Package 'vigicaen'

July 22, 2025

Title 'VigiBase' Pharmacovigilance Database Toolbox

Version 0.15.6

Description Perform the analysis of the World Health Organization (WHO) Pharmacovigilance database 'VigiBase' (Extract Case Level version), <https://who-umc.org/>
e.g., load data, perform data management, disproportionality analysis, and descriptive statistics. Intended for pharmacovigilance routine use or studies.
This package is NOT supported nor reflect the opinion of the WHO, or the Uppsala Monitoring Centre.
Disproportionality methods are described by Norén et al (2013) <doi:10.1177/0962280211403604>.

Depends R (>= 4.1.0),

License CeCILL-2.1

Encoding UTF-8

LazyData true

LazyDataCompression xz

RoxygenNote 7.3.2

URL https://github.com/pharmacologie-caen/vigicaen, https://pharmacologie-caen.github.io/vigicaen/

BugReports https://github.com/pharmacologie-caen/vigicaen/issues

Suggests here, knitr, rmarkdown, testthat (>= 3.0.0), tzdb, vdiffr

Config/testthat/edition 3

Imports arrow, cli, dplyr, data.table, fst, ggplot2, glue, gridExtra, lifecycle, purrr, rlang, stringr, tidyr

VignetteBuilder knitr

NeedsCompilation no

Author Charles Dolladille [aut, cre] (ORCID: <https://orcid.org/0000-0003-0449-6261>), Basile Chrétien [aut] (ORCID: <https://orcid.org/0000-0002-7483-2489>),

Universite de Caen Normandie [cph] (Caen, France),
Unite de pharmaco-epidemiologie [cph] (Service de pharmacologie, Centre Hospitalier Universitaire de Caen, Caen, France)

Maintainer Charles Dolladille <cdolladille@hotmail.com>

Repository CRAN

Date/Publication 2025-03-13 15:30:02 UTC

Contents

add_adr *Add ADR column(s) to a dataset*

Description

[Stable]Creates adr columns in vigibase datasets (demo, link, drug, but also adr).

Usage

```
add_adr(
  .data,
  a_code,
  a_names = names(a_code),
  adr_data,
  data_type = deprecated()
)
```

Arguments

.data	The dataset to update (demo, link, drug, adr).
a_code	A named list of low level terms codes (llt_codes).
a_names	A character vector. Names for adr columns (must be the same length as adr_list), default to names(a_code)
adr_data	A data.frame containing the adr data (usually, it is adr)
data_type	**[Deprecated]**. Data_type is now detected internally.

Details

Low-level term codes are the preferred level of requesting in Vigibase extract case level since they capture all possible codes for a given Preferred Term. Collect low-level terms with get_llt_soc() and get_llt_smq(). You can add adr identification to a demo, a link, drug or even an adr dataset (in this latter case, you must provide adr twice, as .data and adr_data). Column names of these dataset should not have been modified from the original vigibase dataset (as created with tb_vigibase()).

Value

A dataset with the new adr columns. Each element of a_names will add a column with the same name in .data. The value can be 0 (the corresponding adr is absent) or 1 (the adr is present in the case if .data is demo or drug, or "this row correspond to this adr", if .data is adr or link).

See Also

add_drug(), get_llt_soc(), get_llt_smq()

Examples

```
# create adr_colitis, adr_embolism and adr_pneumonitis columns in demo

# be careful, this example may overwrite your own demo dataset
demo <- demo_

a_pt_sel <- ex_$pt_sel


adr <- adr_

a_llt <-
  get_llt_soc(
  term_sel = a_pt_sel,
  term_level = "pt",
  meddra = meddra_
  )

demo <-
  demo |>
    add_adr(
      a_code = a_llt,
      adr_data = adr
    )

demo |>
  check_dm(names(a_pt_sel))
```

---

add_drug *Add DRUG column(s) to a dataset (tidyverse syntax)*

Description

[Stable] Creates drug columns. in vigibase datasets (demo, link, adr, but also drug).

Usage

```
add_drug(
  .data,
  d_code,
  d_names = names(d_code),
  repbasis = "sci",
  method = c("DrecNo", "MedicinalProd_Id"),
  drug_data,
  data_type = deprecated()
)
```

Arguments

.data	The dataset used to identify individual reports (usually, it is demo)
d_code	A named list of drug codes (DrecNos or MPI). See Details.
d_names	A character vector. Names for drug columns (must be the same length as d_code), default to names(d_code)
repbasis	Suspect, interacting and/or concomitant. Type initial of those you wish to select ("s" for suspect, "c" for concomitant and "i" for interacting ; default to all, e.g. "sci").
method	A character string. The type of drug code (DrecNo or MedicinalProd_Id). See details.
drug_data	A data.frame containing the drug data (usually, it is drug)
data_type	**[Deprecated]**. Data_type is now detected internally.

Details

d_code is a named list containing drug codes. Either drug record numbers (e.g., from get_drecno()), or medicinalprod_ids (e.g., from get_atc_code()). Default method is to DrecNos.

Value

A dataset with the new drug columns. Each element of d_names will add a column with the same name in .data. The value can be 0 (the corresponding drug is absent) or 1 (the drug is present in the case if .data is demo or adr, or "this row correspond to this drug", if .data is drug or link).

Argument repbasis

Drugs can be reported according to one of three reputation bases:

- s for suspect
- c for concomitant
- i for interacting

in the occurrence of the adverse drug reaction. To study only one of these reputation basis, type only the corresponding letter in repbasis, e.g. "s" for suspects, or "si" for suspect **or** interacting.

You can add drug identification to a demo, link, adr or even drug dataset.(in this latter case, you must provide adr twice, as .data and drug_data)

See Also

add_adr(), get_drecno(), get_atc_code()

Examples

```
# create a nivolumab column in demo_

d_sel_names <- list(nivolumab = "nivolumab")

d_drecno <- get_drecno(d_sel_names,
                       mp = mp_)
demo_ <-
  add_drug(
    .data = demo_,
    d_code = d_drecno,
    method = "DrecNo",
    repbasis = "sci",
    drug_data = drug_
  )

# remember to assign the result to your actual demo dataset

# do you want to work only with cases where nivolumab was a "suspected" drug?
# change argument repbasis to "s"

demo_ <-
  add_drug(
    .data = demo_,
    d_code = d_drecno,
    d_names = "nivolumab_suspected",
    method = "DrecNo",
    repbasis = "s",
    drug_data = drug_
  )

check_dm(demo_, cols = c("nivolumab", "nivolumab_suspected"))
```

---

cff *Fast formatting of numbers*

Description

This is a formatting function for consistent number reporting.

Usage

```
cff(num, low_ci, up_ci, dig = 0, method = c("num_only", "num_ci", "ci"))
```

Arguments

num	A numeric. The number to format.
low_ci	A numeric. Lower end of a confidence interval
up_ci	A numeric. Upper end of a confidence interval

dig	A numeric. Number of digits
method	What sort of printing do you need? (see Details)

Details

Set method according to the printing you like: a unique number with num_only (default), the number and its confidence interval with num_ci, a ci only (for example a range of time to onset) The function properly returns NA when input is missing.

Value

A character vector with the formatted number(s)

Examples

```
num <- c(0.1, 0.02, 1.658)

cff(num)

cff(num, dig = 2)

cff(num = num[[1]],
     low_ci = num[[2]],
     up_ci = num[[3]],
     method = "num_ci",
     dig = 2)
```

---

check_dm *Check binary variables*

Description

[Stable] Quick check that your data management steps through add_adr or add_drug found cases.

Usage

```
check_dm(.data, cols)
```

Arguments

.data	A data.frame to be checked
cols	A character vector, name of columns to look at (usually will be d_names, a_names)

Details

It is a simple wrapper around dplyr::summarise(). Be careful not to supply factors with > 2 levels or continuous outcome (the function does NOT have a checker for this, so that it is faster). Also, the function WONT work with NAs. Use desc_facvar(). if you need more detailed description of your dataset.

Value

A transposed data.frame, with row.names equal to `cols`, and first column is the number of lines in `.data` where each col is equal to 1.

See Also

`desc_facvar()`, `add_adr()`, `add_drug()`

Examples

```
# first create some new variables

demo <- demo_

demo <-
  demo |>
    add_adr(
      a_code = ex_$a_llt,
      adr_data = adr_
    )

 # then check the number of reports with each feature

demo |>
  check_dm(names(ex_$a_llt))
```

`compute_dispro` *Compute disproportionality*

Description

[Stable] Computes bivariate (reporting) Odds-Ratio and Information Component for a drug-adr pair.

Usage

```
compute_dispro(
  .data,
  y,
  x,
  alpha = 0.05,
  na_format = "-",
  dig = 2,
  export_raw_values = FALSE,
  min_n_obs = 0
)
```

Arguments

.data	The data.table to compute from.
y	A character vector, one or more variable to explain (usually an adr).
x	A character vector, one or more explaining variable (usually a drug).
alpha	Alpha risk.
na_format	Character string to fill NA values in ror and ci legends.
dig	Number of digits for rounding (this argument is passed to cff)
export_raw_values	A logical. Should the raw values be exported?
min_n_obs	A numeric, compute disproportionality only for pairs with at least min_n_obs cases.

Details

Significance in pharmacovigilance analysis is only defined if the lower bound of the confidence/credibility interval is above 1 (i.e. low_ci > 1, or ic_tail > 0). Actually, the function computes an Odds-Ratio, which is not necessarily a **reporting** Odds-Ratio.

Value

A data.table, with ROR, IC, and their confidence/credibility interval (at 1 - alpha). Significance of both (as signif_or and signif_ic, if export_raw_values is TRUE).

A data.table with columns

- y and x, same as input
- n_obs the number of observed cases
- n_exp the number of expected cases
- orl the formatted Odds-Ratio
- or_ci the formatted confidence interval
- ic the Information Component
- ic_tail the tail probability of the IC
- ci_level the confidence interval level
- Additional columns, if export_raw_values is TRUE:
- a, b, c, d the counts in the contingency table
- std_er the standard error of the log(OR)
- or the Odds-Ratio
- low_ci the lower bound of the confidence interval
- up_ci the upper bound of the confidence interval
- signif_or the significance of the Odds-Ratio
- signif_ic the significance of the Information Component

See Also

compute_or_mod(), add_drug(), add_adr()

Examples

```
# Say you want to perform a disproportionality analysis between colitis and
# nivolumab among ICI cases

demo <-
  demo_ |>
  add_drug(
    d_code = ex_$d_drecno,
    drug_data = drug_
  ) |>
  add_adr(
    a_code = ex_$a_llt,
    adr_data = adr_
  )

demo |>
  compute_dispro(
    y = "a_colitis",
    x = "nivolumab"
  )

# You don't have to use the pipe syntax, if you're not familiar

compute_dispro(
    .data = demo,
    y = "a_colitis",
    x = "nivolumab"
  )


# Say you want to compute more than one univariate ror at a time.

many_drugs <-
  names(ex_$d_drecno)

demo |>
  compute_dispro(
    y = "a_colitis",
    x = many_drugs
  )


# could do the same with adrs

many_adrs <-
  names(ex_$a_llt)

demo |>
```

```
compute_dispro(
  y = many_adrs,
  x = many_drugs
)

# Export raw values if you want to built plots, or other tables.

demo |>
  compute_dispro(
    y = "a_colitis",
    x = "nivolumab",
    export_raw_values = TRUE
  )

# Set a minimum number of observed cases to compute disproportionality

demo |>
 compute_dispro(
 y = "a_colitis",
 x = "nivolumab",
 min_n_obs = 5
 )
```

`compute_interaction` *Compute interaction disproportionality*

Description

[Experimental] Returns the information component of interaction for a set of 3 variables, usually 2 drugs and an adr.

Usage

```
compute_interaction(
  .data,
  y,
  x,
  z,
  alpha = 0.05,
  na_format = "-",
  dig = 2,
  export_raw_values = FALSE,
  min_n_obs = 0
)
```

Arguments

`.data`	The data.table to compute from.
`y`	A character vector, one or more variable to explain.

x	A character vector, one or more explaining variable.
z	A character vector, one or more explaining variable.
alpha	Alpha risk.
na_format	Character string to fill NA values in ror and ci legends.
dig	Number of digits for rounding (this argument is passed to cff)
export_raw_values	A logical. Should the raw values be exported?
min_n_obs	A numeric, compute disproportionality only for pairs with at least min_n_obs cases.

Details

Significance is similar to usual disproportionality (see compute_dispro()).

Value

A data.table, with Information Component (IC) of interaction, and its credibility interval (at 1 - alpha). Significance as signif_ic, if export_raw_values is TRUE).

A data.table with columns

- y, x and z, same as input
- n_obs the number of observed cases
- n_exp the number of expected cases
- ic the Information Component
- ic_tail the tail probability of the IC
- ci_level the confidence interval level
- Additional columns, if export_raw_values is TRUE:
- a, b, c, d the counts in the contingency table
- signif_ic the significance of the Information Component
- Additional columns, if export_raw_values is TRUE:
- n_* the counts of each setting
- signif_ic the significance of the Information Component

See Also

compute_dispro(), compute_or_mod(), add_drug(), add_adr()

Examples

```
# Interaction on reporting of colitis with ipilimumab and nivolumab
demo <-
  demo_ |>
  add_drug(
    d_code = ex_$d_drecno,
    drug_data = drug_
```

```
  ) |>
  add_adr(
    a_code = ex_$a_llt,
    adr_data = adr_
  )

demo |>
  compute_interaction(
    y = "a_colitis",
    x = "nivolumab",
    z = "ipilimumab"
  )
```

`compute_or_mod` *Compute (r)OR from a model summary*

Description

[Stable] Compute and format Odds-Ratio from a model summary.

Usage

```
compute_or_mod(.coef_table, estimate, std_er, p_val = NULL, alpha = 0.05)
```

Arguments

`.coef_table`	A coefficient table, see details.
`estimate`	Quasiquoted name of estimate parameter.
`std_er`	Quasiquoted name of standard error parameter.
`p_val`	Quasiquoted name of p-value parameter. Optional.
`alpha`	alpha risk.

Details

Helper to compute and format Odds-Ratio based on `summary(glm)$coefficients`, or any equivalent in other modelling packages. (see examples). Preferably, it is transformed into a data.table or data.frame before being evaluated in the function. Otherwise, `compute_or_mod()` will transform it. Significant OR-or column means low_ci is > 1. The `p_val` argument is only required if you wished to display a `nice_p()`.

Output is a data.table. Actually, the function computes an Odds-Ratio, which is not necessarily a *reporting* Odds-Ratio.

Value

A data.table with OR, confidence intervals (at `1 - alpha`), significance (`low_ci > 1`) and (optionally) p-value.

See Also

compute_dispro(), add_drug(), add_adr()

Examples

```
# Reporting Odds-Ratio of colitis with nivolumab among ICI cases.

demo <-
  demo_ |>
  add_drug(
    d_code = ex_$d_drecno,
    drug_data = drug_
  ) |>
  add_adr(
    a_code = ex_$a_llt,
    adr_data = adr_
  )

# Compute the model
mod <- glm(a_colitis ~ nivolumab, data = demo, family = "binomial")

# Extract coefficients
mod_summary <-
 mod |>
 summary()

coef_table <-
 mod_summary$coefficients

# Transform coefficients into ORs with their CI

coef_table |>
  compute_or_mod(
  estimate = Estimate,
  std_er = Std..Error,
  p_val = Pr...z..)

# Also works if you don't have a p_val column
 coef_table |>
  compute_or_mod(
  estimate = Estimate,
  std_er = Std..Error)
```

---

create_example_tables *Example source tables for VigiBase and MedDRA*

Description

[Experimental] Write some example tables as source text/ascii/parquet files.

Usage

```
create_ex_main_txt(path)

create_ex_sub_txt(path)

create_ex_who_txt(path)

create_ex_meddra_asc(path)

create_ex_main_pq(path)
```

Arguments

path	Character string. A folder on your computer where the tables should be written.

Details

VigiBase tables and MedDRA tables are provided respectively as text files and ascii files. The `tb_*` family turns them into parquet files. These `create_example_*` functions are only used to produce example source files to illustrate the `tb_*` family, and parquet files for the same purpose.

Value

A set of text/ascii files, as received by the Uppsala Monitoring Centre or MedDRA

- For `create_ex_main_txt()`, DEMO.txt, DRUG.txt, LINK.txt, FOLLOWUP.txt, ADR.txt, OUT.txt, SRCE.txt, and IND.txt
- For `create_ex_sub_txt()`, AgeGroup_Lx.txt, Dechallenge_Lx.txt, Dechallenge2_Lx.txt, Frequency_Lx.txt, Gender_Lx.txt, Notifier_Lx.txt, Outcome_Lx.txt, Rechallenge_Lx.txt, Rechallenge2_Lx.txt, Region_Lx.txt, RepBasis_Lx.txt, ReportType_Lx.txt, RouteOfAdm_Lx.txt, Seriousness_Lx.txt, and SizeUnit_Lx.txt
- For `create_ex_who_txt()`, ATC.txt, CCODE.txt, ING.txt, MP.txt, ORG.txt, PF.txt, PP.txt, PRT.txt, PRG.txt, SRCE.txt, STR.txt, SUN.txt, ThG.txt, and Unit-X.txt
- For `create_ex_meddra_asc()`, llt.asc, mdhier.asc, smq_content.asc, smq_list.asc
- For `create_ex_main_pq()`, demo.parquet, adr.parquet, drug.parquet, link.parquet, srce.parquet, ind.parquet, out.parquet, followup.parquet, suspdup.parquet

Functions

- `create_ex_sub_txt()`: sub txt tables
- `create_ex_who_txt()`: WHO txt tables
- `create_ex_meddra_asc()`: MedDRA txt tables
- `create_ex_main_pq()`: main parquet tables

See Also

`tb_vigibase()`, `tb_who()`, `tb_meddra()`

Examples

```
path <- paste0(tempdir(), "/crex/")

dir.create(path)

# You may want to use different paths for each type of tables
create_ex_main_txt(path)

create_ex_sub_txt(path)

create_ex_who_txt(path)

create_ex_meddra_asc(path)

create_ex_main_pq(path)

# Remove temporary folders when you're done
unlink(path, recursive = TRUE)
```

---

demo_ *Data of immune checkpoint inhibitors.*

Description

Demo, drug, adr, link, ind, out, srce, and followup are the main table in Vigibase Extract Case Level data. In a regular workflow, you will work with those tables as R objects (e.g. `demo`, `drug`, `adr`, `link`, `ind`, `out`, `srce`, `followup`). These built-in example datasets use an underscore "_" to avoid ambiguity with your own tables (e.g. `demo_`, `drug_`, `adr_`, `link_`, `ind_`, `out_`, `srce_`, `followup_`). This is a relational database, which means every table has a primary key variable (e.g., `UMCReportId` for `demo_`. Keys will allow joints with other tables The full details on the original structure can be found in "VigiBase Extract Case Level - file description.pdf" in your VigiBase ECL folders. `demo_` will typically be your cornerstone table, since it contains one row per report. It is the preferred table to update for drugs and adrs identification before performing disproportionality analyses. These tables are subsets of the original ones, with some of the immune checkpoint inhibitor cases or immune-related adverse events. All data shown in these example data are **FAKE**, which means you shouldn't consider the counts and computations as accurate. Immune checkpoint inhibitors drugs include "Ipilimumab", "Atezolizumab", "Durvalumab", "Nivolumab", "Pembrolizumab", "Avelumab", "Cemiplimab","REGN 2810", "Tremelimumab". More details on how to use vigibase tables can be found in the vignettes. `vignette("basic_workflow")`, `vignette("descriptive")`. To build your own tables, use `tb_vigibase()`. See `vignette("getting_started")`.

Usage

```
data(demo_)

drug_

adr_
```

link_

followup_

ind_

out_

srce_

Format

demo_ is a data.table with 7 variables and 750 rows.

- UMCReportId Integer. The unique identifier of the case report.
- AgeGroup Character. The age group of the patient. Correspondence table is path_sub/AgeGroup.parquet.
- Gender Character. Case gender. path_sub/Gender.parquet
- DateDatabase Character (not date or numeric!). The date of the latest update of the report in the database.
- Type Character. The type of report. path_sub/ReportType.parquet
- Region Character. The world region where the report comes from path_sub/Region.parquet.
- FirstDateDatabase Character. The date the report was first submitted to the database.

drug_ is a data.table with 10 variables and 3514 rows.

- UMCReportId Integer. See demo_.
- Drug_Id Integer. The unique identifier of each drug report.
- MedicinalProd_Id Integer. The medicinalproduct identifier. See get_atc_code().
- DrecNo Integer. Drug Record Number, pivotal to identify drugs with get_drecno().
- Seq1, Seq2 Character. Seq 1 and 2 complement DrecNo, in WHODrug dictionary.
- Route Character. The route of administration of the drug.
- Basis Character. The reputation basis of the drug (suspect, concomitant, or interacting). path_sub/RepBasis.parquet
- Amount Character. The amount of drug administered.
- AmountU Character. The unit of the amount of drug administered. path_sub/SizeUnit.parquet
- Frequency Character. The frequency of drug administration.
- FrequencyU Character. The unit of the frequency of drug administration. path_sub/Frequency.parquet

adr_ is a data.table with 4 variables and 2133 rows.

- UMCReportId Integer. See demo_.
- Adr_Id Integer. The unique identifier of each adverse event report.
- MedDRA_Id Integer. The MedDRA identifier of the adverse event. It is used in get_llt_soc() and get_llt_smq().

- Outcome Character. The outcome of the adverse event. path_sub/Outcome.parquet

link_ is a data.table with 3 variables and 3514 rows. The version built with tb_vigibase() is slightly different than the original one.

- Drug_Id and Adr_Id . Integers. Together, they are the key variable of link. See drug_ and adr_.
- Dechallenge1 and 2 Characters. Dechallenge action and outcome. path_sub/Dechallenge.parquet, path_sub/Dechallenge2.parquet
- Rechallenge1 and 2 Characters. Rechallenge action and outcome. path_sub/Rechallenge.parquet, path_sub/Rechallenge2.parquet
- TimeToOnsetMin and Max Numerics. The minimum and maximum time to onset of the adverse event.
- tto_mean Numeric. The mean time to onset of the adverse event. It is the average of TimeToOnsetMin and Max.
- range Numeric. The incertitude around tto_mean. See vignette("descriptive").
- UMCReportId Integer. See demo_.

ind_ is a data.table with 2 variables and 2426 rows.

- Drug_Id Integer. See drug_.
- Indication Character. The indication of the drug.

out_ is a data.table with 3 variables and 747 rows.

- UMCReportId Integer. See demo_.
- Seriousness Character. The seriousness criteria of the report. path_sub/Seriousness.parquet
- Serious Character. Whether the case is serious or not ("N" No, "Y" Yes)

srce_ is a data.table with 2 variables and 729 rows.

- UMCReportId Integer. See demo_.
- Type Character. The Type of Reporter. path_sub/Notifier.parquet

followup_ is a data.table with 2 variables and 902 rows.

- UMCReportId Integer. See demo_.
- ReplacedUMCReportId Integer. Previous version of the case, which is no longer available in demo_.

An object of class data.table (inherits from data.frame) with 3514 rows and 12 columns.

An object of class data.table (inherits from data.frame) with 2133 rows and 4 columns.

An object of class data.table (inherits from data.frame) with 5136 rows and 11 columns.

An object of class data.table (inherits from data.frame) with 902 rows and 2 columns.

An object of class data.table (inherits from data.frame) with 2426 rows and 2 columns.

An object of class data.table (inherits from data.frame) with 747 rows and 3 columns.

An object of class data.table (inherits from data.frame) with 729 rows and 2 columns.

Source

None

References

There is none

Examples

```
data(demo_)
demo_ |> dplyr::count(AgeGroup)
```

`desc_cont` *Summarize continuous variables*

Description

[Stable] Summarize continuous data and handle output format.

Usage

```
desc_cont(
  .data,
  vc,
  format = "median (q1-q3) [min-max]",
  digits = 1,
  export_raw_values = FALSE
)
```

Arguments

`.data`	A data.frame, where `vc` are column names of continuous variables
`vc`	A character vector, list of column names. Should only contain continuous variables
`format`	A character string. How would you like the output? See details.
`digits`	A numeric. How many digits? This argument calls internal formatting function
`export_raw_values`	A logical. Should the raw values be exported?

Details

Many other packages provide tools to summarize data. This one is just the package author's favorite. This makes it much easier to map to nice labeling thereafter. The `format` argument shows the output of the function. You can change square and round brackets, spaces, separators... Important `format` inputs are

- `median` the median value

- `q1` the first quartile
- `q3` the third quartile
- `min` the minimum value
- `max` the maximum value

The analogous for categorical variables is `desc_facvar()`.

Value

A data.frame with columns

- `var` the variable name
- `level` NA, it is provided to have a consistent output with `desc_facvar()`
- `value` the formatted value with possibly the median, interquartile range, and range (see details)
- `n_avail` the number of cases with available data for this variable.

See Also

`desc_facvar()`

Examples

```
df <-
  data.frame(
    smoke_status = c("smoker", "non-smoker",
           "smoker", "smoker",
           "smoker", "smoker",
           "non-smoker"
           ),
    age = c(60, 50, 56, 49, 75, 69, 85),
    bmi = c(18, 30, 25, 22, 23, 21, 22)
  )

# Use default formatting

desc_cont(.data = df, vc = c("age", "bmi"))

# Use custom formatting

desc_cont(.data = df,
          vc = c("age", "bmi"),
          format = "median (q1;q3)"
          )

# You might want to export raw values, to run plotting or
# other formatting functions

desc_cont(.data = df, vc = c("age", "bmi"),
          export_raw_values = TRUE)
```

desc_dch *Dechallenge descriptive*

Description

[Stable] Computes positive dechallenge counts over a set of adr and drug pairs.

Usage

```
desc_dch(.data, drug_s = "drug1", adr_s = "adr1")
```

Arguments

.data	A link data.table.
drug_s	A character vector, the drug column(s)
adr_s	A character vector, the adverse drug reaction column(s).

Details

Counts are provided at the **case** level (not the drug-adr pair level). Positive dechallenge refers to cases where drug was withdrawn or dose-reduced and reaction abated (in part or in full). You will need a link data.table, see link_, on which you have added drugs and adrs with add_drug() and add_adr().

Value

A data.table with one row per drug-adr pair.

- drug_s and adr_s, same as input
- pos_dch, number of positive dechallenge cases

See Also

link_, add_drug(), add_adr(), desc_tto(), desc_rch()

Examples

```
link_ <-
  link_ |>
  add_drug(
    d_code = ex_$d_groups_drecno,
    drug_data = drug_
  ) |>
  add_adr(
    a_code = ex_$a_llt,
    adr_data = adr_
  )
```

```
desc_dch(link_,
         drug_s = "pd1",
         adr_s = "a_colitis")


# you can vectorize over multiple adrs and drugs

desc_dch(link_,
         drug_s = c("pd1", "pdl1"),
         adr_s = c("a_colitis", "a_pneumonitis"))
```

`desc_facvar` *Summarise categorical variables*

Description

[Stable] Summarize categorical data and handle output format.

Usage

```
desc_facvar(
  .data,
  vf,
  format = "n_/N_ (pc_%)",
  digits = 0,
  pad_width = 12,
  ncat_max = 20,
  export_raw_values = FALSE
)
```

Arguments

`.data`	A data.frame, where `vf` are column names of categorical variables
`vf`	A character vector
`format`	A character string, formatting options.
`digits`	A numeric. Number of digits for the percentage (passed to interval formatting function).
`pad_width`	A numeric. Minimum character length of value output (passed to `stringr::str_pad()`).
`ncat_max`	A numeric. How many levels should be allowed for all variables? See details.
`export_raw_values`	A logical. Should the raw values be exported?

Details

Many other packages provide tools to summarize data. This one is just the package author's favorite. Important `format` inputs are

- `n_` number of patients with the categorical variable at said level
- `N_` the first quartile number of patients with an available value for this variable
- `pc_` percentage of n / N

The format argument should contain at least the words "n_", "N_", and optionally "pc_". `ncat_max` ensures that you didn't provided a continuous variable to `desc_facvar()`. If you have many levels for one of your variables, set to `Inf` or high value. Equivalent for continuous data is `desc_cont()`.

Value

A data.frame with columns

- `var` the variable name
- `level` the level of the variable
- `value` the formatted value with possible number of cases `n_`, number of available cases `N_`, and percentage `pc_`, depending on format argument.
- `n_avail` the number of cases with available data for this variable.

See Also

`desc_cont()`

Examples

```
df1 <-
  data.frame(
    smoke_status = c("smoker", "non-smoker",
           "smoker", "smoker",
           "smoker", "smoker",
           "non-smoker"
           ),
   hypertension = c(1, 1, 0, 1, 1, 1, 1),
    age = c(60, 50, 56, 49, 75, 69, 85),
    bmi = c(18, 30, 25, 22, 23, 21, 22)
  )

# Use default formatting
desc_facvar(.data = df1, vf = c("hypertension", "smoke_status"))

# Use custom formatting
desc_facvar(.data = df1,
           vf = c("hypertension", "smoke_status"),
           format = "n_ out of N_, pc_%",
           digits = 1)

# You might want to export raw values, to run plotting or
```

```
# other formatting functions

desc_facvar(.data = df1,
            vf = c("hypertension", "smoke_status"),
            export_raw_values = TRUE)
```

`desc_outcome` *Outcome descriptive*

Description

[Experimental] Compute outcome description over a set of adr and drugs.

Usage

```
desc_outcome(.data, drug_s = "drug1", adr_s = "adr1")
```

Arguments

`.data`	An `adr` data.table. See `adr_`
`drug_s`	A character vector, the drug column(s)
`adr_s`	A character vector, the adverse drug reaction column(s).

Details

You need an `adr` data.table. Be careful that you cannot directly filter `adr` data.table on drugs! You first have to add drug columns to `adr`, with `add_drug()`. The function reports the worst outcome into consideration for a given case, if many are reported. Outcomes, from best to worst are:

- Recovered/resolved
- Recovering/resolving
- Recovered/resolved with sequelae
- Not recovered/not resolved
- Fatal
- Died- unrelated to reaction
- Died- reaction may be contributory

See `vignette("descriptive")` for more details.

Value

A data.table with one row per drug-adr pair.

- `drug_s` and `adr_s`, same as input
- `n_cas`, number of cases for each category
- `out_label`, the worst outcome for this drug-adr pair

See Also

adr_, add_drug(), add_adr()

Examples

```
adr_ <-
  adr_ |>
  add_drug(
    d_code = ex_$d_groups_drecno,
    drug_data = drug_
  ) |>
  add_adr(
    a_code = ex_$a_llt,
    adr_data = adr_
  )

desc_outcome(
  adr_,
  drug_s = "pd1",
  adr_s = "a_colitis"
  )

# you can vectorize over multiple adrs and drugs

desc_outcome(
  adr_,
  drug_s = c("pd1", "pdl1"),
  adr_s = c("a_colitis", "a_pneumonitis")
  )
```

---

desc_rch *Rechallenge descriptive*

Description

[Stable] Computes counts of rechallenge cases, over a set of adr and drug pairs.

Usage

```
desc_rch(.data, drug_s = "drug1", adr_s = "adr1")
```

Arguments

.data	A link data.table. See link_.
drug_s	A character string. The name of the drug column. Drug columns can be created with add_drug.
adr_s	A character string. The name of the adr column. Adr columns can be created with add_adr.

Details

Counts are provided at the **case** level (not the drug-adr pair level). Description span from number of rechallenge cases to **informative** rechallenge cases (those cases where the outcome is known). You will need a `link` data.table, see `link_`, on which you have added drugs and adrs with `add_drug()` and `add_adr()`. Terminology

- `Overall` as opposed to `rch` for rechallenged (`rch` + `no_rch` = `overall`).
- Among `rch`, `inf` (informative) as opposed to `non_inf` (`inf` + `non_inf` = `rch`)
- Among `inf`, `rec` (recurring) as opposed to `non_rec` (`rec` + `non_rec` = `inf`)

Value

A data.table with one row per drug-adr pair

- `drug_s` and `adr_s`, same as input.
- Counts of **overall**, **rch**, **inf**, and **rec** cases (see details).

See Also

`link_`, `add_drug()`, `add_adr()`, `desc_dch()`, `desc_tto()`

Examples

```
link_ <-
  link_ |>
  add_drug(
    d_code = ex_$d_groups_drecno,
    drug_data = drug_
  ) |>
  add_adr(
    a_code = ex_$a_llt,
    adr_data = adr_
  )

desc_rch(.data = link_,
         drug_s = "pd1",
         adr_s = "a_colitis")

# You can vectorize over drugs and adrs

desc_rch(.data = link_,
         adr_s = c("a_colitis", "a_pneumonitis"),
         drug_s = c("pd1", "pdl1")
         )
```

desc_tto *Time to onset descriptive*

Description

[Stable] `desc_tto()` provides a drug-adr pair description of time to onset.

Usage

```
desc_tto(.data, adr_s, drug_s, tto_time_range = 1, ...)
```

Arguments

`.data`	A `link` data.table. See `link_`.
`adr_s`	A character string. The name of the adr column. (see details)
`drug_s`	A character string. The name of the drug column. (see details)
`tto_time_range`	Incertitude range of Time to onset, in days. Defaults to 1 as recommended by umc
`...`	Additional parameters to be passed to `desc_cont()`. E.g. `format`, `digits`...

Details

Description of time (maximum available time) between drug initiation and event onset. This runs at the drug-adr pair level. Internally, it uses `extract_tto()` and `desc_cont()`, You will need a `link` data.table, see `link_`, on which you have added drugs and adrs with `add_drug()` and `add_adr()`. you can supply extra arguments to `desc_cont()` with `....` Uppsala Monitoring Centre recommends to use only cases where the incertitude on time to onset is less than **1 day**. You can change this with `tto_time_range`.

Value

A data.table with one row per drug-adr pair

- A descriptive of time to onsets for this combination (column `tto_max`).

See Also

`link_`, `extract_tto()`, `add_drug()`, `add_adr()`, `desc_dch()`, `desc_rch()`

Examples

```
link_ <-
  link_ |>
  add_drug(
    d_code = ex_$d_groups_drecno,
    drug_data = drug_
  ) |>
```

```
  add_adr(
    a_code = ex_$a_llt,
    adr_data = adr_
  )

desc_tto(.data = link_,
         adr_s = "a_colitis",
         drug_s = "pd1")


desc_tto(.data = link_,
         adr_s = c("a_colitis", "a_pneumonitis"),
         drug_s = c("pd1", "ctla4"))
```

`dt_fst` *Read fst and convert to data.table*

Description

[Deprecated] Short hand to `as.data.table(read_fst())`. File extension can be omitted.

Usage

```
dt_fst(path_base, name = NULL, ext = ".fst")
```

Arguments

`path_base`	A character string, providing the path to read from.
`name`	A character string, the file name.
`ext`	A character string, optional, specifying the file extension.

Details

Output is a data.table. The function is deprecated, with the use of parquet tables. Tables can now be loaded **IN**-memory or **OUT** of memory with `dt_parquet`.

Value

A data.table, read from `path_base/(name)`.

See Also

`dt_parquet()`, `tb_vigibase()`, `tb_who()`, `tb_meddra()`

Examples

```
# Say you have a data.frame stored in an fst format, such as this one
df <- data.frame(a = 1:10)

path <- paste0(tempdir(), "/dtfstex")
dir.create(path)

fst::write_fst(x = df,
               path = paste0(path, "/", "df.fst")
               )
# Now you have a new session without df.
rm(df)

# You may import the file directly to data.table format with dt_fst
df <- dt_fst(path, "df")

# Clean up (required for CRAN checks)
unlink(path, recursive = TRUE)
```

---

dt_parquet *Read parquet and convert to data.table*

Description

[Stable] Load data IN- our OUT- of memory. File extension can be omitted.

Usage

```
dt_parquet(path_base, name = NULL, ext = ".parquet", in_memory = TRUE)
```

Arguments

path_base	A character string, providing the path to read from.
name	Optional. A character string. The file name (if absent from path_base).
ext	Optional. A character string. The file extension.
in_memory	Logical, should data be loaded in memory?

Details

Output is a data.table. For meddra and whodrug tables, it is still a good option to load data in-memory. This function is wrapping `arrow::read_parquet()`, `dplyr::collect()` and `data.table::as.data.table()` altogether. If you want to load **OUT** of memory, set arg `in_memory` to FALSE. **Be careful that doing so will change the function output format**. For this latter case, the output is not a data.table, so there is no practical benefit as compared to using `arrow::read_parquet()` directly, with `as_data_frame` = FALSE.

Value

A data.table if `in_memory` is set to `TRUE`, a parquet Table if `in_memory` is set to `FALSE`.

See Also

`tb_vigibase()`, `tb_who()`, `tb_meddra()`

Examples

```
# Say you have a data.frame stored in a parquet format, such as this one
demo <-
  data.table::data.table(
    UMCReportId = c(1, 2, 3, 4),
    AgeGroup = c(1, 7, 7, 8)
  ) |>
  arrow::as_arrow_table()

tmp_folder <- paste0(tempdir(), "/dtparquetex")
dir.create(tmp_folder)
path_data <- paste0(tmp_folder, "/")

arrow::write_parquet(demo,
                     sink = paste0(path_data, "demo.parquet")
)

# Now you have a new session without demo
rm(demo)

# You may import the file directly to data.table format with dt_parquet
demo <-
  dt_parquet(path_data, "demo")

# Clean up (required for CRAN checks)
unlink(tmp_folder, recursive = TRUE)
```

---

extract_tto *Time to onset extraction*

Description

[Stable] `extract_tto()` collects all available time to onsets for a set of drug-adr pairs.

Usage

```
extract_tto(.data, adr_s, drug_s, tto_time_range = 1)
```

Arguments

.data	A link data.table. See link_.
adr_s	A character string. The name of the adr column. (see details)
drug_s	A character string. The name of the drug column. (see details)
tto_time_range	Incertitude range of Time to onset, in days. Defaults to 1 as recommended by umc

Details

Extraction of (maximum available) time between drug initiation and event onset. This runs at the drug-adr pair level. You will need a link data.table, see link_, on which you have added drugs and adrs with add_drug() and add_adr(). Uppsala Monitoring Centre recommends to use only cases where the incertitude on time to onset is less than **1 day**. You can change this with tto_time_range. You might want to use desc_tto() to obtain summary statistics of time to onset, but extract_tto() is useful to get the raw data and plot it, for instance with ggplot2.

Value

A data.frame with

- All available time to onsets for this combination (column tto_max).
- adr_s and drug_s, same as input.
- UMCReportId, the unique identifier of the case.

See Also

link_, desc_tto(), add_drug(), add_adr(), desc_dch(), desc_rch()

Examples

```
link_ <-
  link_ |>
  add_drug(
    d_code = ex_$d_groups_drecno,
    drug_data = drug_
  ) |>
  add_adr(
    a_code = ex_$a_llt,
    adr_data = adr_
  )

extract_tto(.data = link_,
         adr_s = "a_colitis",
         drug_s = "pd1")
extract_tto(.data = link_,
         adr_s = c("a_colitis", "a_pneumonitis"),
         drug_s = c("pd1", "ctla4"))
```

ex_ *Data for the immune checkpoint inhibitors example*

Description

These are a set of data to provide examples on the package.

- `smq_sel` is a named list of smq names
- `pt_sel` is a named list of pt names
- `a_llt` is a named list of meddra llt codes related to adrs from `smq_sel` and `pt_sel`
- `d_drecno` is a named list of drecnos for immune checkpoint inhibitors (some of them)
- `d_groups` is a named list of ici classes according to icis
- `d_groups_drecno` is a named list of drecnos for drug groups

Usage

```
data(ex_)
```

Format

An object of class `list`.

Source

VigiBase Extract Case Level

References

There is none

Examples

```
data(ex_)
ex_$pt_sel
```

get_atc_code *Get ATC codes (DrecNos or MPIs)*

Description

[Stable] Collect Drug Record Numbers or MedicinalProd_Ids associated to one or more ATC classes.

Usage

```
get_atc_code(atc_sel, mp, thg_data, vigilyze = TRUE)
```

Arguments

atc_sel	A named list of ATC codes. See Details.
mp	A modified MP data.table. See mp_
thg_data	A data.table. Correspondence between ATC codes and MedicinalProd_Id (usually, it is thg)
vigilyze	A logical. Should ATC classes be retrieved using the vigilyze style? See details

Details

get_atc_code() is an *ID collector* function. Provide atc_sel in the same way as d_sel in add_drug(), but remember to specify its method arg as MedicinalProd_Id if vigilyze is set to FALSE. Vigilyze style means all conditioning of drugs will be retrieved after requesting an ATC class (i.e., drugs are identified with their DrecNos), even if a specific conditioning is not present in the ATC class. This is the default behavior in vigilyze.

Value

A named list of integers. **DrecNos** if vigilyze is set to TRUE, or **MedicinalProd_Ids** if vigilyze is set to FALSE.

See Also

mp_, thg_, add_drug(), get_drecno()

Examples

```
# ## Find codes associated with one or more atc classes

# First, define which atc you want to use

atc_sel <-
     rlang::list2(l03_j01 = c("L03AA", "J01CA"),
                  c09aa = c("C09AA")
     )
```

```
# You can get DrecNos for you ATCs (if vigilyze is TRUE)

atc_drecno <-
  get_atc_code(atc_sel = atc_sel,
               mp = mp_,
               thg_data = thg_,
               vigilyze = TRUE)

# Or you can get MedicinalProd_Ids (if vigilyze is FALSE)

atc_mpi <-
  get_atc_code(atc_sel = atc_sel,
               mp = mp_,
               thg_data = thg_,
               vigilyze = FALSE)
```

`get_drecno` *Get DrecNo from drug names or MedicinalProd_Id*

Description

[Stable] Collect Drug Record Numbers associated to one or more drugs.

Usage

```
get_drecno(
  d_sel,
  mp,
  allow_combination = TRUE,
  method = c("drug_name", "mpi_list"),
  verbose = TRUE,
  show_all = deprecated(),
  inspect = deprecated()
)
```

Arguments

`d_sel`	A named list. Selection of drug names or medicinalprod_id. See details
`mp`	A modified MP data.table. See `mp_`
`allow_combination`	A logical. Should fixed associations including the drug of interest be retrieved? See details.
`method`	Should DrecNo be found from drug names or from MedicinalProd_Id?
`verbose`	A logical. Allows you to see matching drug names in the console. Turn to FALSE once you've checked the matching.
`show_all`	**[Deprecated]** Use `verbose` instead.
`inspect`	**[Deprecated]** Use `verbose` instead.

Details

`get_drecno()` is an *ID collector* function. Collected IDs can be used to create drug columns in datasets like `demo`, `link`, etc. (see `vignette("basic_workflow")`)

Value

A named list of integers. DrecNos.

Argument `verbose`

The `verbose` argument is here to let you check the result of `get_drecno()`. This is an important step in your project setup: You must ensure that the drugs you are looking for are correctly matched.

Argument `d_sel`

`d_sel` must be a named list of character vectors. To learn why, see `vignette("basic_workflow")`. Names of `d_sel` are automatically lowered and trimed.

Matching drugs

With "drug_name" method, either exact match or perl regex match can be used. The latter is built upon lookarounds to ensure that a string does not match to composite drug names including the string, i.e. `trastuzumab emtasine` for `trastuzumab`, or close names like `alitretinoin` when looking for `tretinoin`.

Exact match is used for "mpi_list" method.

Choosing a method

"drug_name" let you work with drug names. It's likely to be the appropriate method in most of the cases.

"mpi_list" is used when you have a list of MedicinalProd_Ids. A drug can have multiple MedicinalProd_Ids, corresponding to different packagings. The MedicinalProd_Id matching is typically used to identify DrecNo(s) contained in an ATC class (extracted from `thg`), since not all MPI of drugs are present in `thg` (explanations in `get_atc_code()`).

WHO names

WHO names are attributed to drugs by... the WHO. A drug only has one WHO name, but can have multiple international nonproprietary names (e.g. "tretinoin" and "all-trans retinoic acid").

You should use WHO names to ensure proper identification of drugs and DrecNos, especially if you work with combinations.

Argument `allow_combination`

Fixed associations of drugs refers to specialty containing more than one active ingredient (for example, acetylsalicylic acid and clopidogrel). In VigiLyze, the default is **NOT** to account for these fixed associations. For example, when you call "acetylsalicylic acid" in VigiLyze, you don't have the cases reported with the fixed-association "acetylsalicylic acid; clopidogrel" **unless the substances**

were distinctly coded by the reporter. Here, the default is to find a drug even if it is prescribed in a fixed association. Importantly, when retrieving fixed-association drugs, the non-of-interest drug alone drecno is not found, hence the cases related to this drug will not be added to those of the drug of interest.

See Also

add_drug(), get_atc_code()

Examples

```
# ## Get drecnos for a list a drugs. Check spelling and use WHO name,
# in lowercase

d_sel_names <- list(
  nivolumab = "nivolumab",
  ipilimumab = "ipilimumab",
  nivo_ipi = c("nivolumab", "ipilimumab")
  )

# Read mp with get_drecno(), to identify drugs without combinations

# Take the time to read the matching drugs. Did you forget a drug?

d_drecno <-
  get_drecno(d_sel_names,
             mp = mp_,
             allow_combination = FALSE,
             method = "drug_name")
d_drecno

# And DrecNos of drugs allowing for combinations

d_drecno <-
  get_drecno(d_sel = d_sel_names,
             mp = mp_,
             allow_combination = TRUE,
             method = "drug_name")
d_drecno
```

---

get_llt_smq *Extract low level terms from SMQs*

Description

[Stable] Collect llts from `smq_list` and `smq_content` data.tables, given an SMQ.

Usage

```
get_llt_smq(
  smq,
  smq_scope = c("narrow", "broad"),
  smq_list,
  smq_content,
  smq_list_content = deprecated()
)
```

Arguments

smq	A named list of character vector of length 1.
smq_scope	A character vector. One of "narrow" or "broad".
smq_list	A data.table. A list of SMQs.
smq_content	A data.table. A list of SMQs content.
smq_list_content	**[Deprecated]**

Details

`get_llt_smq()` is an *ID collector* function. SMQ stands for Standardized MedDRA query. `get_llt_smq()`only works with NON-algorithmic SMQs (this status is given in the smq_list table). See `smq_list_` and `smq_content_`. You can choose between the narrow and the broad scope of the SMQ. If you want to work with the SOC hierarchy, use `get_llt_soc()`.

Value

A named list of integers. Low-level term codes.

See Also

`get_llt_soc()`

Examples

```
## Finding llt codes for Embolism (SMQ)

smq_sel <- rlang::list2(
  embolism = "Embolic and thrombotic events, venous (SMQ)"
 )
get_llt_smq(smq_sel,
                smq_scope = "narrow",
                smq_list = smq_list_,
                smq_content = smq_content_
                )

# You can query multiple SMQs in one item, and query high level SMQs
smq_sel2 <-
  rlang::list2(
```

```
    sepsis = c("Sepsis (SMQ)","Toxic-septic shock conditions (SMQ)"),
    ischemic_heart_disease = c("Ischaemic heart disease (SMQ)"),
  )

get_llt_smq(smq_sel2,
            smq_scope = "narrow",
            smq_list = smq_list_,
            smq_content = smq_content_
            )
```

`get_llt_soc` *Extract low level terms from soc classification*

Description

[Stable] Collect llt codes from a `meddra` data.table, given another term of the MedDRA SOC Hierarchy.

Usage

```
get_llt_soc(
  term_sel,
  term_level = c("soc", "hlgt", "hlt", "pt", "llt"),
  meddra,
  verbose = TRUE
)
```

Arguments

`term_sel`	A named list of character vector(s). The terms to extract llts codes from. See details.
`term_level`	A character string. One of "soc", "hlgt", "hlt", "pt", or "llt"
`meddra`	A data.table. Built from meddra_builders functions
`verbose`	Logical. Allows you to see matching reactions in the console.

Details

`get_llt_soc()` is an *ID collector* function. The function extracts low level term codes. `get_llt_soc()` is **case-sensitive**, and MedDRA terms always begin with a capital letter, in their native version. In `term_sel`, all terms should come from the same hierarchical level, e.g. all preferred terms, all high level terms, etc.

Value

A named list of integers. Low-level term codes.

See Also

get_llt_smq()

Examples

```
## Finding llt codes for colitis

pt_sel <- rlang::list2(
  colitis = c("Colitis",
              "Autoimmune colitis"),
  pneumonitis = c("Pneumonitis",
                  "Organising pneumonia")
  )

hlt_sel <- rlang::list2(
  colitis = c("Gastrointestinal inflammatory disorders NEC"),
  pneumonitis = c("Pulmonary thrombotic and embolic conditions")
  )

# Remember you can use more than one term to define each adverse reaction,
# but they should all be at the same hierarchical level in meddra.

# with preferred terms

get_llt_soc(
  term_sel = pt_sel,
  term_level = "pt",
  meddra = meddra_
  )

# with high level terms

get_llt_soc(
  term_sel = hlt_sel,
  term_level = "hlt",
  meddra = meddra_
  )
```

---

ic_tail *Credibility interval limits for the information component*

Description

[Stable] Compute the Information Component credibility interval, typically the lower end of the 95% CI, also known as the IC025.

Usage

```
ic_tail(n_obs, n_exp, p = 0.025)
```

Arguments

n_obs	Number of observed cases
n_exp	Number of expected cases (see Details)
p	End of chosen credibility interval

Details

The ends of the credibility interval of the information component are estimated with the gamma distribution. n_exp is defined as n_drug * n_event / n_total for the basic IC (formula is different for interactions) Do not add +.5 to n_obs and n_exp as it is automatically done in the function. By default, IC025 is computed. Change p for different ends. It may be easier to use compute_dispro(), which internally calls this function.

Value

A numeric vector. The lower end of the credibility interval

See Also

compute_dispro()

Examples

```
ic_tail(n_obs = 12,
        n_exp = 5)
```

---

meddra_ *Sample of Meddra.*

Description

Anonymized data from MedDRA, used to illustrate the package examples and vignettes. You can find term codes related to colitis, pneumonitis, hepatitis, a SMQ of embolisms. Compounds are meddra_, smq_list_, smq_content_ and smq_list_content_. Create dedicated .parquet files using tb_meddra(). See examples in get_llt_soc and get_llt_smq

Usage

```
data(meddra_)

smq_list_content_

smq_list_

smq_content_
```

Format

`meddra_` is a data.table with 15 variables and 677 rows.

- The `*_code` columns. Integers. MedDRA code for the given term.
- The `*_name` columns. Characters. The name of the term.
- `soc_abbrev` Character. The abbreviation of the SOC.
- `null_field` Logical. Empty column.
- `pt_soc_code` Integer. The preferred term code of the SOC itself.
- `primary_soc_fg` Character. Whether the SOC is primary for this code. "Y" or "N", Yes or No.
- `empty_col` Logical. Empty column.

`smq_list_` is a data.table with 9 variables and 11 rows. It is the list of SMQ.

- `smq_code` Integer. The code of the SMQ.
- `smq_name` Character. The name of the SMQ.
- `smq_level` Integer. The hierarchical level of the SMQ.
- `smq_description` Character. The description of the SMQ.
- `smq_source` Character. The source of the SMQ.
- `smq_note` Character. Additional note on the SMQ.
- `MedDRA_version` Numeric. The version of MedDRA.
- `status` Character. The status of the SMQ (active or not)
- `smq_algorithm` Character. Whether the SMQ is algorithmic or not.
- `empty_col` Logical. Empty column.

`smq_content_` is a data.table with 9 variables and 3386 rows. It is the content of each SMQ.

- `smq_code` Integer. The code of the SMQ.
- `term_code` Integer. The low-level term code.
- `term_level` Integer. The hierarchical level of the term.
- `term_scope` Integer. The scope of the term (narrow 2 or broad 1)
- `term_category` Character. In algorithmic SMQs, the category of the term.
- `term_weight` Integer. The weight of the term (algorithmic SMQs).

- term_status Integer. The status of the term (active or not)
- term_addition_version Numeric. The version of the term addition.
- term_last_modified_version Numeric. The last MedDRA version the term was modified.
- empty_col Logical. Empty column.

smq_list_content_ is a data.table with 19 variables and 3386 rows. It is a fusion of smq_list and smq_content, as created with tb_meddra().

- smq_code Integer. The code of the SMQ.
- smq_name Character. The name of the SMQ.
- smq_level Integer. The hierarchical level of the SMQ.
- smq_description Character. The description of the SMQ.
- smq_source Character. The source of the SMQ.
- smq_note Character. Additional note on the SMQ.
- MedDRA_version Numeric. The version of MedDRA.
- status Character. The status of the SMQ (active or not)
- smq_algorithm Character. Whether the SMQ is algorithmic or not.
- empty_col.x Logical. Empty column.
- term_code Integer. The low-level term code.
- term_level Integer. The hierarchical level of the term.
- term_scope Integer. The scope of the term (narrow 2 or broad 1)
- term_category Character. In algorithmic SMQs, the category of the term.
- term_weight Integer. The weight of the term (algorithmic SMQs).
- term_status Integer. The status of the term (active or not)
- term_addition_version Numeric. The version of the term addition.
- term_last_modified_version Numeric. The last MedDRA version the term was modified.
- empty_col.y Logical. Empty column.

An object of class data.table (inherits from data.frame) with 3386 rows and 19 columns.

An object of class data.table (inherits from data.frame) with 11 rows and 9 columns.

An object of class data.table (inherits from data.frame) with 3386 rows and 9 columns.

Source

None

References

There is none

Examples

```
data(meddra_)
```

mp_ *Sample of WHODrug*

Description

A small part of WHODrug, used to illustrate the package examples and vignettes. You can find DrecNo related to immune checkpoint inhibitors, paracetamol, tramadol, tretinoin, anti-thrombin iii, and ATC classes L03AA Colony stimulating factors, C09AA ACE inhibitors, plain, J01CA Penicillins with extended spectrum. Compounds are `thg_` and `mp_`. See examples in `get_drecno` and `get_atc_code`

Usage

```
data(mp_)

thg_
```

Format

`mp_` is a data.table with 8 variables and 14146 rows.

- `MedicinalProd_Id` Integer. The medicinalproduct identifier.
- `Sequence.number.1` and `2` Characters. Complement to DrecNo.
- `DrecNo` Character. Drug Record Number, pivotal to identify drugs with `get_drecno()`.
- `drug_name_t` Character. The name of the drug. Compared to the original `drug_name` variable in `mp` table, this variable is trimmed for white spaces, and names are in lowercase.
- `Create.date` Character. The date the record was created.
- `Date.changed` Character. The date the record was last changed.
- `Country` Character. The country where the record was created.

`thg_` is a data.table with 5 variables and 4079 rows.

- `Therapgroup_Id` Integer. The identifier of the therapeutic group.
- `ATC.code` Character. The ATC code of the drug.
- `Create.date` Character. The date the record was created.
- `Official.ATC.code` Character. Whether the ATC code is official (Yes/No).
- `MedicinalProd_Id` Integer. The medicinalproduct identifier.

An object of class `data.table` (inherits from `data.frame`) with 4079 rows and 5 columns.

Source

None

References

There is none

Examples

```
data(mp_)
```

`nice_p` *Nice printing of p-values*

Description

[Stable] Formatting function for consistent p-value reporting.

You can choose to print the leading zero (e.g. `0.01`) or not (e.g. `.01`) with `print_zero`.

Usage

```
nice_p(p_val, print_zero = FALSE)
```

Arguments

`p_val`	A numeric. The p-value to format.
`print_zero`	A logical. Should leading zero be printed? (see Details)

Value

A character vector with the formatted p-value(s)

Examples

```
pvals <-
  c(0.056548, 0.0002654, 0.816546, 0.0493321)
nice_p(pvals)

nice_p(pvals, print_zero = TRUE)
```

screen_adr *Screening of Adverse Drug Reactions*

Description

[Experimental] Identify and rank the most frequently reported adverse drug reaction (ADR) terms in a dataset, based on a specified MedDRA term level. It allows users to filter terms by a frequency threshold or extract the top n most frequently occurring terms.

Arguments

.data	An adr data.table. See adr_
meddra	A meddra data.table. See meddra_
term_level	A character string specifying the MedDRA hierarchy level. Must be one of "soc", "hlgt", "hlt", "pt", or "llt".
freq_threshold	A numeric value indicating the minimum frequency (as a proportion) of cases where a term must appear to be included in the results. For example, 0.05 means 5%. Defaults to NULL, meaning no threshold is applied unless top_n is different from NULL.
top_n	An integer specifying the number of most frequently occurring terms to return. Defaults to NULL. Overrides freq_threshold if both are provided.

Details

- If freq_threshold is set (e.g., 0.05), the function filters ADR terms appearing in at least 5% of unique reports in .data.
- If top_n is specified, only the most frequent n terms are returned. If both freq_threshold and top_n are provided, only top_n is applied (a warning is issued in such cases).
- Counts are computed at the *case* level, not the ADR level. This means frequencies reflect the proportion of unique reports (cases) where a term is mentioned, rather than the total mentions across all reports.

The function processes an ADR dataset (adr_) and a MedDRA dataset (meddra_) to generate results that are linked to a specific MedDRA hierarchy level (soc, hlgt, hlt, pt, or llt).

Value

A data.frame with the following columns:

- **term**: The MedDRA term at the specified hierarchy level.
- **n**: The number of unique reports (cases) where the term appears.
- **percentage**: The percentage of total unique reports where the term appears.

The results are sorted in descending order of percentage.

Examples

```
# Example 1: Filter terms appearing in at least 5% of reports
screen_adr(
  .data = adr_,
  meddra = meddra_,
  term_level = "pt",
  freq_threshold = 0.05
)

# Example 2: Get the top 5 most frequent terms
screen_adr(
  .data = adr_,
  meddra = meddra_,
  term_level = "hlt",
  top_n = 5
)
```

---

screen_drug *Screening of Drugs*

Description

[Experimental] The `screen_drug()` function identifies and ranks the most frequently reported drugs (by active ingredient) in a dataset.

Usage

```
screen_drug(.data, mp_data, freq_threshold = NULL, top_n = NULL)
```

Arguments

`.data`	An drug data.table. See drug_
`mp_data`	An MP data.table. See mp_
`freq_threshold`	A numeric value indicating the minimum frequency (as a proportion) of cases where a drug must appear to be included in the results. Defaults to `NULL`.
`top_n`	An integer specifying the number of most frequently occurring drugs to return. Defaults to `NULL`.

Details

- If `freq_threshold` is set (e.g., `0.05`), the function filters drugs appearing in at least 5% of unique reports in `.data`.
- If `top_n` is specified, only the most frequent `n` drugs are returned. If both `freq_threshold` and `top_n` are provided, only `top_n` is applied (a warning is raised in such cases).
- Counts are computed at the *case* level, not the drug mention level. This means frequencies reflect the proportion of unique reports (cases) where a drug is mentioned, rather than the total mentions across all reports.

Value

A `data.frame` with the following columns:

- `Drug name`: The drug name.
- `DrecNo`: The drug record number
- `N`: The number of unique reports (cases) where the drug appears.
- `percentage`: The percentage of total unique reports where the drug appears.

The results are sorted in descending order of `percentage`.

Examples

```
# Set up start
data.table::setDTthreads(1)

# Filter drugs appearing in at least 10% of reports
screen_drug(
  .data = drug_,
  mp_data = mp_,
  freq_threshold = 0.10
)

# Get the top 5 most reported drugs
screen_drug(
  .data = drug_,
  mp_data = mp_,
  top_n = 5
)

# nb: in the example datasets, not all drugs are recorded in mp_,
# leading to NAs in screen_drug output.

# Set up end
data.table::setDTthreads(0)
```

---

tb_meddra *Create MedDRA tables*

Description

[Stable] Transform MedDRA .ascii files to .parquet files

MedDRA is delivered as ascii files, that you should transform to a more efficient format. Parquet format from arrow has many advantages: It works with out-of-memory data, which makes it possible to process tables on a computer with not-so-much RAM. It is also lightweighted and standard across different langages. The function also creates variables in each table. You should note that NOT all MedDRA tables are processed with this function. Three tables are created: `meddra_hierarchy`, that respects the System Organ Class hierarchic classification. `smq_list` and `smq_content` for Standardized MedDRA Queries. **Caution** There tends to be small variations in the MedDRA ascii files structure. Last verified version on which this function is working is **26.1**. Use `dt_parquet()` to load the tables afterward.

Usage

```
tb_meddra(path_meddra)
```

Arguments

path_meddra	Character string, a directory containing MedDRA ascii tables. It is also the output directory.

Value

.parquet files into the path_meddra directory. Three tables: meddra_hierarchy, smq_list, and smq_content. Some columns are returned as integer (all *_code columns). All other columns are character.

See Also

tb_vigibase(), tb_who(), tb_subset(), dt_parquet()

Examples

```
# Use the examples from tb_main if you want to see these functions in action.

path_meddra <- paste0(tempdir(), "/meddra_directory/")
dir.create(path_meddra)
create_ex_meddra_asc(path_meddra)

tb_meddra(path_meddra = path_meddra)

# Clear temporary files when you're done
unlink(path_meddra, recursive = TRUE)
```

---

tb_subset *Extract of subset of Vigibase*

Description

[Stable] Create a subset of the VigiBase ECL datasets

Usage

```
tb_subset(
  wd_in,
  wd_out,
  subset_var = c("drecno", "medprod_id", "meddra_id", "age"),
  sv_selection,
  rm_suspdup = TRUE
)
```

Arguments

wd_in	Source directory pathway (character)
wd_out	Output directory pathway (character)
subset_var	One of "drecno", "medprod_id", "meddra_id", "age"
sv_selection	A named list or a vector containing the appropriate ids (according to the method, see details)
rm_suspdup	A logical. Should suspected duplicates be removed? TRUE by default

Details

You must select a subset variable with subset_var and provide an appropriate list according to this variable in sv_selection. Available subset_var :

- drecno will use Drug Record Number (DrecNo), from WHO Drug, and will subset from drug (see get_drecno()).
- medprod_id will use MedicinalProd_Id, also from drug. May be useful if requesting from ATC classes. (see get_atc_code()).
- meddra_id will use MedDRA_Id, subset from adr. (see get_llt_soc() or See get_llt_smq()).
- age will use AgeGroup from demo. See below.

Age groups ids are as follows:

- 1 0 - 27 days
- 2 28 days to 23 months
- 3 2 - 11 years
- 4 12 - 17 years
- 5 18 - 44 years
- 6 45 - 64 years
- 7 65 - 74 years
- 8 >= 75 years
- 9 Unknown

Example: To work with patients aged 18 to 74, provide c(5, 6, 7) as sv_selection.

Use dt_parquet() to load the tables afterward.

Value

Parquet files in the output directory. All files from a vigibase ECL main folder are returned subsetted (including suspectedduplicates).

See Also

get_drecno(), get_atc_code(), get_llt_soc(), get_llt_smq(), dt_parquet()

Examples

```
# --- technical steps ---- #

wd_in <- paste0(tempdir(), "/", "tbsubsetex")
dir.create(wd_in)
create_ex_main_pq(wd_in)

# Select a subset_var and corresponding data

# Subset on adr colitis codes

adr_llt <-
 list(
   colitis = "Colitis"
   ) |>
   get_llt_soc(term_level = "pt", meddra_, verbose = FALSE)

wd_out <- paste0(wd_in, "/", "colitis_subset", "/")

tb_subset(wd_in, wd_out,
          subset_var = "meddra_id",
          sv_selection = adr_llt)

# Subset on drug codes

 d_drecno <-
   list(
    ipi = "ipilimumab") |>
    get_drecno(mp = mp_, verbose = FALSE)

wd_out <- paste0(wd_in, "/", "nivolumab_subset", "/")

tb_subset(wd_in, wd_out,
          subset_var = "drecno",
          sv_selection = d_drecno)

 # Subset on age > 65 year-old

 sv_selection <-
    c(7, 8)

wd_out <- paste0(wd_in, "/", "more_than_65_subset", "/")

tb_subset(wd_in, wd_out,
          subset_var = "age",
          sv_selection = sv_selection)

unlink(wd_in, recursive = TRUE)
```

tb_vigibase *Create main VigiBase ECL tables*

Description

[Stable] Transform VigiBase .txt files to .parquet files.

Usage

```
tb_vigibase(path_base, path_sub, force = FALSE)
```

Arguments

path_base	Character string, a directory containing vigibase txt tables. It is also the output directory.
path_sub	Character string, a directory containing subsidiary tables.
force	Logical, to be passed to `cli::cli_progress_update()`. Used for internal purposes.

Details

Vigibase Extract Case Level is delivered as zipped text files, that you should transform to a more efficient format. Parquet format from `arrow` has many advantages: It works with out-of-memory data, which makes it possible to process Vigibase tables on a computer with not-so-much RAM. It is also lightweighted and standard across different langages. The function also creates variables in each table. The `suspectedduplicates` table will be added to the base directory. Use `dt_parquet()` to load the tables afterward.

Value

- .parquet files of all main tables into the `path_base` directory: demo, adr, drug, link, ind, out, srce, followup, and the suspdup (suspected duplicates) table. Check `?demo_` for more information on the tables.
- The link table is augmented with `tto_mean` and `range`, to analyze time to onset according to WHo's recommendations (see `vignette("descriptive")`.
- .parquet files of all other subsidiary tables into the `path_sub` directory: AgeGroup, Dechallenge, Dechallenge2, Frequency, Gender, Notifier, Outcome, Rechallenge, Rechallenge2, Region, RepBasis, ReportType, RouteOfAdm, Seriousness, and SizeUnit.

.parquet files into the `path_base` directory (**including suspected duplicates tables**). Some columns are returned as `integer` (UMCReportId, Drug_Id, MedicinalProd_Id, Adr_Id, MedDRA_Id), and some columns as `numeric` (TimeToOnsetMin, TimeToOnsetMax) All other columns are `character`.

See Also

`tb_who()`, `tb_meddra()`, `tb_subset()`, `dt_parquet()`

Examples

```
# --- Set up example source files ---- ####

path_base <- paste0(tempdir(), "/", "main", "/")

path_sub  <- paste0(tempdir(), "/", "sub",  "/")

dir.create(path_base)
dir.create(path_sub)

create_ex_main_txt(path_base)
create_ex_sub_txt(path_sub)

 # ---- Running tb_vigibase

 tb_vigibase(path_base = path_base,
        path_sub  = path_sub)

 # Clear temporary files when you're done
 unlink(path_base, recursive = TRUE)
 unlink(path_sub, recursive = TRUE)
```

---

tb_who *Create main WHO tables*

Description

[Stable] Transform Vigibase WHO .txt files to .parquet files

WHODrug is delivered as zipped text files folder, that you should transform to a more efficient format. Parquet format from arrow has many advantages: It can work with out-of-memory data, which makes it possible to process tables on a computer with not-so-much RAM. It is also lightweighted and standard across different languages. The function also creates variables in each table. See `tb_vigibase()` for some running examples, and try `?mp_` or `?thg_` for more details. Use `dt_parquet()` to load the tables afterward.

Usage

```
tb_who(path_who, force = FALSE)
```

Arguments

`path_who`	Character string, a directory containing whodrug txt tables. It is also the output directory.
`force`	Logical, to be passed to `cli::cli_progress_update()`. Used for internal purposes.

Value

.parquet files into the path_who directory. Some columns are returned as integer (all Id columns, including MedicinalProd_Id, with notable exception of DrecNo), and some columns as numeric (Quantity from ingredient table) All other columns are character.

See Also

tb_vigibase(), tb_meddra(), tb_subset(), dt_parquet()

Examples

```
# Use the examples from tb_main if you want to see these functions in action.

path_who <- paste0(tempdir(), "/whodrug_directory/")
dir.create(path_who)
create_ex_who_txt(path_who)

tb_who(path_who = path_who)

# Clear temporary files when you're done
unlink(path_who, recursive = TRUE)
```

---

vigi_routine *Pharmacovigilance routine function*

Description

[Experimental] vigi_routine() draws an Information Component plot and a Time to Onset plot for a given drug-adr pair.

Usage

```
vigi_routine(
  demo_data,
  drug_data,
  adr_data,
  link_data,
  d_code,
  a_code,
  case_tto = NULL,
  vigibase_version,
  analysis_setting = "All reports",
  d_label = NULL,
  a_label = NULL,
  export_to = NULL
)
```

Arguments

demo_data	A demo data.table.
drug_data	A drug data.table.
adr_data	An adr data.table.
link_data	A link data.table.
d_code	A named list. The drug code(s) to be used. There must be only one item in d_code.
a_code	A named list. The adr code(s) to be used. There must be only one item in a_code.
case_tto	A numeric. The time to onset of the studied case. See details.
vigibase_version	A character. The version of VigiBase used (e.g. "September 2024"). This is passed to the plot legend.
analysis_setting	A character. The setting of the analysis. See details.
d_label	A character. The name of the drug, as passed to the plot legend. Defaults to names(d_code).
a_label	A character. The name of the adr, as passed to the plot legend. Defaults to names(a_code).
export_to	A character. The path to export the plot. If NULL, the plot is not exported. Should end by ".eps", ".ps", ".tex" (pictex), ".pdf", ".jpeg", ".tiff", ".png", ".bmp", ".svg" or ".wmf" (windows only).

Details

See `vignette("routine_pharmacovigilance")` for examples. The output can be exported. Time to onset data are bounded between 1 day and 10 years. Data outside this range are reassigned a 1 day and 10 years value, respectively. The function only works if there is one item in `d_code` and `a_code`. If you are working on a specific case, you can provide a `case_tto` value. This value will be displayed on the Time to Onset plot. If you're demo table was filtered on specific cases (e.g. older adults, a subset of all drugs), then you may want to indicate this setting on the plot legend, with arg `analysis_setting`.

Value

A ggplot2 graph, with two panels. The first panel, on top, is the Information Component (IC) plot. The arrow and "IC025 label" indicate the IC value for the selected drug-adr pair. The second panel, on the bottom, is the Time to Onset (TTO) density plot. It is derived only of cases where the drug was **suspected** to be responsible of the adr. If you provide a case_tto value, it is represented by the red line, and the label.

Examples

```
# Say you want to perform a disproportionality analysis between colitis and
# nivolumab among ICI cases
```

```
# identify drug DrecNo, and adr LLT code

d_drecno <-
  ex_$d_drecno["nivolumab"]

a_llt <-
  ex_$a_llt["a_colitis"]

# But you could also use get_drecno() and get_llt_soc()

# load tables demo, drug, adr, and link

demo <- demo_
adr  <- adr_
drug <- drug_
link <- link_

# run routine

vigi_routine(
  demo_data = demo,
  drug_data = drug,
  adr_data  = adr,
  link_data = link,
  d_code = d_drecno,
  a_code = a_llt,
  vigibase_version = "September 2024"
)

# if you're working on a case, you can provide his/her time to onset
# with arg `case_tto`

vigi_routine(
  case_tto = 150,
  demo_data = demo,
  drug_data = drug,
  adr_data  = adr,
  link_data = link,
  d_code = d_drecno,
  a_code = a_llt,
  vigibase_version = "September 2024"
)


# Customize with d_name and a_name, export the plot with export_to

vigi_routine(
  case_tto = 150,
  demo_data = demo,
  drug_data = drug,
  adr_data  = adr,
  link_data = link,
  d_code = d_drecno,
```

```
  a_code = a_llt,
  vigibase_version = "September 2024",
  d_label = "Nivolumab",
  a_label = "Colitis",
  export_to = paste0(tempdir(), "/", "vigicaen_graph.png")
)
```

Index