

A dataset containing synthetic lab data for testing epidemiological data transformation functions.

Usage

```
data(lab_data)
```

```
data(lab_data)
```

Format

A data frame with the following columns:

nhs_number NHS number

local_patient_identifier Patient identifier such as hospital number

patient_birth_date Date of birth of the patients.

sex Gender of the patients (Factor with levels: "Female", "Male").

surname Patient surname

forename Patient forename

organism_species_name Organism species name (Factor with levels: "KLEBSIELLA PNEUMONIAE").

specimen_date Date of specimen collection.

specimen_type Type of specimen: BLOOD or URINE.

lab_code Laboratory codes (Factor with unique levels).

local_authority_name Name of the local authority.

local_authority_code Code of the local authority.

postcode Postcode

A data frame with the following columns:

nhs_number NHS number

local_patient_identifier Patient identifier such as hospital number

patient_birth_date Date of birth of the patients.

sex Gender of the patients (Factor with levels: "Female", "Male").

surname Patient surname

forename Patient forename

organism_species_name Organism species name (Factor with levels: "KLEBSIELLA PNEUMONIAE").

specimen_date Date of specimen collection.

specimen_type Type of specimen: BLOOD or URINE.

lab_code Laboratory codes (Factor with unique levels).

local_authority_name Name of the local authority.

local_authority_code Code of the local authority.

postcode Postcode

Examples

```
data(lab_data)
head(lab_data)

data(lab_data)
head(lab_data)
```

`lookup_recode` *Lookup table switch handler*

Description

[Stable]

A function to recode values via named lookup tables (i.e call an epidm lookup table and recode where we are aware of a new value). It routes to a specific lookup based on type, returning a character vector where each input value has been mapped to its corresponding replacement. If a value is not found in the lookup then the original value is returned.

Built-in lookups include:

- `species`: Uses the `respeciate_organism` dataset to standardise and reclassify organism names (e.g., historic → current nomenclature). This supports consistent reporting across SGSS and other laboratory datasets.
- `specimen`: Uses the `specimen_type_grouping` dataset to assign raw laboratory specimen types into harmonised specimen groups. This enables consistent grouping for reporting, aggregation, and filtering.
- `genus_gram_stain`: Uses the `genus_gram_stain` lookup table, which provides Gram stain classifications by bacterial genus. This reference is manually maintained against the UKHSA SGSS database and supports rapid filtering and high-level organism categorisation. Users should raise an issue or submit a pull request to the `epidm` GitHub repository if an organism/genus is missing.
- `lab_data`: Uses the `lab_data` lookup dataset for harmonising laboratory code systems and internal SGSS mappings, supporting standardised laboratory result interpretation within surveillance pipelines.
- `inpatient_admission_method`: Uses the internal lookup table `epidm:::group_inpatient_admission_method` to categorise raw hospital admission method codes into operationally meaningful groups.
- `inpatient_discharge_destination`: Uses the internal table `epidm:::group_inpatient_discharge_destination` to group hospital discharge destination codes into standardised categories for inpatient pathway analysis.
- `ecds_destination_code`: Uses the internal table `epidm:::group_ecds_discharge_destination`, providing grouped mappings for ECDS (Emergency Care Data Set) discharge codes.
- `manual`: Allows the user to supply their own lookup through `.import = list(new, old)`. This is useful when working with local, provisional, or evolving code sets not yet included in the package's centralised lookup tables.

Usage

```
lookup_recode(
  src,
  type = c("species", "specimen", "inpatient_admission_method",
    "inpatient_discharge_destination", "ecds_destination_code", "manual"),
  .import = NULL
)
```

Arguments

src	Character vector (or column) of values to recode. Coerced to character if needed.
type	Character scalar specifying the lookup to use. One of: 'species', 'specimen', 'inpatient_admission_method', 'inpatient_discharge_destination', 'ecds_destination_cod 'manual'.
.import	A two-element list in the format list(new, old) used only when type = 'manual'. Each element must be a vector of equal length.

Value

A character vector containing the recoded values, aligned 1:1 with src. Values not present in the lookup are returned unchanged.

Examples

```
df <- data.frame(
  spec = c(
    sample(grep(")",
                respeciate_organism$previous_organism_name,
                value=TRUE,
                invert = TRUE),
           9),
    "ESCHERICHIA COLI","SARS-COV-2","CANDIDA AUREUS"),
  type = sample(specimen_type_grouping$specimen_type,12),
  date = sample(seq.Date(from = Sys.Date()-365,
                         to = Sys.Date(),
                         by = "day"),12)
)
df <- df[order(df$date),]

# show the data before the changes
df

# check the lookup tables
# observe the changes
head(respeciate_organism[1:2])
df$species <- lookup_recode(df$spec,'species')
df[,c('spec','species','date')]

head(specimen_type_grouping)
df$grp <- lookup_recode(df$type,'specimen')
df[,c('species','type','grp','date')]

# for a tidyverse use
# df %>% mutate(spec=lookup_recode(spec,'species))

# manual input of your own lookup
# .import=list(new,old)
lookup_recode(
  "ALCALIGENES DENITRIFICANS",
  type = 'manual',
```

```
  .import=list(respeciate_organism$organism_species_name,
               respeciate_organism$previous_organism_name)
  )
```

`proxy_episode_dates` *Clean and Impute HES/SUS Episode Start and End Dates*

Description

[Stable]

A utility for cleaning and imputing missing or inconsistent episode end dates in HES/SUS–style inpatient data. The function identifies missing, invalid, or overlapping spell dates within patient/provider groups and applies deterministic rules to correct them. It also assigns a flag (`proxy_missing`) indicating whether a value was modified and why.

Usage

```
proxy_episode_dates(
  x,
  group_vars,
  spell_start_date,
  spell_end_date,
  discharge_destination,
  .dropTmp = TRUE,
  .forceCopy = FALSE
)
```

Arguments

`x`	A `data.frame` or `data.table`. Will be converted to a `data.table` if not already.
`group_vars`	Character vector of grouping variables (e.g., patient ID, provider). At least one identifier must be supplied.
`spell_start_date`	Name of the column containing the episode or spell start date.
`spell_end_date`	Name of the column containing the episode or spell end date.
`discharge_destination`	Name of the column containing the CDS discharge destination code.
`.dropTmp`	Logical (default `TRUE`). If `TRUE`, temporary processing columns are removed before returning the result.
`.forceCopy`	Logical (default `FALSE`). If `FALSE`, the input is converted to a `data.table` and modified by reference. If `TRUE`, the input must already be a `data.table`, and the function will create an explicit copy to avoid modifying the original object.

Value

A `data.table` containing:

- Cleaned spell start and end dates.
- A flag variable (`proxy_missing`) indicating whether a date was modified and the rule applied (0–4).

Examples

```
proxy_test <- data.frame(
  id = c(
    rep(3051, 4),
    rep(7835,3),
    rep(9891,3),
    rep(1236,3)
  ),
  provider = c(
    rep("QKJ", 4),
    rep("JSD",3),
    rep("YJG",3),
    rep("LJG",3)
  ),
  spell_start = as.Date(c(
    "2020-07-03", "2020-07-14", "2020-07-23", "2020-08-05",
    "2020-11-01", "2020-11-13", "2020-12-01",
    "2020-03-28", "2020-04-06", "2020-04-09",
    "2020-10-06", "2020-11-05", "2020-12-25"
  )),
  spell_end = as.Date(c(
    "2020-07-11", "2020-07-22", "2020-07-30", "2020-07-30",
    "2020-11-11", NA, "2020-12-03",
    "2020-03-28", NA, "2020-04-09",
    "2020-10-06", "2020-11-05", NA
  )),
  disdest = c(
    19, 19, 51, 19,
    19, 19, 19,
    51, 98, 19,
    19, 19, 98
  )
)


proxy_episode_dates(
  x=proxy_test,
  group_vars = c('id','provider'),
  spell_start_date = 'spell_start',
  spell_end_date = 'spell_end',
  discharge_destination = 'disdest'
)[]
```

respeciate_generic *Respeciate unspecified samples*

Description

[Stable]

Some samples within SGSS are submitted by laboratories as "GENUS SP" or "GENUS UNNAMED". However, they may also have a fully identified sample taken from the same site within a recent time period. This function captures species_col from another sample within X-days of an unspeciated isolate. Respeciation is restricted to organisms of the same genus; species will not be inferred from isolates belonging to a different genus. Trailing "UNNAMED" is normalised to "SP" before any processing.

Usage

```
respeciate_generic(
  x,
  group_vars,
  species_col,
  date_col,
  window = c(0:Inf),
  .forceCopy = FALSE
)
```

Arguments

x	a data.frame or data.table object
group_vars	the minimum grouping set of variables for like samples in a character vector; suggest c('patient_id','specimen_type') - genus will automatically be included in the groupby. This is built from the species_col
species_col	a character containing the column with the organism species_col name
date_col	a character containing the column with the specimen/sample date_col
window	an integer representing the number of days for which you will allow a sample to be respeciated
.forceCopy	Logical (default FALSE). If FALSE, the input is converted to a data.table and modified by reference. If TRUE, the input must already be a data.table, and the function will create an explicit copy to avoid modifying the original object.

Value

a data.table with a recharacterised species_col column

Examples

```
df <- data.frame(
ptid = c(round(runif(25,1,5))),
spec = sample(c("KLEBSIELLA SP",
                "KLEBSIELLA UNNAMED",
                "KLEBSIELLA PNEUMONIAE",
                "KLEBEIELLA OXYTOCA"),
              25,replace = TRUE),
type = "BLOOD",
specdate = sample(seq.Date(Sys.Date()-21,Sys.Date(),"day"),25,replace = TRUE)
)

respeciate_generic(x=df,
                   group_vars=c('ptid','type'),
                   species_col='spec',
                   date_col='specdate',
                   window = 14)[]
```

---

respeciate_organism *Respeciated organisms*

Description

Occasionally, research shows that two organisms, previously thought to be different are in fact one and the same. The reverse is also true. This is a manually updated list. If there are organisms missing, or new respeciates to be added, please raise and issue or push request on the epidm GitHub

Usage

```
respeciate_organism
```

Format

previous_organism_name What the organism used to be known as, in the form GENUS SPECIES

organism_species_name What the organism is known as now, in the form GENUS SPECIES

organism_genus_name The genus of the recoded organism

genus_change A 0/1 flag to indicate if the genus has changed

genu_all_species A 0/1 flag to indicate if all species under that genus should change

specimen_type_grouping *Specimen type grouping*

Description

In order to help clean up an analysis based on a group of specimen types, a lookup table has been created to help group sampling sites. This is a manually updated list. If there are organisms missing, or new respeciates to be added, please raise and issue or push request on the epidm GitHub

Usage

```
specimen_type_grouping
```

Format

specimen_type The primary specimen type with detail

specimen_group A simple grouping of like specimen sites

sql_clean *Clean and Read a SQL query*

Description

[Stable]

A utility function to read in a SQL query from a character object, clipboard or text file and remove all comments for use with database query packages

Usage

```
sql_clean(sql)
```

Arguments

sql	a SQL file or text string

Value

a cleaned SQL query without comments as a character string

Examples

```
testSQL <- c(
"/********* INTRO HEADER COMMENTS",
"*********/",
"  SELECT ",
"  [VAR 1]  -- with comments",
",[VAR 2]",",[VAR 3]",
"FROM DATASET ","-- output here")
sql_clean(testSQL)
```

`sql_connect` *Connect to a SQL database*

Description

[Stable]

An function to help setup connections to SQL databases acting as a wrapper for the odbc and DBI packages. Used by other sql_* tools within epidm. This uses the credential manager within the system and assumes you are using a trusted connection.

Usage

```
sql_connect(server, database)
```

Arguments

`server`	a string containing the server connection; note that servers may require the use of double backslash \\
`database`	a string containing the database name within the data store

Value

a SQL connection object

See Also

sql_clean sql_read sql_write

Examples

```
## Not run:
sql <- list(
  dsn = list(ser = 'covid.ukhsa.gov.uk',
             dbn = 'infections')
)

sgss_con = sql_connect(server = sql$dsn$ser, database = sql$dsn$dbn)
```

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

`sql_read` *Read a table from a SQL database*

Description

[Stable]

Read a table object to a SQL database. Acts a wrapper for odbc and DBI packages.

Usage

```
sql_read(server, database, sql)
```

Arguments

`server`	a string containing the server connection
`database`	a string containing the database name within the data store
`sql`	a string containing a SQL query or to a .sql/.txt SQL query

Value

a table from a SQL database

See Also

sql_clean sql_connect

`sql_write` *Write a table to a SQL database*

Description

[Stable]

Write a table object to a SQL database. Acts a wrapper for odbc and DBI packages with additional checks to ensure upload completes.

Usage

```
sql_write(x, server, database, tablename)
```

Arguments

x	a data.frame/data.table/tibble object
server	a string containing the server connection
database	a string containing the database name within the data store
tablename	a string containing the chosen SQL database table name

Value

writes a data.frame/data.table/tibble to a SQL database

---

uk_patient_id *Patient ID record grouping*

Description

[Stable]

Assigns a **single integer** `id` to records that belong to the same patient by applying a sequence of **deterministic matching stages** across common identifiers (NHS number, hospital number, DOB, name, sex, postcode). Identifiers are standardised, validated using NHS checksum function, and fuzzy name keys are used in later stages. ' Matching is performed in order through the following stages (first match is applied):

1. NHS number + date of birth
2. Hospital number + date of birth
3. NHS number + hospital number
4. NHS number + surname
5. Hospital number + surname
6. Date of birth + surname (only where NHS is invalid/absent)
7. Sex + full name (forename + surname)
8. Sex + date of birth + fuzzy name (Soundex; surname + initial)
9. Date of birth (YYYY-MM) + fuzzy name
10. Surname/forename + postcode
11. Name swaps (forename/surname reversed) + date of birth

Use `.useStages` to restrict which stages are applied (default: `1:11`). The function generates a reproducible `id` per patient within the sort order; you can provide `.sortOrder` (e.g., a date column) to make assignment deterministic.

Validity rules applied:

- **NHS number** validated using the standard checksum (`epidm::valid_nhs()`).
- **Hospital number**: excludes known placeholders (e.g., `"UNKNOWN"`, `"NO PATIENT ID"`).
- **DOB**: excludes proxy or missing dates (`"1900-01-01"`, `"1800-01-01"`, `NA`).
- **Sex**: normalised to `"M"` / `"F"`; others → `NA`.
- **Names**: uppercased, Latin characters normalised; Soundex used for fuzzy matching.

Identifiers are copied over where they are missing or invalid to the grouped records.

Usage

```
uk_patient_id(
  x,
 id = list(nhs_number = "nhs_number", hospital_number = "patient_hospital_number",
  date_of_birth = "date_of_birth", sex_mfu = "sex", forename = "forename", surname =
    "surname", postcode = "postcode"),
  .useStages = c(1:11),
  .keepStages = FALSE,
  .keepValidNHS = FALSE,
  .sortOrder,
  .forceCopy = FALSE
)
```

Arguments

x	A `data.frame` or `data.table` with patient identifiers.
id	A **named list** of quoted column names: `nhs_number` NHS number. `hospital_number` Local patient identifier (hospital number). `date_of_birth` Date of birth. `sex_mfu` Sex/gender (M/F/Unknown). `forename` Forename / first name. `surname` Surname / last name. `postcode` Patient postcode.
.useStages	optional, default 1:11; set to 1 if you wish patient ID to be assigned cases with the same DOB and NHS number, set to 2 if you wish patient ID to be assigned to cases with the same hospital number (HOS) and DOB, set to 3 if you wish patient ID to be assigned cases with the same NHS and HOS number, set to 4 if you wish patient ID to be assigned cases with the same NHS number and surname, set to 5 if you wish patient ID to be assigned cases with the same hospital number and surname, set to 6 if you wish patient ID to be assigned cases with the same DOB and surname, set to 7 if you wish patient ID to be assigned cases with the same sex and full name, set to 8 if you wish patient ID to be assigned cases with the same sex, DOB and fuzzy name, set to 9 if you wish patient ID to be assigned cases with the same DOB and fuzzy name, set to 10 if you wish patient ID to be assigned cases with the same name and postcode, set to 11 if you wish patient ID to be assigned cases with the same first name or second name in changing order and date of birth.
.keepStages	optional, default FALSE; to generate a new column (stageMatch) to retain the stage information for which the record matched the group.
.keepValidNHS	optional, default FALSE; set TRUE if you wish to retain the column with the NHS checksum result stored as a BOOLEAN
.sortOrder	optional; a column as a character to allow a sorting order on the id generation
.forceCopy	optional, default FALSE; TRUE will force data.table to take a copy instead of editing the data without reference

Value

A `data.table` with the original columns plus:

`id` Integer patient identifier assigned by staged matching.

`valid_nhs` (Optional) BOOLEAN NHS checksum flag; included when `.keepValidNHS = TRUE`.

Workflow context

`uk_patient_id()` is typically used early to harmonise patient identity across isolates before downstream tasks such as specimen episode grouping (`group_time()`), dataset linkage (e.g., to HES/SUS/ECDS), and epidemiological reporting.

Examples

```
id_test <-
  data.frame(
    stringsAsFactors = FALSE,
    record_id = c(1L,2L,3L,4L,
                  5L,6L,7L,8L,9L,10L,11L,12L,13L,14L,15L,
                  16L,17L,18L,19L,20L,21L,22L,23L,24L),
    nhs_number = c(9435754422,
                   9435754422,NA,9435754422,5555555555,NA,
                   9435773982,NA,9999999999,NA,9435773982,NA,
                   9435802508,9435802508,NA,NA,9435802508,9435802508,NA,
                   3333333333,NA,9999999999,9435817777,
                   9435817777),
    local_patient_identifier = c(NA,"IG12067",
                                 NA,NA,"IG12067","IG12067","KR2535","KR2535",
                                 "KR2535",NA,NA,NA,"UK8734","UK8734",NA,NA,
                                 "UK8734","UK8734",NA,NA,"JH45204",
                                 "HS45202","HS45202","JH45204"),
    patient_birth_date = c("1993-07-16",
                           "1993-07-16","1993-07-16","1993-07-16",
                           "1993-07-16",NA,"1967-02-10",NA,"1967-02-10",NA,NA,
                           "1967-02-10",NA,NA,"1952-10-22","1952-10-22",
                           "1952-10-22",NA,"1947-09-14","1947-09-14",
                           "1947-09-14","1947-09-14","1947-09-14",
                           "1947-09-14"),
    sex = c("Male","Male",
            "Male","Male",NA,"Male","Female","Female",
            "Female","Female","Female","Female","Male",
            "Male","Male","Male","Male","Male","Male",
            "Male","Male","Male",NA,"Male"),
    forename = c(NA,"DENNIS",
                 NA,NA,"DENNIS",NA,"ELLIE","ELLIE",NA,
                 "ELLIE","ELLIE","ELLIE","IAN","IAN","MALCOLM",
                 "IAN","IAN",NA,"GRANT","ALAN","ALAN","ALAN",
                 "GRANT","ALAN"),
    surname = c(NA,"NEDRY",
                "NEDRY",NA,"NEDRY","NEDRY","SATTLER","SATTLER",
                NA,"SATTLER","SATTLER","SATTLER","M",NA,
                "IAN","MALCOLM","MALCOLM",NA,"ALAN","GRANT",
```

```
                "GRANT","GRANT","ALAN","GRANT"),
    postcode = c("HA4 0FF",
                 "HA4 0FF","HA4 0FF",NA,"HA4 0FF","HA4 0FF",
                 "L3 1DZ","L3 1DZ","L3 1DZ","L3 1DZ",NA,"L3 1DZ",
                 "BN14 9EP",NA,"BN14 9EP",NA,NA,NA,"CW6 9TX",
                 "CW6 9TX",NA,NA,NA,NA),
    specimen_date = c("2024-08-14",
                      "2023-02-03","2023-02-07","2023-02-04",
                      "2023-02-09","2024-08-14","2021-03-28","2021-03-28",
                      "2021-03-28","2021-03-28","2021-03-28",
                      "2021-03-28","2024-07-06","2024-07-06","2024-07-06",
                      "2023-10-31","2023-10-31","2023-10-31",
                      "2022-01-23","2022-01-24","2022-01-25","2022-01-26",
                      "2022-01-27","2022-01-28")
  )

data.table::setDT(id_test)

uk_patient_id(
  x = id_test,
  id = list(
    nhs_number = 'nhs_number',
    hospital_number = 'local_patient_identifier',
    date_of_birth = 'patient_birth_date',
    sex_mfu = 'sex',
    forename = 'forename',
    surname = 'surname',
    postcode = 'postcode'
  ),
  .sortOrder = 'specimen_date',
  .useStages = c(1:11),
  .keepStages = TRUE,
  .forceCopy = TRUE)[]
```

`valid_nhs` *NHS Number Validity Check*

Description

[Stable]

Validates NHS numbers using the **NHS checksum** applied to the first 9 digits and compared with the 10th digit. Inputs that are not exactly **10 numeric characters** are invalid. Numbers made of the **same repeated digit** (e.g., `"1111111111"`, `"0000000000"`) are also rejected.

Usage

```
valid_nhs(nhs_number)
```

Arguments

nhs_number	A vector of values to validate. Each element is coerced to character for length checking and digit extraction.

Details

Algorithm (summary):

1. Take the first 9 digits and multiply by weights **10 down to 2** (i.e., d1×10 + d2×9 + … + d9×2).
2. Compute `11 - (sum %% 11)` → this is the **expected check digit**.
3. If the expected check digit is **11**, treat as **0**.
4. Compare with the **10th digit**. If they match, the number is valid.

Additional guards implemented:

- If the NHS number is `NA` or not **10 characters**, it is **invalid**.
- If **all 10 digits are identical** (e.g., `"1111111111"`), it is **invalid**.

The function is vectorised and returns **1 for valid** and **0 for invalid** for each element in the input vector.

Value

A numeric vector of the same length as `nhs_number` containing:

- `1` if the value is a valid NHS number
- `0` otherwise

Examples

```
test <- floor(runif(1000,1000000000,9999999999))
valid_nhs(test)
valid_nhs(9434765919)
```

Index