Package 'AMR'

June 23, 2019

Version 0.7.1

Date 2019-06-23

Title Antimicrobial Resistance Analysis

Description Functions to simplify the analysis and prediction of Antimicrobial Resistance (AMR) and to work with microbial and antimicrobial properties by using evidence-based methods.

Depends R (>= 3.1.0)

Imports backports, crayon (>= 1.3.0), data.table (>= 1.9.0), dplyr (>= 0.7.0), ggplot2, hms, knitr (>= 1.0.0), microbenchmark, rlang (>= 0.3.1), scales, tidyr (>= 0.7.0)

Suggests covr (>= 3.0.1), curl, readxl, rmarkdown, rstudioapi, rvest (>= 0.3.2), testthat (>= 1.0.2), xml2 (>= 1.0.0)

VignetteBuilder knitr

URL https://msberends.gitlab.io/AMR, https://gitlab.com/msberends/AMR

BugReports https://gitlab.com/msberends/AMR/issues

License GPL-2 | file LICENSE

Encoding UTF-8

LazyData true

RoxygenNote 6.1.1

NeedsCompilation no

Author Matthijs S. Berends [aut, cre] (<https://orcid.org/0000-0001-7620-1800>), Christian F. Luz [aut] (<https://orcid.org/0000-0001-5809-5995>), Erwin E. A. Hassing [ctb], Corinna Glasner [aut, ths] (<https://orcid.org/0000-0003-1241-1328>), Alex W. Friedrich [aut, ths] (<https://orcid.org/0000-0003-4881-038X>), Bhanu N. M. Sinha [aut, ths] (<https://orcid.org/0000-0003-1634-0010>)

Maintainer Matthijs S. Berends <m.s.berends@umcg.nl>

Repository CRAN

Date/Publication 2019-06-23 09:20:02 UTC

R topics documented:

ab_property *Property of an antibiotic*

Description

Use these functions to return a specific property of an antibiotic from the antibiotics data set. All input values will be evaluated internally with as.ab.

Usage

```
ab_name(x, language = get_locale(), tolower = FALSE, ...)

ab_atc(x, ...)

ab_cid(x, ...)

ab_synonyms(x, ...)

ab_tradenames(x, ...)

ab_group(x, language = get_locale(), ...)

ab_atc_group1(x, language = get_locale(), ...)

ab_atc_group2(x, language = get_locale(), ...)

ab_ddd(x, administration = "oral", units = FALSE, ...)

ab_info(x, language = get_locale(), ...)

ab_property(x, property = "name", language = get_locale(), ...)
```

Arguments

x	any (vector of) text that can be coerced to a valid microorganism code with as.ab
language	language of the returned text, defaults to system language (see get_locale) and can also be set with getOption("AMR_locale"). Use language = NULL or language = "" to prevent translation.
tolower	logical to indicate whether the first character of every output should be transformed to a lower case character. This will lead to e.g. "polymyxin B" and not "polymyxin b".
...	other parameters passed on to as.ab
administration	way of administration, either "oral" or "iv"
units	a logical to indicate whether the units instead of the DDDs itself must be returned, see Examples

property one of the column names of one of the antibiotics data set

Details

All output will be translated where possible.

Value

- An integer in case of ab_cid
- A named list in case of ab_info and multiple ab_synonyms/ab_tradenames
- A double in case of ab_ddd
- A character in all other cases

Source

World Health Organization (WHO) Collaborating Centre for Drug Statistics Methodology: https://www.whocc.no/atc_ddd_index/

WHONET 2019 software: http://www.whonet.org/software.html

European Commission Public Health PHARMACEUTICALS - COMMUNITY REGISTER: http://ec.europa.eu/health/documents/community-register/html/atc.htm

Read more on our website!

On our website https://msberends.gitlab.io/AMR you can find a tutorial about how to conduct AMR analysis, the complete documentation of all functions (which reads a lot easier than here in R) and an example analysis using WHONET data.

See Also

antibiotics

Examples

```
# all properties:
ab_name("AMX")       # "Amoxicillin"
ab_atc("AMX")        # J01CA04 (ATC code from the WHO)
ab_cid("AMX")        # 33613 (Compound ID from PubChem)

ab_synonyms("AMX")   # a list with brand names of amoxicillin
ab_tradenames("AMX") # same

ab_group("AMX")      # "Beta-lactams/penicillins"
ab_atc_group1("AMX") # "Beta-lactam antibacterials, penicillins"
ab_atc_group2("AMX") # "Penicillins with extended spectrum"

ab_name(x = c("AMC", "PLB"))  # "Amoxicillin/clavulanic acid" "Polymyxin B"
ab_name(x = c("AMC", "PLB"),
        tolower = TRUE)       # "amoxicillin/clavulanic acid" "polymyxin B"

ab_ddd("AMX", "oral")              #  1
```

```
ab_ddd("AMX", "oral", units = TRUE) # "g"
ab_ddd("AMX", "iv")                 #  1
ab_ddd("AMX", "iv", units = TRUE)   # "g"

ab_info("AMX")       # all properties as a list

# all ab_* functions use as.ab() internally:
ab_name("Fluclox")   # "Flucloxacillin"
ab_name("fluklox")   # "Flucloxacillin"
ab_name("floxapen")  # "Flucloxacillin"
ab_name(21319)       # "Flucloxacillin" (using CID)
ab_name("J01CF05")   # "Flucloxacillin" (using ATC)
```

---

age *Age in years of individuals*

Description

Calculates age in years based on a reference date, which is the sytem date at default.

Usage

```
age(x, reference = Sys.Date(), exact = FALSE)
```

Arguments

x	date(s), will be coerced with as.POSIXlt
reference	reference date(s) (defaults to today), will be coerced with as.POSIXlt and cannot be lower than x
exact	a logical to indicate whether age calculation should be exact, i.e. with decimals. It divides the number of days of year-to-date (YTD) of x by the number of days in a year of reference (either 365 or 366).

Value

An integer (no decimals) if exact = FALSE, a double (with decimals) otherwise

Read more on our website!

On our website https://msberends.gitlab.io/AMR you can find a tutorial about how to conduct AMR analysis, the complete documentation of all functions (which reads a lot easier than here in R) and an example analysis using WHONET data.

See Also

age_groups to split age into age groups

Examples

```
# 10 random birth dates
df <- data.frame(birth_date = Sys.Date() - runif(10) * 25000)
# add ages
df$age <- age(df$birth_date)
# add exact ages
df$age_exact <- age(df$birth_date, exact = TRUE)

df
```

`age_groups` *Split ages into age groups*

Description

Split ages into age groups defined by the `split` parameter. This allows for easier demographic (antimicrobial resistance) analysis.

Usage

```
age_groups(x, split_at = c(12, 25, 55, 75))
```

Arguments

`x`	age, e.g. calculated with `age`
`split_at`	values to split `x` at, defaults to age groups 0-11, 12-24, 25-54, 55-74 and 75+. See Details.

Details

To split ages, the input can be:

- A numeric vector. A vector of e.g. `c(10, 20)` will split on 0-9, 10-19 and 20+. A value of only `50` will split on 0-49 and 50+. The default is to split on young children (0-11), youth (12-24), young adults (25-54), middle-aged adults (55-74) and elderly (75+).
- A character:
  - `"children"` or `"kids"`, equivalent of: `c(0, 1, 2, 4, 6, 13, 18)`. This will split on 0, 1, 2-3, 4-5, 6-12, 13-17 and 18+.
  - `"elderly"` or `"seniors"`, equivalent of: `c(65, 75, 85)`. This will split on 0-64, 65-74, 75-84, 85+.
  - `"fives"`, equivalent of: `1:20 * 5`. This will split on 0-4, 5-9, 10-14, ..., 90-94, 95-99, 100+.
  - `"tens"`, equivalent of: `1:10 * 10`. This will split on 0-9, 10-19, 20-29, ... 80-89, 90-99, 100+.

Value

Ordered factor

Read more on our website!

On our website https://msberends.gitlab.io/AMR you can find a tutorial about how to conduct AMR analysis, the complete documentation of all functions (which reads a lot easier than here in R) and an example analysis using WHONET data.

See Also

age to determine ages based on one or more reference dates

Examples

```
ages <- c(3, 8, 16, 54, 31, 76, 101, 43, 21)

# split into 0-49 and 50+
age_groups(ages, 50)

# split into 0-19, 20-49 and 50+
age_groups(ages, c(20, 50))

# split into groups of ten years
age_groups(ages, 1:10 * 10)
age_groups(ages, split_at = "tens")

# split into groups of five years
age_groups(ages, 1:20 * 5)
age_groups(ages, split_at = "fives")

# split specifically for children
age_groups(ages, "children")
# same:
age_groups(ages, c(1, 2, 4, 6, 13, 17))

# resistance of ciprofloxacine per age group
library(dplyr)
septic_patients %>%
  filter_first_isolate() %>%
  filter(mo == as.mo("E. coli")) %>%
  group_by(age_group = age_groups(age)) %>%
  select(age_group, CIP) %>%
  ggplot_rsi(x = "age_group")
```

Description

Welcome to the AMR package.

Details

AMR is a free and open-source R package to simplify the analysis and prediction of Antimicrobial Resistance (AMR) and to work with microbial and antimicrobial properties by using evidence-based methods. It supports any table format, including WHONET/EARS-Net data.

We created this package for both academic research and routine analysis at the Faculty of Medical Sciences of the University of Groningen and the Medical Microbiology & Infection Prevention (MMBI) department of the University Medical Center Groningen (UMCG). This R package is actively maintained and free software; you can freely use and distribute it for both personal and commercial (but not patent) purposes under the terms of the GNU General Public License version 2.0 (GPL-2), as published by the Free Software Foundation.

This package can be used for:

- Reference for microorganisms, since it contains all microbial (sub)species from the Catalogue of Life
- Interpreting raw MIC and disk diffusion values, based on the latest CLSI or EUCAST guidelines
- Calculating antimicrobial resistance
- Determining multi-drug resistance (MDR) / multi-drug resistant organisms (MDRO)
- Calculating empirical susceptibility of both mono therapy and combination therapy
- Predicting future antimicrobial resistance using regression models
- Getting properties for any microorganism (like Gram stain, species, genus or family)
- Getting properties for any antibiotic (like name, ATC code, defined daily dose or trade name)
- Plotting antimicrobial resistance
- Determining first isolates to be used for AMR analysis
- Applying EUCAST expert rules
- Descriptive statistics: frequency tables, kurtosis and skewness

Authors

Matthijs S. Berends[1,2] Christian F. Luz[1], Erwin E.A. Hassing[2], Corinna Glasner[1], Alex W. Friedrich[1], Bhanu N.M. Sinha[1]

[1] Department of Medical Microbiology, University of Groningen, University Medical Center Groningen, Groningen, the Netherlands - https://www.rug.nl https://www.umcg.nl
[2] Certe Medical Diagnostics & Advice, Groningen, the Netherlands - https://www.certe.nl

Read more on our website!

On our website https://msberends.gitlab.io/AMR you can find a tutorial about how to conduct AMR analysis, the complete documentation of all functions (which reads a lot easier than here in R) and an example analysis using WHONET data.

Contact us

For suggestions, comments or questions, please contact us at:

Matthijs S. Berends
m.s.berends [at] umcg [dot] nl
Department of Medical Microbiology, University of Groningen
University Medical Center Groningen
Post Office Box 30001
9700 RB Groningen

If you have found a bug, please file a new issue at:
https://gitlab.com/msberends/AMR/issues

---

antibiotics *Data set with ~450 antibiotics*

Description

A data set containing all antibiotics. Use `as.ab` or one of the `ab_property` functions to retrieve values from this data set. Three identifiers are included in this data set: an antibiotic ID (`ab`, primarily used in this package) as defined by WHONET/EARS-Net, an ATC code (`atc`) as defined by the WHO, and a Compound ID (`cid`) as found in PubChem. Other properties in this data set are derived from one or more of these codes.

Usage

```
antibiotics
```

Format

A `data.frame` with 453 observations and 13 variables:

`ab` Antibiotic ID as used in this package (like `AMC`), using the official EARS-Net (European Antimicrobial Resistance Surveillance Network) codes where available

`atc` ATC code (Anatomical Therapeutic Chemical) as defined by the WHOCC, like `J01CR02`

`cid` Compound ID as found in PubChem

`name` Official name as used by WHONET/EARS-Net or the WHO

`group` A short and concise group name, based on WHONET and WHOCC definitions

`atc_group1` Official pharmacological subgroup (3rd level ATC code) as defined by the WHOCC, like `"Macrolides, lincosamides and streptogramins"`

`atc_group2` Official chemical subgroup (4th level ATC code) as defined by the WHOCC, like `"Macrolides"`

`abbr` List of abbreviations as used in many countries, also for antibiotic susceptibility testing (AST)

`synonyms` Synonyms (often trade names) of a drug, as found in PubChem based on their compound ID

`oral_ddd` Defined Daily Dose (DDD), oral treatment

`oral_units` Units of `ddd_units`

`iv_ddd` Defined Daily Dose (DDD), parenteral treatment

`iv_units` Units of `iv_ddd`

Details

Properties that are based on an ATC code are only available when an ATC is available. These properties are: `atc_group1`, `atc_group2`, `oral_ddd`, `oral_units`, `iv_ddd` and `iv_units`

Synonyms (i.e. trade names) are derived from the Compound ID (`cid`) and consequently only available where a CID is available.

WHOCC

This package contains **all ~450 antimicrobial drugs** and their Anatomical Therapeutic Chemical (ATC) codes, ATC groups and Defined Daily Dose (DDD) from the World Health Organization Collaborating Centre for Drug Statistics Methodology (WHOCC, https://www.whocc.no) and the Pharmaceuticals Community Register of the European Commission (http://ec.europa.eu/health/documents/community-register/html/atc.htm).

These have become the gold standard for international drug utilisation monitoring and research.

The WHOCC is located in Oslo at the Norwegian Institute of Public Health and funded by the Norwegian government. The European Commission is the executive of the European Union and promotes its general interest.

Read more on our website!

On our website https://msberends.gitlab.io/AMR you can find a tutorial about how to conduct AMR analysis, the complete documentation of all functions (which reads a lot easier than here in R) and an example analysis using WHONET data.

Source

World Health Organization (WHO) Collaborating Centre for Drug Statistics Methodology (WHOCC): https://www.whocc.no/atc_ddd_index/

WHONET 2019 software: http://www.whonet.org/software.html

European Commission Public Health PHARMACEUTICALS - COMMUNITY REGISTER: http://ec.europa.eu/health/documents/community-register/html/atc.htm

See Also

`microorganisms`

as.ab *Transform to antibiotic ID*

Description

Use this function to determine the antibiotic code of one or more antibiotics. The data set antibiotics will be searched for abbreviations, official names and synonyms (brand names).

Usage

```
as.ab(x)

is.ab(x)
```

Arguments

x	character vector to determine to antibiotic ID

Details

All entries in the antibiotics data set have three different identifiers: a human readable EARS-Net code (column ab, used by ECDC and WHONET), an ATC code (column atc, used by WHO), and a CID code (column cid, Compound ID, used by PubChem). The data set contains more than 5,000 official brand names from many different countries, as found in PubChem.

Use the ab_property functions to get properties based on the returned antibiotic ID, see Examples.

Value

Character (vector) with class "ab". Unknown values will return NA.

Source

World Health Organization (WHO) Collaborating Centre for Drug Statistics Methodology: https://www.whocc.no/atc_ddd_index/

WHONET 2019 software: http://www.whonet.org/software.html

European Commission Public Health PHARMACEUTICALS - COMMUNITY REGISTER: http://ec.europa.eu/health/documents/community-register/html/atc.htm

WHOCC

This package contains **all ~450 antimicrobial drugs** and their Anatomical Therapeutic Chemical (ATC) codes, ATC groups and Defined Daily Dose (DDD) from the World Health Organization Collaborating Centre for Drug Statistics Methodology (WHOCC, https://www.whocc.no) and the Pharmaceuticals Community Register of the European Commission (http://ec.europa.eu/health/documents/community-register/html/atc.htm).

These have become the gold standard for international drug utilisation monitoring and research.

The WHOCC is located in Oslo at the Norwegian Institute of Public Health and funded by the Norwegian government. The European Commission is the executive of the European Union and promotes its general interest.

Read more on our website!

On our website https://msberends.gitlab.io/AMR you can find a tutorial about how to conduct AMR analysis, the complete documentation of all functions (which reads a lot easier than here in R) and an example analysis using WHONET data.

See Also

antibiotics for the dataframe that is being used to determine ATCs.

Examples

```
# These examples all return "ERY", the ID of Erythromycin:
as.ab("J01FA01")
as.ab("J 01 FA 01")
as.ab("Erythromycin")
as.ab("eryt")
as.ab("   eryt 123")
as.ab("ERYT")
as.ab("ERY")
as.ab("eritromicine") # spelled wrong, yet works
as.ab("Erythrocin")   # trade name
as.ab("Romycin")      # trade name

# Use ab_* functions to get a specific properties (see ?ab_property);
# they use as.ab() internally:
ab_name("J01FA01")    # "Erythromycin"
ab_name("eryt")       # "Erythromycin"
```

---

as.atc *Transform to ATC code*

Description

Use this function to determine the ATC code of one or more antibiotics. The data set antibiotics will be searched for abbreviations, official names and trade names.

Usage

```
as.atc(x)

is.atc(x)
```

Arguments

x	character vector to determine ATC code

Details

Use the ab_property functions to get properties based on the returned ATC code, see Examples.

In the ATC classification system, the active substances are classified in a hierarchy with five different levels. The system has fourteen main anatomical/pharmacological groups or 1st levels. Each ATC main group is divided into 2nd levels which could be either pharmacological or therapeutic groups. The 3rd and 4th levels are chemical, pharmacological or therapeutic subgroups and the 5th level is the chemical substance. The 2nd, 3rd and 4th levels are often used to identify pharmacological subgroups when that is considered more appropriate than therapeutic or chemical subgroups. Source: https://www.whocc.no/atc/structure_and_principles/

Value

Character (vector) with class "atc". Unknown values will return NA.

WHOCC

This package contains **all ~450 antimicrobial drugs** and their Anatomical Therapeutic Chemical (ATC) codes, ATC groups and Defined Daily Dose (DDD) from the World Health Organization Collaborating Centre for Drug Statistics Methodology (WHOCC, https://www.whocc.no) and the Pharmaceuticals Community Register of the European Commission (http://ec.europa.eu/health/documents/community-register/html/atc.htm).

These have become the gold standard for international drug utilisation monitoring and research.

The WHOCC is located in Oslo at the Norwegian Institute of Public Health and funded by the Norwegian government. The European Commission is the executive of the European Union and promotes its general interest.

Read more on our website!

On our website https://msberends.gitlab.io/AMR you can find a tutorial about how to conduct AMR analysis, the complete documentation of all functions (which reads a lot easier than here in R) and an example analysis using WHONET data.

See Also

antibiotics for the dataframe that is being used to determine ATCs.

Examples

```
# These examples all return "J01FA01", the ATC code of Erythromycin:
as.atc("J01FA01")
as.atc("Erythromycin")
as.atc("eryt")
as.atc("   eryt 123")
as.atc("ERYT")
as.atc("ERY")
```

`as.disk` *Class 'disk'*

Description

This transforms a vector to a new class `disk`, which is a growth zone size (around an antibiotic disk) in millimeters between 6 and 99.

Usage

```
as.disk(x, na.rm = FALSE)

is.disk(x)
```

Arguments

`x`	vector
`na.rm`	a logical indicating whether missing values should be removed

Details

Interpret disk values as RSI values with `as.rsi`. It supports guidelines from EUCAST and CLSI.

Value

Ordered integer factor with new class `disk`

Read more on our website!

On our website `https://msberends.gitlab.io/AMR` you can find a tutorial about how to conduct AMR analysis, the complete documentation of all functions (which reads a lot easier than here in R) and an example analysis using WHONET data.

See Also

`as.rsi`

Examples

```
# interpret disk values
as.rsi(x = 12,
       mo = as.mo("S. pneumoniae"),
       ab = "AMX",
       guideline = "EUCAST")
as.rsi(x = 12,
       mo = as.mo("S. pneumoniae"),
       ab = "AMX",
       guideline = "CLSI")
```

`as.mic` *Class 'mic'*

Description

This transforms a vector to a new class `mic`, which is an ordered factor with valid MIC values as levels. Invalid MIC values will be translated as `NA` with a warning.

Usage

```
as.mic(x, na.rm = FALSE)

is.mic(x)
```

Arguments

`x`	vector
`na.rm`	a logical indicating whether missing values should be removed

Details

Interpret MIC values as RSI values with `as.rsi`. It supports guidelines from EUCAST and CLSI.

Value

Ordered factor with new class `mic`

Read more on our website!

On our website `https://msberends.gitlab.io/AMR` you can find a tutorial about how to conduct AMR analysis, the complete documentation of all functions (which reads a lot easier than here in R) and an example analysis using WHONET data.

See Also

`as.rsi`

Examples

```
mic_data <- as.mic(c(">=32", "1.0", "1", "1.00", 8, "<=0.128", "8", "16", "16"))
is.mic(mic_data)

# this can also coerce combined MIC/RSI values:
as.mic("<=0.002; S") # will return <=0.002

# interpret MIC values
as.rsi(x = as.mic(2),
       mo = as.mo("S. pneumoniae"),
       ab = "AMX",
```

```
       guideline = "EUCAST")
as.rsi(x = as.mic(4),
       mo = as.mo("S. pneumoniae"),
       ab = "AMX",
       guideline = "EUCAST")

plot(mic_data)
barplot(mic_data)
freq(mic_data)
```

`as.mo` *Transform to microorganism ID*

Description

Use this function to determine a valid microorganism ID (`mo`). Determination is done using intelligent rules and the complete taxonomic kingdoms Bacteria, Chromista, Protozoa, Archaea and most microbial species from the kingdom Fungi (see Source). The input can be almost anything: a full name (like `"Staphylococcus aureus"`), an abbreviated name (like `"S. aureus"`), an abbreviation known in the field (like `"MRSA"`), or just a genus. Please see Examples.

Usage

```
as.mo(x, Becker = FALSE, Lancefield = FALSE, allow_uncertain = TRUE,
  reference_df = get_mo_source(), ...)

is.mo(x)

mo_failures()

mo_uncertainties()

mo_renamed()
```

Arguments

`x`	a character vector or a `data.frame` with one or two columns
`Becker`	a logical to indicate whether *Staphylococci* should be categorised into coagulase-negative *Staphylococci* ("CoNS") and coagulase-positive *Staphylococci* ("CoPS") instead of their own species, according to Karsten Becker *et al.* [1,2]. Note that this does not include species that were newly named after these publications, like *S. caeli*. This excludes *Staphylococcus aureus* at default, use `Becker = "all"` to also categorise *S. aureus* as "CoPS".
`Lancefield`	a logical to indicate whether beta-haemolytic *Streptococci* should be categorised into Lancefield groups instead of their own species, according to Rebecca C. Lancefield [3]. These *Streptococci* will be categorised in their first group, e.g.

Streptococcus dysgalactiae will be group C, although officially it was also categorised into groups G and L.

This excludes *Enterococci* at default (who are in group D), use `Lancefield = "all"` to also categorise all *Enterococci* as group D.

`allow_uncertain`
: a logical (`TRUE` or `FALSE`) or a value between 0 and 3 to indicate whether the input should be checked for less possible results, see Details

`reference_df`
: a `data.frame` to use for extra reference when translating `x` to a valid `mo`. See `set_mo_source` and `get_mo_source` to automate the usage of your own codes (e.g. used in your analysis or organisation).

`...`
: other parameters passed on to functions

Details

General info

A microbial ID from this package (class: `mo`) typically looks like these examples:

```
Code              Full name
---------------   --------------------------------------
B_KLBSL           Klebsiella
B_KLBSL_PNE       Klebsiella pneumoniae
B_KLBSL_PNE_RHI   Klebsiella pneumoniae rhinoscleromatis
|   |    |   |
|   |    |   |
|   |    |    ----> subspecies, a 3-4 letter acronym
|   |     ----> species, a 3-4 letter acronym
|    ----> genus, a 5-7 letter acronym, mostly without vowels
 ----> taxonomic kingdom: A (Archaea), AN (Animalia), B (Bacteria),
                          C (Chromista), F (Fungi), P (Protozoa) or
                          PL (Plantae)
```

Values that cannot be coered will be considered 'unknown' and will get the MO code `UNKNOWN`.

Use the `mo_property_*` functions to get properties based on the returned code, see Examples.

The algorithm uses data from the Catalogue of Life (see below) and from one other source (see `?microorganisms`).

Intelligent rules

This function uses intelligent rules to help getting fast and logical results. It tries to find matches in this order:

- Valid MO codes and full names: it first searches in already valid MO code and known genus/species combinations
- Human pathogenic prevalence: it first searches in more prevalent microorganisms, then less prevalent ones (see *Microbial prevalence of pathogens in humans* below)
- Taxonomic kingdom: it first searches in Bacteria/Chromista, then Fungi, then Protozoa
- Breakdown of input values: from here it starts to breakdown input values to find possible matches

A couple of effects because of these rules:

- `"E. coli"` will return the ID of *Escherichia coli* and not *Entamoeba coli*, although the latter would alphabetically come first
- `"H. influenzae"` will return the ID of *Haemophilus influenzae* and not *Haematobacter influenzae* for the same reason
- Something like `"stau"` or `"S aur"` will return the ID of *Staphylococcus aureus* and not *Staphylococcus auricularis*

This means that looking up human pathogenic microorganisms takes less time than looking up human non-pathogenic microorganisms.

Uncertain results

The algorithm can additionally use three different levels of uncertainty to guess valid results. The default is `allow_uncertain = TRUE`, which is equal to uncertainty level 2. Using `allow_uncertain = FALSE` will skip all of these additional rules:

- (uncertainty level 1): It tries to look for only matching genera
- (uncertainty level 1): It tries to look for previously accepted (but now invalid) taxonomic names
- (uncertainty level 2): It strips off values between brackets and the brackets itself, and re-evaluates the input with all previous rules
- (uncertainty level 2): It strips off words from the end one by one and re-evaluates the input with all previous rules
- (uncertainty level 3): It strips off words from the start one by one and re-evaluates the input with all previous rules
- (uncertainty level 3): It tries any part of the name

You can also use e.g. `as.mo(..., allow_uncertain = 1)` to only allow up to level 1 uncertainty.

Examples:

- `"Streptococcus group B (known as S. agalactiae)"`. The text between brackets will be removed and a warning will be thrown that the result *Streptococcus group B* (`B_STRPT_GRB`) needs review.
- `"S. aureus - please mind: MRSA"`. The last word will be stripped, after which the function will try to find a match. If it does not, the second last word will be stripped, etc. Again, a warning will be thrown that the result *Staphylococcus aureus* (`B_STPHY_AUR`) needs review.
- `"Fluoroquinolone-resistant Neisseria gonorrhoeae"`. The first word will be stripped, after which the function will try to find a match. A warning will be thrown that the result *Neisseria gonorrhoeae* (`B_NESSR_GON`) needs review.

Use `mo_failures()` to get a vector with all values that could not be coerced to a valid value.

Use `mo_uncertainties()` to get a data.frame with all values that were coerced to a valid value, but with uncertainty.

Use `mo_renamed()` to get a vector with all values that could be coerced based on an old, previously accepted taxonomic name.

Microbial prevalence of pathogens in humans

The intelligent rules take into account microbial prevalence of pathogens in humans. It uses three groups and all (sub)species are in only one group. These groups are:

- 1 (most prevalent): class is Gammaproteobacteria **or** genus is one of: *Enterococcus*, *Staphylococcus*, *Streptococcus*.
- 2: phylum is one of: Proteobacteria, Firmicutes, Actinobacteria, Sarcomastigophora **or** genus is one of: *Aspergillus*, *Bacteroides*, *Candida*, *Capnocytophaga*, *Chryseobacterium*, *Cryptococcus*, *Elisabethkingia*, *Flavobacterium*, *Fusobacterium*, *Giardia*, *Leptotrichia*, *Mycoplasma*, *Prevotella*, *Rhodotorula*, *Treponema*, *Trichophyton*, *Ureaplasma*.
- 3 (least prevalent): all others.

Group 1 contains all common Gram positives and Gram negatives, like all Enterobacteriaceae and e.g. *Pseudomonas* and *Legionella*.

Group 2 probably contains less microbial pathogens; all other members of phyla that were found in humans in the Northern Netherlands between 2001 and 2018.

Value

Character (vector) with class `"mo"`

Source

[1] Becker K *et al.* **Coagulase-Negative Staphylococci**. 2014. Clin Microbiol Rev. 27(4): 870–926. https://dx.doi.org/10.1128/CMR.00109-13

[2] Becker K *et al.* **Implications of identifying the recently defined members of the *S. aureus* complex, *S. argenteus* and *S. schweitzeri*: A position paper of members of the ESCMID Study Group for staphylococci and Staphylococcal Diseases (ESGS).** 2019. Clin Microbiol Infect. https://doi.org/10.1016/j.cmi.2019.02.028

[3] Lancefield RC **A serological differentiation of human and other groups of hemolytic streptococci**. 1933. J Exp Med. 57(4): 571–95. https://dx.doi.org/10.1084/jem.57.4.571

[4] Catalogue of Life: Annual Checklist (public online taxonomic database), http://www.catalogueoflife.org (check included annual version with `catalogue_of_life_version()`).

Catalogue of Life

This package contains the complete taxonomic tree of almost all microorganisms (~65,000 species) from the authoritative and comprehensive Catalogue of Life (http://www.catalogueoflife.org). The Catalogue of Life is the most comprehensive and authoritative global index of species currently available.

Click here for more information about the included taxa. The Catalogue of Life releases updates annually; check which version was included in this package with `catalogue_of_life_version()`.

Read more on our website!

On our website https://msberends.gitlab.io/AMR you can find a tutorial about how to conduct AMR analysis, the complete documentation of all functions (which reads a lot easier than here in R) and an example analysis using WHONET data.

See Also

microorganisms for the data.frame that is being used to determine ID's.
The mo_property functions (like mo_genus, mo_gramstain) to get properties based on the returned code.

Examples

```
# These examples all return "B_STPHY_AUR", the ID of S. aureus:
as.mo("sau") # WHONET code
as.mo("stau")
as.mo("STAU")
as.mo("staaur")
as.mo("S. aureus")
as.mo("S aureus")
as.mo("Staphylococcus aureus")
as.mo("Staphylococcus aureus (MRSA)")
as.mo("Sthafilokkockus aaureuz") # handles incorrect spelling
as.mo("MRSA")   # Methicillin Resistant S. aureus
as.mo("VISA")   # Vancomycin Intermediate S. aureus
as.mo("VRSA")   # Vancomycin Resistant S. aureus
as.mo(22242419) # Catalogue of Life ID

# Dyslexia is no problem - these all work:
as.mo("Ureaplasma urealyticum")
as.mo("Ureaplasma urealyticus")
as.mo("Ureaplasmium urealytica")
as.mo("Ureaplazma urealitycium")

as.mo("Streptococcus group A")
as.mo("GAS") # Group A Streptococci
as.mo("GBS") # Group B Streptococci

as.mo("S. epidermidis")                 # will remain species: B_STPHY_EPI
as.mo("S. epidermidis", Becker = TRUE)  # will not remain species: B_STPHY_CNS

as.mo("S. pyogenes")                    # will remain species: B_STRPT_PYO
as.mo("S. pyogenes", Lancefield = TRUE) # will not remain species: B_STRPT_GRA

# All mo_* functions use as.mo() internally too (see ?mo_property):
mo_genus("E. coli")           # returns "Escherichia"
mo_gramstain("E. coli")       # returns "Gram negative"#'

## Not run:
df$mo <- as.mo(df$microorganism_name)

# the select function of tidyverse is also supported:
library(dplyr)
df$mo <- df %>%
  select(microorganism_name) %>%
  as.mo()

# and can even contain 2 columns, which is convenient for genus/species combinations:
```

```
df$mo <- df %>%
  select(genus, species) %>%
  as.mo()
# although this works easier and does the same:
df <- df %>%
  mutate(mo = as.mo(paste(genus, species)))

## End(Not run)
```

`as.rsi` *Class 'rsi'*

Description

Interpret MIC values according to EUCAST or CLSI, or clean up existing RSI values. This transforms the input to a new class `rsi`, which is an ordered factor with levels `S < I < R`. Invalid antimicrobial interpretations will be translated as `NA` with a warning.

Usage

```
as.rsi(x, ...)

## S3 method for class 'mic'
as.rsi(x, mo, ab, guideline = "EUCAST", ...)

## S3 method for class 'disk'
as.rsi(x, mo, ab, guideline = "EUCAST", ...)

## S3 method for class 'data.frame'
as.rsi(x, col_mo = NULL, guideline = "EUCAST",
  ...)

is.rsi(x)

is.rsi.eligible(x, threshold = 0.05)
```

Arguments

`x`	vector of values (for class `mic`: an MIC value in mg/L, for class `disk`: a disk diffusion radius in millimeters)
`...`	parameters passed on to methods
`mo`	a microorganism code, generated with `as.mo`
`ab`	an antibiotic code, generated with `as.ab`
`guideline`	defaults to the latest included EUCAST guideline, run `unique(AMR::rsi_translation$guideline)` for all options
`col_mo`	column name of the unique IDs of the microorganisms (see `mo`), defaults to the first column of class `mo`. Values will be coerced using `as.mo`.
`threshold`	maximum fraction of `x` that is allowed to fail transformation, see Examples

Details

Run `unique(AMR::rsi_translation$guideline)` for a list of all supported guidelines.

After using `as.rsi`, you can use `eucast_rules` to (1) apply inferred susceptibility and resistance based on results of other antibiotics and (2) apply intrinsic resistance based on taxonomic properties of a microorganism.

The function `is.rsi.eligible` returns `TRUE` when a columns contains at most 5% invalid antimicrobial interpretations (not S and/or I and/or R), and `FALSE` otherwise. The threshold of 5% can be set with the `threshold` parameter.

Value

Ordered factor with new class `rsi`

Interpretation of S, I and R

In 2019, EUCAST has decided to change the definitions of susceptibility testing categories S, I and R as shown below. Results of several consultations on the new definitions are available on the EUCAST website under "Consultations".

- **S** - Susceptible, standard dosing regimen: A microorganism is categorised as "Susceptible, standard dosing regimen", when there is a high likelihood of therapeutic success using a standard dosing regimen of the agent.
- **I** - Susceptible, increased exposure: A microorganism is categorised as "Susceptible, Increased exposure" when there is a high likelihood of therapeutic success because exposure to the agent is increased by adjusting the dosing regimen or by its concentration at the site of infection.
- **R** - Resistant: A microorganism is categorised as "Resistant" when there is a high likelihood of therapeutic failure even when there is increased exposure.

Exposure is a function of how the mode of administration, dose, dosing interval, infusion time, as well as distribution and excretion of the antimicrobial agent will influence the infecting organism at the site of infection.

Source: `http://www.eucast.org/newsiandr/`.

This AMR package honours this new insight.

Read more on our website!

On our website `https://msberends.gitlab.io/AMR` you can find a tutorial about how to conduct AMR analysis, the complete documentation of all functions (which reads a lot easier than here in R) and an example analysis using WHONET data.

See Also

`as.mic`

Examples

```
rsi_data <- as.rsi(c(rep("S", 474), rep("I", 36), rep("R", 370)))
rsi_data <- as.rsi(c(rep("S", 474), rep("I", 36), rep("R", 370), "A", "B", "C"))
is.rsi(rsi_data)

# this can also coerce combined MIC/RSI values:
as.rsi("<= 0.002; S") # will return S

# interpret MIC values
as.rsi(x = as.mic(2),
       mo = as.mo("S. pneumoniae"),
       ab = "AMX",
       guideline = "EUCAST")
as.rsi(x = as.mic(4),
       mo = as.mo("S. pneumoniae"),
       ab = "AMX",
       guideline = "EUCAST")

plot(rsi_data)    # for percentages
barplot(rsi_data) # for frequencies
freq(rsi_data)    # frequency table with informative header

# using dplyr's mutate
library(dplyr)
septic_patients %>%
  mutate_at(vars(PEN:RIF), as.rsi)


# fastest way to transform all columns with already valid AB results to class `rsi`:
septic_patients %>%
  mutate_if(is.rsi.eligible,
            as.rsi)

# default threshold of `is.rsi.eligible` is 5%.
is.rsi.eligible(WHONET$`First name`) # fails, >80% is invalid
is.rsi.eligible(WHONET$`First name`, threshold = 0.99) # succeeds
```

---

atc_online_property *Get ATC properties from WHOCC website*

Description

Gets data from the WHO to determine properties of an ATC (e.g. an antibiotic) like name, defined daily dose (DDD) or standard unit.
This function requires an internet connection.

Usage

```
atc_online_property(atc_code, property, administration = "O",
```

```
  url = "https://www.whocc.no/atc_ddd_index/?code=%s&showdescription=no")

atc_online_groups(atc_code, ...)

atc_online_ddd(atc_code, ...)
```

Arguments

atc_code	a character or character vector with ATC code(s) of antibiotic(s)
property	property of an ATC code. Valid values are "ATC", "Name", "DDD", "U" ("unit"), "Adm.R", "Note" and groups. For this last option, all hierarchical groups of an ATC code will be returned, see Examples.
administration	type of administration when using property = "Adm.R", see Details
url	url of website of the WHO. The sign %s can be used as a placeholder for ATC codes.
...	parameters to pass on to atc_property

Details

Options for parameter administration:

- "Implant" = Implant
- "Inhal" = Inhalation
- "Instill" = Instillation
- "N" = nasal
- "O" = oral
- "P" = parenteral
- "R" = rectal
- "SL" = sublingual/buccal
- "TD" = transdermal
- "V" = vaginal

Abbreviations of return values when using property = "U" (unit):

- "g" = gram
- "mg" = milligram
- "mcg" = microgram
- "U" = unit
- "TU" = thousand units
- "MU" = million units
- "mmol" = millimole
- "ml" = milliliter (e.g. eyedrops)

Read more on our website!

On our website https://msberends.gitlab.io/AMR you can find a tutorial about how to conduct AMR analysis, the complete documentation of all functions (which reads a lot easier than here in R) and an example analysis using WHONET data.

Source

https://www.whocc.no/atc_ddd_alterations__cumulative/ddd_alterations/abbrevations/

Examples

```
# oral DDD (Defined Daily Dose) of amoxicillin
atc_online_property("J01CA04", "DDD", "O")
# parenteral DDD (Defined Daily Dose) of amoxicillin
atc_online_property("J01CA04", "DDD", "P")

atc_online_property("J01CA04", property = "groups") # search hierarchical groups of amoxicillin
# [1] "ANTIINFECTIVES FOR SYSTEMIC USE"
# [2] "ANTIBACTERIALS FOR SYSTEMIC USE"
# [3] "BETA-LACTAM ANTIBACTERIALS, PENICILLINS"
# [4] "Penicillins with extended spectrum"
```

---

availability *Check availability of columns*

Description

Easy check for availability of columns in a data set. This makes it easy to get an idea of which antibiotic combination can be used for calculation with e.g. portion_IR.

Usage

```
availability(tbl, width = NULL)
```

Arguments

tbl	a data.frame or list
width	number of characters to present the visual availability, defaults to filling the width of the console

Value

data.frame with column names of tbl as row names

Read more on our website!

On our website https://msberends.gitlab.io/AMR you can find a tutorial about how to conduct AMR analysis, the complete documentation of all functions (which reads a lot easier than here in R) and an example analysis using WHONET data.

Examples

```
availability(septic_patients)

library(dplyr)
septic_patients %>% availability()

septic_patients %>%
  select_if(is.rsi) %>%
  availability()

septic_patients %>%
  filter(mo == as.mo("E. coli")) %>%
  select_if(is.rsi) %>%
  availability()
```

---

catalogue_of_life *The Catalogue of Life*

Description

This package contains the complete taxonomic tree of almost all microorganisms from the authoritative and comprehensive Catalogue of Life.

Catalogue of Life

This package contains the complete taxonomic tree of almost all microorganisms (~65,000 species) from the authoritative and comprehensive Catalogue of Life (http://www.catalogueoflife.org). The Catalogue of Life is the most comprehensive and authoritative global index of species currently available.

Click here for more information about the included taxa. The Catalogue of Life releases updates annually; check which version was included in this package with catalogue_of_life_version().

Included taxa

Included are:

- All ~55,000 (sub)species from the kingdoms of Archaea, Bacteria and Protozoa
- All ~3,500 (sub)species from these orders of the kingdom of Fungi: Eurotiales, Onygenales, Pneumocystales, Saccharomycetales, Schizosaccharomycetales and Tremellales. The kingdom of Fungi is a very large taxon with almost 300,000 different (sub)species, of which most are not microbial (but rather macroscopic, like mushrooms). Because of this, not all fungi fit

the scope of this package and including everything would tremendously slow down our algorithms too. By only including the aforementioned taxonomic orders, the most relevant fungi are covered (like all species of *Aspergillus*, *Candida*, *Cryptococcus*, *Histplasma*, *Pneumocystis*, *Saccharomyces* and *Trichophyton*).

- All ~2,000 (sub)species from ~100 other relevant genera, from the kingdoms of Animalia and Plantae (like *Strongyloides* and *Taenia*)
- All ~21,000 previously accepted names of included (sub)species that have been taxonomically renamed
- The complete taxonomic tree of all included (sub)species: from kingdom to subspecies
- The responsible author(s) and year of scientific publication

The Catalogue of Life (http://www.catalogueoflife.org) is the most comprehensive and authoritative global index of species currently available. It holds essential information on the names, relationships and distributions of over 1.6 million species. The Catalogue of Life is used to support the major biodiversity and conservation information services such as the Global Biodiversity Information Facility (GBIF), Encyclopedia of Life (EoL) and the International Union for Conservation of Nature Red List. It is recognised by the Convention on Biological Diversity as a significant component of the Global Taxonomy Initiative and a contribution to Target 1 of the Global Strategy for Plant Conservation.

The syntax used to transform the original data to a cleansed R format, can be found here: https://gitlab.com/msberends/AMR/blob/master/data-raw/reproduction_of_microorganisms.R.

Read more on our website!

On our website https://msberends.gitlab.io/AMR you can find a tutorial about how to conduct AMR analysis, the complete documentation of all functions (which reads a lot easier than here in R) and an example analysis using WHONET data.

See Also

Data set `microorganisms` for the actual data.
Function `as.mo()` to use the data for intelligent determination of microorganisms.

Examples

```
# Get version info of included data set
catalogue_of_life_version()


# Get a note when a species was renamed
mo_shortname("Chlamydia psittaci")
# Note: 'Chlamydia psittaci' (Page, 1968) was renamed
#       'Chlamydophila psittaci' (Everett et al., 1999)
# [1] "C. psittaci"

# Get any property from the entire taxonomic tree for all included species
mo_class("E. coli")
# [1] "Gammaproteobacteria"
```

```
mo_family("E. coli")
# [1] "Enterobacteriaceae"

mo_gramstain("E. coli") # based on kingdom and phylum, see ?mo_gramstain
# [1] "Gram negative"

mo_ref("E. coli")
# [1] "Castellani et al., 1919"

# Do not get mistaken - the package only includes microorganisms
mo_kingdom("C. elegans")
# [1] "Bacteria"                        # Bacteria?!
mo_name("C. elegans")
# [1] "Chroococcus limneticus elegans"  # Because a microorganism was found
```

`catalogue_of_life_version`

Version info of included Catalogue of Life

Description

This function returns information about the included data from the Catalogue of Life.

Usage

```
catalogue_of_life_version()
```

Details

For DSMZ, see `?microorganisms`.

Value

a `list`, which prints in pretty format

Catalogue of Life

This package contains the complete taxonomic tree of almost all microorganisms (~65,000 species) from the authoritative and comprehensive Catalogue of Life (http://www.catalogueoflife.org). The Catalogue of Life is the most comprehensive and authoritative global index of species currently available.

Click here for more information about the included taxa. The Catalogue of Life releases updates annually; check which version was included in this package with `catalogue_of_life_version()`.

Read more on our website!

On our website https://msberends.gitlab.io/AMR you can find a tutorial about how to conduct AMR analysis, the complete documentation of all functions (which reads a lot easier than here in R) and an example analysis using WHONET data.

See Also

microorganisms

Examples

```
library(dplyr)
microorganisms %>% freq(kingdom)
microorganisms %>% group_by(kingdom) %>% freq(phylum, nmax = NULL)
```

---

count *Count isolates*

Description

These functions can be used to count resistant/susceptible microbial isolates. All functions support quasiquotation with pipes, can be used in `dplyr`s `summarise` and support grouped variables, see *Examples*.

`count_R` and `count_IR` can be used to count resistant isolates, `count_S` and `count_SI` can be used to count susceptible isolates.

Usage

```
count_R(..., also_single_tested = FALSE)

count_IR(..., also_single_tested = FALSE)

count_I(..., also_single_tested = FALSE)

count_SI(..., also_single_tested = FALSE)

count_S(..., also_single_tested = FALSE)

count_all(...)

n_rsi(...)

count_df(data, translate_ab = "name", language = get_locale(),
  combine_SI = TRUE, combine_IR = FALSE)
```

Arguments

`...`	one or more vectors (or columns) with antibiotic interpretations. They will be transformed internally with `as.rsi` if needed.

`also_single_tested`
: a logical to indicate whether (in combination therapies) also observations should be included where not all antibiotics were tested, but at least one of the tested antibiotics contains a target interpretation (e.g. S in case of `portion_S` and R in case of `portion_R`). **This would lead to selection bias in almost all cases.**

`data`
: a `data.frame` containing columns with class `rsi` (see `as.rsi`)

`translate_ab`
: a column name of the `antibiotics` data set to translate the antibiotic abbreviations to, using `ab_property`

`language`
: language of the returned text, defaults to system language (see `get_locale`) and can also be set with `getOption("AMR_locale")`. Use `language = NULL` or `language = ""` to prevent translation.

`combine_SI`
: a logical to indicate whether all values of S and I must be merged into one, so the output only consists of S+I vs. R (susceptible vs. resistant). This used to be the parameter `combine_IR`, but this now follows the redefinition by EUCAST about the interpretion of I (increased exposure) in 2019, see section 'Interpretation of S, I and R' below. Default is `TRUE`.

`combine_IR`
: a logical to indicate whether all values of I and R must be merged into one, so the output only consists of S vs. I+R (susceptible vs. non-susceptible). This is outdated, see parameter `combine_SI`.

Details

These functions are meant to count isolates. Use the `portion_*` functions to calculate microbial resistance.

The function `n_rsi` is an alias of `count_all`. They can be used to count all available isolates, i.e. where all input antibiotics have an available result (S, I or R). Their use is equal to `n_distinct`. Their function is equal to `count_S(...) + count_IR(...)`.

The function `count_df` takes any variable from `data` that has an `"rsi"` class (created with `as.rsi`) and counts the amounts of S, I and R. The resulting *tidy data* (see Source) `data.frame` will have three rows (S/I/R) and a column for each variable with class `"rsi"`.

The function `rsi_df` works exactly like `count_df`, but adds the percentage of S, I and R.

Value

Integer

Interpretation of S, I and R

In 2019, EUCAST has decided to change the definitions of susceptibility testing categories S, I and R as shown below. Results of several consultations on the new definitions are available on the EUCAST website under "Consultations".

- **S** - Susceptible, standard dosing regimen: A microorganism is categorised as "Susceptible, standard dosing regimen", when there is a high likelihood of therapeutic success using a standard dosing regimen of the agent.

- **I** - Susceptible, increased exposure: A microorganism is categorised as "Susceptible, Increased exposure" when there is a high likelihood of therapeutic success because exposure to the agent is increased by adjusting the dosing regimen or by its concentration at the site of infection.
- **R** - Resistant: A microorganism is categorised as "Resistant" when there is a high likelihood of therapeutic failure even when there is increased exposure.

Exposure is a function of how the mode of administration, dose, dosing interval, infusion time, as well as distribution and excretion of the antimicrobial agent will influence the infecting organism at the site of infection.

Source: http://www.eucast.org/newsiandr/.

This AMR package honours this new insight.

Read more on our website!

On our website https://msberends.gitlab.io/AMR you can find a tutorial about how to conduct AMR analysis, the complete documentation of all functions (which reads a lot easier than here in R) and an example analysis using WHONET data.

Source

Wickham H. **Tidy Data.** The Journal of Statistical Software, vol. 59, 2014. http://vita.had.co.nz/papers/tidy-data.html

See Also

portion_* to calculate microbial resistance and susceptibility.

Examples

```
# septic_patients is a data set available in the AMR package. It is true, genuine data.
?septic_patients

# Count resistant isolates
count_R(septic_patients$AMX)
count_IR(septic_patients$AMX)

# Or susceptible isolates
count_S(septic_patients$AMX)
count_SI(septic_patients$AMX)

# Count all available isolates
count_all(septic_patients$AMX)
n_rsi(septic_patients$AMX)

# Since n_rsi counts available isolates, you can
# calculate back to count e.g. non-susceptible isolates.
# This results in the same:
count_IR(septic_patients$AMX)
portion_IR(septic_patients$AMX) * n_rsi(septic_patients$AMX)
```

```
library(dplyr)
septic_patients %>%
  group_by(hospital_id) %>%
  summarise(R  = count_R(CIP),
            I  = count_I(CIP),
            S  = count_S(CIP),
            n1 = count_all(CIP),  # the actual total; sum of all three
            n2 = n_rsi(CIP),      # same - analogous to n_distinct
            total = n())          # NOT the number of tested isolates!

# Count co-resistance between amoxicillin/clav acid and gentamicin,
# so we can see that combination therapy does a lot more than mono therapy.
# Please mind that `portion_S` calculates percentages right away instead.
count_S(septic_patients$AMC)   # S = 1342 (71.4%)
count_all(septic_patients$AMC) # n = 1879

count_S(septic_patients$GEN)   # S = 1372 (74.0%)
count_all(septic_patients$GEN) # n = 1855

with(septic_patients,
     count_S(AMC, GEN))         # S = 1660 (92.3%)
with(septic_patients,           # n = 1798
     n_rsi(AMC, GEN))

# Get portions S/I/R immediately of all rsi columns
septic_patients %>%
  select(AMX, CIP) %>%
  count_df(translate = FALSE)

# It also supports grouping variables
septic_patients %>%
  select(hospital_id, AMX, CIP) %>%
  group_by(hospital_id) %>%
  count_df(translate = FALSE)
```

`eucast_rules` *EUCAST rules*

Description

Apply susceptibility rules as defined by the European Committee on Antimicrobial Susceptibility Testing (EUCAST, http://eucast.org), see *Source*. This includes (1) expert rules, (2) intrinsic resistance and (3) inferred resistance as defined in their breakpoint tables.

Usage

```
eucast_rules(x, col_mo = NULL, info = TRUE, rules = c("breakpoints",
  "expert", "other", "all"), verbose = FALSE, ...)
```

Arguments

x	data with antibiotic columns, like e.g. AMX and AMC
col_mo	column name of the unique IDs of the microorganisms (see mo), defaults to the first column of class mo. Values will be coerced using as.mo.
info	print progress
rules	a character vector that specifies which rules should be applied - one or more of c("breakpoints", "expert", "other", "all")
verbose	a logical to indicate whether extensive info should be returned as a data.frame with info about which rows and columns are effected. It runs all EUCAST rules, but will not be applied to an output - only an informative data.frame with changes will be returned as output.
...	column name of an antibiotic, see section Antibiotics

Details

Note: This function does not translate MIC values to RSI values. Use as.rsi for that.
Note: When ampicillin (AMP, J01CA01) is not available but amoxicillin (AMX, J01CA04) is, the latter will be used for all rules where there is a dependency on ampicillin. These drugs are interchangeable when it comes to expression of antimicrobial resistance.

The file containing all EUCAST rules is located here: https://gitlab.com/msberends/AMR/blob/master/data-raw/eucast_rules.tsv.

Value

The input of x, possibly with edited values of antibiotics. Or, if verbose = TRUE, a data.frame with all original and new values of the affected bug-drug combinations.

Antibiotics

To define antibiotics column names, leave as it is to determine it automatically with guess_ab_col or input a text (case-insensitive), or use NULL to skip a column (e.g. TIC = NULL to skip ticarcillin). Manually defined but non-existing columns will be skipped with a warning.

The following antibiotics are used for the functions eucast_rules and mdro. These are shown in the format '**antimicrobial ID**: name (*ATC code*)', sorted by name:

AMK: amikacin (J01GB06), **AMX**: amoxicillin (J01CA04), **AMC**: amoxicillin/clavulanic acid (J01CR02), **AMP**: ampicillin (J01CA01), **AZM**: azithromycin (J01FA10), **AZL**: azlocillin (J01CA09), **ATM**: aztreonam (J01DF01), **CAP**: capreomycin (J04AB30), **RID**: cefaloridine (J01DB02), **CZO**: cefazolin (J01DB04), **FEP**: cefepime (J01DE01), **CTX**: cefotaxime (J01DD01), **FOX**: cefoxitin (J01DC01), **CED**: cefradine (J01DB09), **CAZ**: ceftazidime (J01DD02), **CRO**: ceftriaxone (J01DD04), **CXM**: cefuroxime (J01DC02), **CHL**: chloramphenicol (J01BA01), **CIP**: ciprofloxacin (J01MA02), **CLR**: clarithromycin (J01FA09), **CLI**: clindamycin (J01FF01), **COL**: colistin (J01XB01), **DAP**: daptomycin (J01XX09), **DOX**: doxycycline (J01AA02), **ETP**: ertapenem (J01DH03), **ERY**: erythromycin (J01FA01), **ETH**: ethambutol (J04AK02), **FLC**: flucloxacillin (J01CF05), **FOS**: fosfomycin (J01XX01), **FUS**: fusidic acid (J01XC01), **GAT**: gatifloxacin (J01MA16), **GEN**: gentamicin (J01GB03), **IPM**: imipenem (J01DH51), **INH**: isoniazid (J04AC01), **KAN**: kanamycin (J01GB04), **LVX**: levofloxacin (J01MA12), **LIN**: lincomycin (J01FF02), **LNZ**: linezolid (J01XX08),

MEM: meropenem (J01DH02), **MTR**: metronidazole (J01XD01), **MEZ**: mezlocillin (J01CA10), **MNO**: minocycline (J01AA08), **MFX**: moxifloxacin (J01MA14), **NAL**: nalidixic acid (J01MB02), **NEO**: neomycin (J01GB05), **NET**: netilmicin (J01GB07), **NIT**: nitrofurantoin (J01XE01), **NOR**: norfloxacin (J01MA06), **NOV**: novobiocin (an ATCvet code: QJ01XX95), **OFX**: ofloxacin (J01MA01), **OXA**: oxacillin (J01CF04), **PEN**: penicillin G (J01CE01), **PIP**: piperacillin (J01CA12), **TZP**: piperacillin/tazobactam (J01CR05), **PLB**: polymyxin B (J01XB02), **PRI**: pristinamycin (J01FG01), **PZA**: pyrazinamide (J04AK01), **QDA**: quinupristin/dalfopristin (J01FG02), **RIB**: rifabutin (J04AB04), **RIF**: rifampicin (J04AB02), **RIF**: rifampin (J04AB02), **RFP**: rifapentine (J04AB05), **RXT**: roxithromycin (J01FA06), **SIS**: sisomicin (J01GB08), **TEC**: teicoplanin (J01XA02), **TCY**: tetracycline (J01AA07), **TIC**: ticarcillin (J01CA13), **TGC**: tigecycline (J01AA12), **TOB**: tobramycin (J01GB01), **TMP**: trimethoprim (J01EA01), **SXT**: trimethoprim/sulfamethoxazole (J01EE01), **VAN**: vancomycin (J01XA01).

Read more on our website!

On our website https://msberends.gitlab.io/AMR you can find a tutorial about how to conduct AMR analysis, the complete documentation of all functions (which reads a lot easier than here in R) and an example analysis using WHONET data.

Source

- EUCAST Expert Rules. Version 2.0, 2012.
  Leclercq et al. **EUCAST expert rules in antimicrobial susceptibility testing.** *Clin Microbiol Infect.* 2013;19(2):141-60.
  https://doi.org/10.1111/j.1469-0691.2011.03703.x
- EUCAST Expert Rules, Intrinsic Resistance and Exceptional Phenotypes Tables. Version 3.1, 2016.
  http://www.eucast.org/fileadmin/src/media/PDFs/EUCAST_files/Expert_Rules/Expert_rules_intrinsic_exceptional_V3.1.pdf
- EUCAST Breakpoint tables for interpretation of MICs and zone diameters. Version 9.0, 2019.
  http://www.eucast.org/fileadmin/src/media/PDFs/EUCAST_files/Breakpoint_tables/v_9.0_Breakpoint_Tables.xlsx

Examples

```
a <- eucast_rules(septic_patients)

a <- data.frame(mo = c("Staphylococcus aureus",
                       "Enterococcus faecalis",
                       "Escherichia coli",
                       "Klebsiella pneumoniae",
                       "Pseudomonas aeruginosa"),
                VAN = "-",       # Vancomycin
                AMX = "-",       # Amoxicillin
                COL = "-",       # Colistin
                CAZ = "-",       # Ceftazidime
                CXM = "-",       # Cefuroxime
                PEN = "S",       # Penicillin G
                FOX = "S",       # Cefoxitin
                stringsAsFactors = FALSE)
```

```
a
#                       mo  VAN  AMX  COL  CAZ  CXM  PEN  FOX
# 1  Staphylococcus aureus    -    -    -    -    -    S    S
# 2  Enterococcus faecalis    -    -    -    -    -    S    S
# 3       Escherichia coli    -    -    -    -    -    S    S
# 4  Klebsiella pneumoniae    -    -    -    -    -    S    S
# 5 Pseudomonas aeruginosa    -    -    -    -    -    S    S


# apply EUCAST rules: 18 results are forced as R or S
b <- eucast_rules(a)

b
#                       mo  VAN  AMX  COL  CAZ  CXM  PEN  FOX
# 1  Staphylococcus aureus    -    S    R    R    S    S    S
# 2  Enterococcus faecalis    -    -    R    R    R    S    R
# 3       Escherichia coli    R    -    -    -    -    R    S
# 4  Klebsiella pneumoniae    R    R    -    -    -    R    S
# 5 Pseudomonas aeruginosa    R    R    -    -    R    R    R


# do not apply EUCAST rules, but rather get a a data.frame
# with 18 rows, containing all details about the transformations:
c <- eucast_rules(a, verbose = TRUE)
```

`filter_ab_class` *Filter isolates on result in antibiotic class*

Description

Filter isolates on results in specific antibiotic variables based on their class (ATC groups). This makes it easy to get a list of isolates that were tested for e.g. any aminoglycoside.

Usage

```
filter_ab_class(tbl, ab_class, result = NULL, scope = "any", ...)

filter_aminoglycosides(tbl, result = NULL, scope = "any", ...)

filter_carbapenems(tbl, result = NULL, scope = "any", ...)

filter_cephalosporins(tbl, result = NULL, scope = "any", ...)

filter_1st_cephalosporins(tbl, result = NULL, scope = "any", ...)

filter_2nd_cephalosporins(tbl, result = NULL, scope = "any", ...)

filter_3rd_cephalosporins(tbl, result = NULL, scope = "any", ...)
```

```
filter_4th_cephalosporins(tbl, result = NULL, scope = "any", ...)

filter_fluoroquinolones(tbl, result = NULL, scope = "any", ...)

filter_glycopeptides(tbl, result = NULL, scope = "any", ...)

filter_macrolides(tbl, result = NULL, scope = "any", ...)

filter_tetracyclines(tbl, result = NULL, scope = "any", ...)
```

Arguments

tbl	a data set
ab_class	an antimicrobial class, like "carbapenems". More specifically, this should be a text that can be found in a 4th level ATC group (chemical subgroup) or a 5th level ATC group (chemical substance), please see this explanation on the WHOCC website.
result	an antibiotic result: S, I or R (or a combination of more of them)
scope	the scope to check which variables to check, can be "any" (default) or "all"
...	parameters passed on to filter_at

Details

The antibiotics data set will be searched for ab_class in the columns atc_group1 and atc_group2 (case-insensitive). Next, tbl will be checked for column names with a value in any abbreviations, codes or official names found in the antibiotics data set.

Examples

```
library(dplyr)

# filter on isolates that have any result for any aminoglycoside
septic_patients %>% filter_aminoglycosides()

# this is essentially the same as (but without determination of column names):
septic_patients %>%
  filter_at(.vars = vars(c("GEN", "TOB", "AMK", "KAN")),
            .vars_predicate = any_vars(. %in% c("S", "I", "R")))


# filter on isolates that show resistance to ANY aminoglycoside
septic_patients %>% filter_aminoglycosides("R")

# filter on isolates that show resistance to ALL aminoglycosides
septic_patients %>% filter_aminoglycosides("R", "all")

# filter on isolates that show resistance to
# any aminoglycoside and any fluoroquinolone
septic_patients %>%
```

```
  filter_aminoglycosides("R") %>%
  filter_fluoroquinolones("R")

# filter on isolates that show resistance to
# all aminoglycosides and all fluoroquinolones
septic_patients %>%
  filter_aminoglycosides("R", "all") %>%
  filter_fluoroquinolones("R", "all")
```

`first_isolate` *Determine first (weighted) isolates*

Description

Determine first (weighted) isolates of all microorganisms of every patient per episode and (if needed) per specimen type.

Usage

```
first_isolate(x, col_date = NULL, col_patient_id = NULL,
  col_mo = NULL, col_testcode = NULL, col_specimen = NULL,
  col_icu = NULL, col_keyantibiotics = NULL, episode_days = 365,
  testcodes_exclude = NULL, icu_exclude = FALSE,
  specimen_group = NULL, type = "keyantibiotics", ignore_I = TRUE,
  points_threshold = 2, info = TRUE, ...)

filter_first_isolate(x, col_date = NULL, col_patient_id = NULL,
  col_mo = NULL, ...)

filter_first_weighted_isolate(x, col_date = NULL,
  col_patient_id = NULL, col_mo = NULL, col_keyantibiotics = NULL,
  ...)
```

Arguments

`x`	a `data.frame` containing isolates.
`col_date`	column name of the result date (or date that is was received on the lab), defaults to the first column of with a date class
`col_patient_id`	column name of the unique IDs of the patients, defaults to the first column that starts with 'patient' or 'patid' (case insensitive)
`col_mo`	column name of the unique IDs of the microorganisms (see `mo`), defaults to the first column of class `mo`. Values will be coerced using `as.mo`.
`col_testcode`	column name of the test codes. Use `col_testcode = NULL` to **not** exclude certain test codes (like test codes for screening). In that case `testcodes_exclude` will be ignored.
`col_specimen`	column name of the specimen type or group

col_icu	column name of the logicals (TRUE/FALSE) whether a ward or department is an Intensive Care Unit (ICU)
col_keyantibiotics	column name of the key antibiotics to determine first *weighted* isolates, see key_antibiotics. Defaults to the first column that starts with 'key' followed by 'ab' or 'antibiotics' (case insensitive). Use col_keyantibiotics = FALSE to prevent this.
episode_days	episode in days after which a genus/species combination will be determined as 'first isolate' again
testcodes_exclude	character vector with test codes that should be excluded (case-insensitive)
icu_exclude	logical whether ICU isolates should be excluded (rows with value TRUE in column col_icu)
specimen_group	value in column col_specimen to filter on
type	type to determine weighed isolates; can be "keyantibiotics" or "points", see Details
ignore_I	logical to determine whether antibiotic interpretations with "I" will be ignored when type = "keyantibiotics", see Details
points_threshold	points until the comparison of key antibiotics will lead to inclusion of an isolate when type = "points", see Details
info	print progress
...	parameters passed on to the first_isolate function

Details

WHY THIS IS SO IMPORTANT

To conduct an analysis of antimicrobial resistance, you should only include the first isolate of every patient per episode [1]. If you would not do this, you could easily get an overestimate or underestimate of the resistance of an antibiotic. Imagine that a patient was admitted with an MRSA and that it was found in 5 different blood cultures the following week. The resistance percentage of oxacillin of all *S. aureus* isolates would be overestimated, because you included this MRSA more than once. It would be selection bias.

The functions filter_first_isolate and filter_first_weighted_isolate are helper functions to quickly filter on first isolates. The function filter_first_isolate is essentially equal to:

```
x %>%
  mutate(only_firsts = first_isolate(x, ...)) %>%
  filter(only_firsts == TRUE) %>%
  select(-only_firsts)
```

The function filter_first_weighted_isolate is essentially equal to:

```
x %>%
  mutate(keyab = key_antibiotics(.)) %>%
```

```
  mutate(only_weighted_firsts = first_isolate(x,
                                              col_keyantibiotics = "keyab", ...)) %>%
  filter(only_weighted_firsts == TRUE) %>%
  select(-only_weighted_firsts)
```

Value

Logical vector

Key antibiotics

There are two ways to determine whether isolates can be included as first *weighted* isolates which will give generally the same results:

1. Using `type = "keyantibiotics"` **and parameter** `ignore_I`
Any difference from S to R (or vice versa) will (re)select an isolate as a first weighted isolate. With `ignore_I = FALSE`, also differences from I to S|R (or vice versa) will lead to this. This is a reliable method and 30-35 times faster than method 2. Read more about this in the `key_antibiotics` function.

2. Using `type = "points"` **and parameter** `points_threshold`
A difference from I to S|R (or vice versa) means 0.5 points, a difference from S to R (or vice versa) means 1 point. When the sum of points exceeds `points_threshold`, which default to `2`, an isolate will be (re)selected as a first weighted isolate.

Read more on our website!

On our website https://msberends.gitlab.io/AMR you can find a tutorial about how to conduct AMR analysis, the complete documentation of all functions (which reads a lot easier than here in R) and an example analysis using WHONET data.

Source

Methodology of this function is based on: **M39 Analysis and Presentation of Cumulative Antimicrobial Susceptibility Test Data, 4th Edition**, 2014, *Clinical and Laboratory Standards Institute (CLSI)*. https://clsi.org/standards/products/microbiology/documents/m39/.

See Also

`key_antibiotics`

Examples

```
# septic_patients is a dataset available in the AMR package. It is true, genuine data.
?septic_patients

library(dplyr)
# Filter on first isolates:
septic_patients %>%
  mutate(first_isolate = first_isolate(.,
```

```
                                      col_date = "date",
                                      col_patient_id = "patient_id",
                                      col_mo = "mo")) %>%
  filter(first_isolate == TRUE)

# Which can be shortened to:
septic_patients %>%
  filter_first_isolate()
# or for first weighted isolates:
septic_patients %>%
  filter_first_weighted_isolate()

# Now let's see if first isolates matter:
A <- septic_patients %>%
  group_by(hospital_id) %>%
  summarise(count = n_rsi(GEN),            # gentamicin availability
            resistance = portion_IR(GEN))  # gentamicin resistance

B <- septic_patients %>%
  filter_first_weighted_isolate() %>%      # the 1st isolate filter
  group_by(hospital_id) %>%
  summarise(count = n_rsi(GEN),            # gentamicin availability
            resistance = portion_IR(GEN))  # gentamicin resistance

# Have a look at A and B.
# B is more reliable because every isolate is only counted once.
# Gentamicin resitance in hospital D appears to be 3.1% higher than
# when you (erroneously) would have used all isolates for analysis.


## OTHER EXAMPLES:

## Not run:

# set key antibiotics to a new variable
x$keyab <- key_antibiotics(x)

x$first_isolate <-
  first_isolate(x)

x$first_isolate_weighed <-
  first_isolate(x,
                col_keyantibiotics = 'keyab')

x$first_blood_isolate <-
  first_isolate(x,
                specimen_group = 'Blood')

x$first_blood_isolate_weighed <-
  first_isolate(x,
                specimen_group = 'Blood',
                col_keyantibiotics = 'keyab')
```

```
x$first_urine_isolate <-
  first_isolate(x,
                specimen_group = 'Urine')

x$first_urine_isolate_weighed <-
  first_isolate(x,
                specimen_group = 'Urine',
                col_keyantibiotics = 'keyab')

x$first_resp_isolate <-
  first_isolate(x,
                specimen_group = 'Respiratory')

x$first_resp_isolate_weighed <-
  first_isolate(x,
                specimen_group = 'Respiratory',
                col_keyantibiotics = 'keyab')

## End(Not run)
```

`freq` *Frequency table*

Description

Create a frequency table of a vector with items or a `data.frame`. Supports quasiquotation and markdown for reports. Best practice is: `data %>% freq(var)`.
`top_freq` can be used to get the top/bottom *n* items of a frequency table, with counts as names.

Usage

```
freq(x, ..., sort.count = TRUE, nmax = getOption("max.print.freq"),
  na.rm = TRUE, row.names = TRUE, markdown = !interactive(),
  digits = 2, quote = FALSE, header = TRUE, title = NULL,
  na = "<NA>", droplevels = TRUE, sep = " ",
  decimal.mark = getOption("OutDec"), big.mark = ifelse(decimal.mark !=
  ",", ",", ","))

frequency_tbl(x, ..., sort.count = TRUE,
  nmax = getOption("max.print.freq"), na.rm = TRUE, row.names = TRUE,
  markdown = !interactive(), digits = 2, quote = FALSE,
  header = TRUE, title = NULL, na = "<NA>", droplevels = TRUE,
  sep = " ", decimal.mark = getOption("OutDec"),
  big.mark = ifelse(decimal.mark != ",", ",", "."))

top_freq(f, n)

header(f, property = NULL)
```

```
## S3 method for class 'freq'
print(x, nmax = getOption("max.print.freq", default = 15),
  markdown = !interactive(), header = TRUE,
  decimal.mark = getOption("OutDec"), big.mark = ifelse(decimal.mark !=
  ",", ",", "."), ...)
```

Arguments

x	vector of any class or a data.frame, tibble (may contain a grouping variable) or table
...	up to nine different columns of x when x is a data.frame or tibble, to calculate frequencies from - see Examples. Also supports quasiquotion.
sort.count	sort on count, i.e. frequencies. This will be TRUE at default for everything except when using grouping variables.
nmax	number of row to print. The default, 15, uses getOption("max.print.freq"). Use nmax = 0, nmax = Inf, nmax = NULL or nmax = NA to print all rows.
na.rm	a logical value indicating whether NA values should be removed from the frequency table. The header (if set) will always print the amount of NAs.
row.names	a logical value indicating whether row indices should be printed as 1:nrow(x)
markdown	a logical value indicating whether the frequency table should be printed in markdown format. This will print all rows (except when nmax is defined) and is default behaviour in non-interactive R sessions (like when knitting RMarkdown files).
digits	how many significant digits are to be used for numeric values in the header (not for the items themselves, that depends on getOption("digits"))
quote	a logical value indicating whether or not strings should be printed with surrounding quotes
header	a logical value indicating whether an informative header should be printed
title	text to show above frequency table, at default to tries to coerce from the variables passed to x
na	a character string that should be used to show empty (NA) values (only useful when na.rm = FALSE)
droplevels	a logical value indicating whether in factors empty levels should be dropped
sep	a character string to separate the terms when selecting multiple columns
decimal.mark	used for prettying (longish) numerical and complex sequences. Passed to prettyNum: that help page explains the details.
big.mark	used for prettying (longish) numerical and complex sequences. Passed to prettyNum: that help page explains the details.
f	a frequency table
n	number of top *n* items to return, use -n for the bottom *n* items. It will include more than n rows if there are ties.
property	property in header to return this value directly

Details

Frequency tables (or frequency distributions) are summaries of the distribution of values in a sample. With the 'freq' function, you can create univariate frequency tables. Multiple variables will be pasted into one variable, so it forces a univariate distribution. This package also has a vignette available to explain the use of this function further, run `browseVignettes("AMR")` to read it.

For numeric values of any class, these additional values will all be calculated with `na.rm = TRUE` and shown into the header:

- Mean, using `mean`
- Standard Deviation, using `sd`
- Coefficient of Variation (CV), the standard deviation divided by the mean
- Mean Absolute Deviation (MAD), using `mad`
- Tukey Five-Number Summaries (minimum, Q1, median, Q3, maximum), using `fivenum`
- Interquartile Range (IQR) calculated as `Q3 - Q1` using the Tukey Five-Number Summaries, i.e. **not** using the `quantile` function
- Coefficient of Quartile Variation (CQV, sometimes called coefficient of dispersion), calculated as `(Q3 - Q1) / (Q3 + Q1)` using the Tukey Five-Number Summaries
- Outliers (total count and unique count), using `boxplot.stats`

For dates and times of any class, these additional values will be calculated with `na.rm = TRUE` and shown into the header:

- Oldest, using `min`
- Newest, using `max`, with difference between newest and oldest
- Median, using `median`, with percentage since oldest

In factors, all factor levels that are not existing in the input data will be dropped.

The function `top_freq` uses `top_n` internally and will include more than `n` rows if there are ties.

Value

A `data.frame` (with an additional class `"freq"`) with five columns: `item`, `count`, `percent`, `cum_count` and `cum_percent`.

Read more on our website!

On our website https://msberends.gitlab.io/AMR you can find a tutorial about how to conduct AMR analysis, the complete documentation of all functions (which reads a lot easier than here in R) and an example analysis using WHONET data.

Examples

```
library(dplyr)

# this all gives the same result:
freq(septic_patients$hospital_id)
freq(septic_patients[, "hospital_id"])
```

```
septic_patients$hospital_id %>% freq()
septic_patients[, "hospital_id"] %>% freq()
septic_patients %>% freq("hospital_id")
septic_patients %>% freq(hospital_id)  #<- easiest to remember (tidyverse)


# you could also use `select` or `pull` to get your variables
septic_patients %>%
  filter(hospital_id == "A") %>%
  select(mo) %>%
  freq()


# multiple selected variables will be pasted together
septic_patients %>%
  left_join_microorganisms %>%
  freq(genus, species)

# functions as quasiquotation are also supported
septic_patients %>%
  freq(mo_genus(mo), mo_species(mo))


# group a variable and analyse another
septic_patients %>%
  group_by(hospital_id) %>%
  freq(gender)


# get top 10 bugs of hospital A as a vector
septic_patients %>%
  filter(hospital_id == "A") %>%
  freq(mo) %>%
  top_freq(10)


# save frequency table to an object
years <- septic_patients %>%
  mutate(year = format(date, "%Y")) %>%
  freq(year)


# show only the top 5
years %>% print(nmax = 5)


# save to an object with formatted percentages
years <- format(years)


# print a histogram of numeric values
septic_patients %>%
  freq(age) %>%
```

```
  hist()

# or a boxplot of numeric values
septic_patients %>%
  freq(age) %>%
  boxplot()

# or even a boxplot per group
septic_patients %>%
  group_by(hospital_id) %>%
  freq(age) %>%
  boxplot()

# or print all points to a regular plot
septic_patients %>%
  freq(age) %>%
  plot()


# transform to a data.frame or tibble
septic_patients %>%
  freq(age) %>%
  as.data.frame()


# or transform (back) to a vector
septic_patients %>%
  freq(age) %>%
  as.vector()

identical(septic_patients %>%
            freq(age) %>%
            as.vector() %>%
            sort(),
          sort(septic_patients$age)) # TRUE


# it also supports `table` objects
table(septic_patients$gender,
      septic_patients$age) %>%
  freq(sep = " **sep** ")


# only get selected columns
septic_patients %>%
  freq(hospital_id) %>%
  select(item, percent)

septic_patients %>%
  freq(hospital_id) %>%
  select(-count, -cum_count)
```

```
# check differences between frequency tables
diff(freq(septic_patients$TMP),
     freq(septic_patients$SXT))
```

`g.test` G-*test for Count Data*

Description

`g.test` performs chi-squared contingency table tests and goodness-of-fit tests, just like `chisq.test` but is more reliable [1]. A *G*-test can be used to see whether the number of observations in each category fits a theoretical expectation (called a ***G*-test of goodness-of-fit**), or to see whether the proportions of one variable are different for different values of the other variable (called a ***G*-test of independence**).

Usage

```
g.test(x, y = NULL, p = rep(1/length(x), length(x)),
  rescale.p = FALSE)
```

Arguments

`x`	a numeric vector or matrix. `x` and `y` can also both be factors.
`y`	a numeric vector; ignored if `x` is a matrix. If `x` is a factor, `y` should be a factor of the same length.
`p`	a vector of probabilities of the same length of `x`. An error is given if any entry of `p` is negative.
`rescale.p`	a logical scalar; if TRUE then `p` is rescaled (if necessary) to sum to 1. If `rescale.p` is FALSE, and `p` does not sum to 1, an error is given.

Details

If `x` is a matrix with one row or column, or if `x` is a vector and `y` is not given, then a *goodness-of-fit test* is performed (`x` is treated as a one-dimensional contingency table). The entries of `x` must be non-negative integers. In this case, the hypothesis tested is whether the population probabilities equal those in `p`, or are all equal if `p` is not given.

If `x` is a matrix with at least two rows and columns, it is taken as a two-dimensional contingency table: the entries of `x` must be non-negative integers. Otherwise, `x` and `y` must be vectors or factors of the same length; cases with missing values are removed, the objects are coerced to factors, and the contingency table is computed from these. Then Pearson's chi-squared test is performed of the null hypothesis that the joint distribution of the cell counts in a 2-dimensional contingency table is the product of the row and column marginals.

The p-value is computed from the asymptotic chi-squared distribution of the test statistic.

In the contingency table case simulation is done by random sampling from the set of all contingency tables with given marginals, and works only if the marginals are strictly positive. Note that this is

not the usual sampling situation assumed for a chi-squared test (like the *G*-test) but rather that for Fisher's exact test.

In the goodness-of-fit case simulation is done by random sampling from the discrete distribution specified by `p`, each sample being of size `n = sum(x)`. This simulation is done in R and may be slow.

Value

A list with class `"htest"` containing the following components:

`statistic`	the value the chi-squared test statistic.
`parameter`	the degrees of freedom of the approximate chi-squared distribution of the test statistic, `NA` if the p-value is computed by Monte Carlo simulation.
`p.value`	the p-value for the test.
`method`	a character string indicating the type of test performed, and whether Monte Carlo simulation or continuity correction was used.
`data.name`	a character string giving the name(s) of the data.
`observed`	the observed counts.
`expected`	the expected counts under the null hypothesis.
`residuals`	the Pearson residuals, `(observed - expected) / sqrt(expected)`.
`stdres`	standardized residuals, `(observed - expected) / sqrt(V)`, where `V` is the residual cell variance (Agresti, 2007, section 2.4.5 for the case where `x` is a matrix, `n * p * (1 - p)` otherwise).

G-test of goodness-of-fit (likelihood ratio test)

Use the *G*-test of goodness-of-fit when you have one nominal variable with two or more values (such as male and female, or red, pink and white flowers). You compare the observed counts of numbers of observations in each category with the expected counts, which you calculate using some kind of theoretical expectation (such as a 1:1 sex ratio or a 1:2:1 ratio in a genetic cross).

If the expected number of observations in any category is too small, the *G*-test may give inaccurate results, and you should use an exact test instead (`fisher.test`).

The *G*-test of goodness-of-fit is an alternative to the chi-square test of goodness-of-fit (`chisq.test`); each of these tests has some advantages and some disadvantages, and the results of the two tests are usually very similar.

G-test of independence

Use the *G*-test of independence when you have two nominal variables, each with two or more possible values. You want to know whether the proportions for one variable are different among values of the other variable.

It is also possible to do a *G*-test of independence with more than two nominal variables. For example, Jackson et al. (2013) also had data for children under 3, so you could do an analysis of old vs. young, thigh vs. arm, and reaction vs. no reaction, all analyzed together.

Fisher's exact test (fisher.test) is an **exact** test, where the *G*-test is still only an **approximation**. For any 2x2 table, Fisher's Exact test may be slower but will still run in seconds, even if the sum of your observations is multiple millions.

The *G*-test of independence is an alternative to the chi-square test of independence (chisq.test), and they will give approximately the same results.

How the test works

Unlike the exact test of goodness-of-fit (fisher.test), the *G*-test does not directly calculate the probability of obtaining the observed results or something more extreme. Instead, like almost all statistical tests, the *G*-test has an intermediate step; it uses the data to calculate a test statistic that measures how far the observed data are from the null expectation. You then use a mathematical relationship, in this case the chi-square distribution, to estimate the probability of obtaining that value of the test statistic.

The *G*-test uses the log of the ratio of two likelihoods as the test statistic, which is why it is also called a likelihood ratio test or log-likelihood ratio test. The formula to calculate a *G*-statistic is:

```
G <- 2 * sum(x * log(x / E))
```

where `E` are the expected values. Since this is chi-square distributed, the p value can be calculated with:

```
p <- stats::pchisq(G, df, lower.tail = FALSE)
```

where `df` are the degrees of freedom.

If there are more than two categories and you want to find out which ones are significantly different from their null expectation, you can use the same method of testing each category vs. the sum of all categories, with the Bonferroni correction. You use *G*-tests for each category, of course.

Read more on our website!

On our website https://msberends.gitlab.io/AMR you can find a tutorial about how to conduct AMR analysis, the complete documentation of all functions (which reads a lot easier than here in R) and an example analysis using WHONET data.

Source

This code is almost identical to chisq.test, except that:

- The calculation of the statistic was changed to `2 * sum(x * log(x / E))`
- Yates' continuity correction was removed as it does not apply to a *G*-test
- The possibility to simulate p values with `simulate.p.value` was removed

References

[1] McDonald, J.H. 2014. **Handbook of Biological Statistics (3rd ed.)**. Sparky House Publishing, Baltimore, Maryland. http://www.biostathandbook.com/gtestgof.html.

See Also

chisq.test

Examples

```
# = EXAMPLE 1 =
# Shivrain et al. (2006) crossed clearfield rice (which are resistant
# to the herbicide imazethapyr) with red rice (which are susceptible to
# imazethapyr). They then crossed the hybrid offspring and examined the
# F2 generation, where they found 772 resistant plants, 1611 moderately
# resistant plants, and 737 susceptible plants. If resistance is controlled
# by a single gene with two co-dominant alleles, you would expect a 1:2:1
# ratio.

x <- c(772, 1611, 737)
G <- g.test(x, p = c(1, 2, 1) / 4)
# G$p.value = 0.12574.

# There is no significant difference from a 1:2:1 ratio.
# Meaning: resistance controlled by a single gene with two co-dominant
# alleles, is plausible.


# = EXAMPLE 2 =
# Red crossbills (Loxia curvirostra) have the tip of the upper bill either
# right or left of the lower bill, which helps them extract seeds from pine
# cones. Some have hypothesized that frequency-dependent selection would
# keep the number of right and left-billed birds at a 1:1 ratio. Groth (1992)
# observed 1752 right-billed and 1895 left-billed crossbills.

x <- c(1752, 1895)
g.test(x)
# p = 0.01787343

# There is a significant difference from a 1:1 ratio.
# Meaning: there are significantly more left-billed birds.
```

---

ggplot_rsi *AMR plots with* ggplot2

Description

Use these functions to create bar plots for antimicrobial resistance analysis. All functions rely on internal ggplot2 functions.

Usage

```
ggplot_rsi(data, position = NULL, x = "antibiotic",
  fill = "interpretation", facet = NULL, breaks = seq(0, 1, 0.1),
  limits = NULL, translate_ab = "name", combine_SI = TRUE,
  combine_IR = FALSE, language = get_locale(), fun = count_df,
  nrow = NULL, colours = c(S = "#61a8ff", SI = "#61a8ff", I =
```

```
  "#61f7ff", IR = "#ff6961", R = "#ff6961"), datalabels = TRUE,
  datalabels.size = 2.5, datalabels.colour = "gray15", title = NULL,
  subtitle = NULL, caption = NULL, x.title = NULL, y.title = NULL,
  ...)

geom_rsi(position = NULL, x = c("antibiotic", "interpretation"),
  fill = "interpretation", translate_ab = "name",
  language = get_locale(), combine_SI = TRUE, combine_IR = FALSE,
  fun = count_df, ...)

facet_rsi(facet = c("interpretation", "antibiotic"), nrow = NULL)

scale_y_percent(breaks = seq(0, 1, 0.1), limits = NULL)

scale_rsi_colours(colours = c(S = "#61a8ff", SI = "#61a8ff", I =
  "#61f7ff", IR = "#ff6961", R = "#ff6961"))

theme_rsi()

labels_rsi_count(position = NULL, x = "antibiotic",
  translate_ab = "name", combine_SI = TRUE, combine_IR = FALSE,
  datalabels.size = 3, datalabels.colour = "gray15")
```

Arguments

data	a data.frame with column(s) of class "rsi" (see as.rsi)
position	position adjustment of bars, either "fill" (default when fun is count_df), "stack" (default when fun is portion_df) or "dodge"
x	variable to show on x axis, either "antibiotic" (default) or "interpretation" or a grouping variable
fill	variable to categorise using the plots legend, either "antibiotic" (default) or "interpretation" or a grouping variable
facet	variable to split plots by, either "interpretation" (default) or "antibiotic" or a grouping variable
breaks	numeric vector of positions
limits	numeric vector of length two providing limits of the scale, use NA to refer to the existing minimum or maximum
translate_ab	a column name of the antibiotics data set to translate the antibiotic abbreviations to, using ab_property
combine_SI	a logical to indicate whether all values of S and I must be merged into one, so the output only consists of S+I vs. R (susceptible vs. resistant). This used to be the parameter combine_IR, but this now follows the redefinition by EUCAST about the interpretion of I (increased exposure) in 2019, see section 'Interpretation of S, I and R' below. Default is TRUE.
combine_IR	a logical to indicate whether all values of I and R must be merged into one, so the output only consists of S vs. I+R (susceptible vs. non-susceptible). This is outdated, see parameter combine_SI.

language	language of the returned text, defaults to system language (see get_locale) and can also be set with getOption("AMR_locale"). Use language = NULL or language = "" to prevent translation.
fun	function to transform data, either count_df (default) or portion_df
nrow	(when using facet) number of rows
colours	a named vector with colours for the bars. The names must be one or more of: S, SI, I, IR, R or be FALSE to use default ggplot2 colours.
datalabels	show datalabels using labels_rsi_count, will only be shown when fun = count_df
datalabels.size	size of the datalabels
datalabels.colour	colour of the datalabels
title	text to show as title of the plot
subtitle	text to show as subtitle of the plot
caption	text to show as caption of the plot
x.title	text to show as x axis description
y.title	text to show as y axis description
...	other parameters passed on to geom_rsi

Details

At default, the names of antibiotics will be shown on the plots using ab_name. This can be set with the translate_ab parameter. See count_df.

The functions

geom_rsi will take any variable from the data that has an rsi class (created with as.rsi) using fun (count_df at default, can also be portion_df) and will plot bars with the percentage R, I and S. The default behaviour is to have the bars stacked and to have the different antibiotics on the x axis.

facet_rsi creates 2d plots (at default based on S/I/R) using facet_wrap.

scale_y_percent transforms the y axis to a 0 to 100% range using scale_continuous.

scale_rsi_colours sets colours to the bars: pastel blue for S, pastel turquoise for I and pastel red for R, using scale_brewer.

theme_rsi is a ggplot theme with minimal distraction.

labels_rsi_count print datalabels on the bars with percentage and amount of isolates using geom_text

ggplot_rsi is a wrapper around all above functions that uses data as first input. This makes it possible to use this function after a pipe (%>%). See Examples.

Read more on our website!

On our website https://msberends.gitlab.io/AMR you can find a tutorial about how to conduct AMR analysis, the complete documentation of all functions (which reads a lot easier than here in R) and an example analysis using WHONET data.

Examples

```
library(dplyr)
library(ggplot2)

# get antimicrobial results for drugs against a UTI:
ggplot(septic_patients %>% select(AMX, NIT, FOS, TMP, CIP)) +
  geom_rsi()

# prettify the plot using some additional functions:
df <- septic_patients %>% select(AMX, NIT, FOS, TMP, CIP)
ggplot(df) +
  geom_rsi() +
  scale_y_percent() +
  scale_rsi_colours() +
  labels_rsi_count() +
  theme_rsi()

# or better yet, simplify this using the wrapper function - a single command:
septic_patients %>%
  select(AMX, NIT, FOS, TMP, CIP) %>%
  ggplot_rsi()

# get only portions and no counts:
septic_patients %>%
  select(AMX, NIT, FOS, TMP, CIP) %>%
  ggplot_rsi(fun = portion_df)

# add other ggplot2 parameters as you like:
septic_patients %>%
  select(AMX, NIT, FOS, TMP, CIP) %>%
  ggplot_rsi(width = 0.5,
             colour = "black",
             size = 1,
             linetype = 2,
             alpha = 0.25)

septic_patients %>%
  select(AMX) %>%
  ggplot_rsi(colours = c(SI = "yellow"))

# resistance of ciprofloxacine per age group
septic_patients %>%
  mutate(first_isolate = first_isolate(.)) %>%
  filter(first_isolate == TRUE,
         mo == as.mo("E. coli")) %>%
  # `age_group` is also a function of this package:
  group_by(age_group = age_groups(age)) %>%
  select(age_group,
         CIP) %>%
  ggplot_rsi(x = "age_group")
```

```
# for colourblind mode, use divergent colours from the viridis package:
septic_patients %>%
  select(AMX, NIT, FOS, TMP, CIP) %>%
  ggplot_rsi() + scale_fill_viridis_d()
# a shorter version which also adjusts data label colours:
septic_patients %>%
  select(AMX, NIT, FOS, TMP, CIP) %>%
  ggplot_rsi(colours = FALSE)


# it also supports groups (don't forget to use the group var on `x` or `facet`):
septic_patients %>%
  select(hospital_id, AMX, NIT, FOS, TMP, CIP) %>%
  group_by(hospital_id) %>%
  ggplot_rsi(x = "hospital_id",
             facet = "antibiotic",
             nrow = 1,
             title = "AMR of Anti-UTI Drugs Per Hospital",
             x.title = "Hospital",
             datalabels = FALSE)

# genuine analysis: check 3 most prevalent microorganisms
septic_patients %>%
  # create new bacterial ID's, with all CoNS under the same group (Becker et al.)
  mutate(mo = as.mo(mo, Becker = TRUE)) %>%
  # filter on top three bacterial ID's
  filter(mo %in% top_freq(freq(.$mo), 3)) %>%
  # filter on first isolates
  filter_first_isolate() %>%
  # get short MO names (like "E. coli")
  mutate(bug = mo_shortname(mo, Becker = TRUE)) %>%
  # select this short name and some antiseptic drugs
  select(bug, CXM, GEN, CIP) %>%
  # group by MO
  group_by(bug) %>%
  # plot the thing, putting MOs on the facet
  ggplot_rsi(x = "antibiotic",
             facet = "bug",
             translate_ab = FALSE,
             nrow = 1,
             title = "AMR of Top Three Microorganisms In Blood Culture Isolates",
           subtitle = expression(paste("Only First Isolates, CoNS grouped according to Becker ",
                                       italic("et al."), " (2014)")),
             x.title = "Antibiotic (EARS-Net code)")
```

---

guess_ab_col *Guess antibiotic column*

Description

This tries to find a column name in a data set based on information from the antibiotics data set. Also supports WHONET abbreviations.

Usage

```
guess_ab_col(x = NULL, search_string = NULL, verbose = FALSE)
```

Arguments

x	a data.frame
search_string	a text to search x for
verbose	a logical to indicate whether additional info should be printed

Details

You can look for an antibiotic (trade) name or abbreviation and it will search x and the antibiotics data set for any column containing a name or ATC code of that antibiotic. **Longer columns names take precendence over shorter column names.**

Value

A column name of x, or NULL when no result is found.

Read more on our website!

On our website https://msberends.gitlab.io/AMR you can find a tutorial about how to conduct AMR analysis, the complete documentation of all functions (which reads a lot easier than here in R) and an example analysis using WHONET data.

Examples

```
df <- data.frame(amox = "S",
                 tetr = "R")

guess_ab_col(df, "amoxicillin")
# [1] "amox"
guess_ab_col(df, "J01AA07") # ATC code of tetracycline
# [1] "tetr"

guess_ab_col(df, "J01AA07", verbose = TRUE)
# Note: Using column `tetr` as input for "J01AA07".
# [1] "tetr"

# WHONET codes
df <- data.frame(AMP_ND10 = "R",
                 AMC_ED20 = "S")
guess_ab_col(df, "ampicillin")
# [1] "AMP_ND10"
guess_ab_col(df, "J01CR02")
```

```
# [1] "AMC_ED20"
guess_ab_col(df, as.ab("augmentin"))
# [1] "AMC_ED20"

# Longer names take precendence:
df <- data.frame(AMP_ED2 = "S",
                 AMP_ED20 = "S")
guess_ab_col(df, "ampicillin")
# [1] "AMP_ED20"
```

`join` *Join a table with* `microorganisms`

Description

Join the dataset `microorganisms` easily to an existing table or character vector.

Usage

```
inner_join_microorganisms(x, by = NULL, suffix = c("2", ""), ...)

left_join_microorganisms(x, by = NULL, suffix = c("2", ""), ...)

right_join_microorganisms(x, by = NULL, suffix = c("2", ""), ...)

full_join_microorganisms(x, by = NULL, suffix = c("2", ""), ...)

semi_join_microorganisms(x, by = NULL, ...)

anti_join_microorganisms(x, by = NULL, ...)
```

Arguments

`x`	existing table to join, or character vector
`by`	a variable to join by - if left empty will search for a column with class `mo` (created with `as.mo`) or will be `"mo"` if that column name exists in `x`, could otherwise be a column name of `x` with values that exist in `microorganisms$mo` (like `by = "bacteria_id"`), or another column in `microorganisms` (but then it should be named, like `by = c("my_genus_species" = "fullname")`)
`suffix`	if there are non-joined duplicate variables in `x` and `y`, these suffixes will be added to the output to disambiguate them. Should be a character vector of length 2.
`...`	other parameters to pass on to `dplyr::join`.

Details

Note: As opposed to the `join` functions of `dplyr`, characters vectors are supported and at default existing columns will get a suffix `"2"` and the newly joined columns will not get a suffix. See `join` for more information.

Read more on our website!

On our website https://msberends.gitlab.io/AMR you can find a tutorial about how to conduct AMR analysis, the complete documentation of all functions (which reads a lot easier than here in R) and an example analysis using WHONET data.

Examples

```
left_join_microorganisms(as.mo("K. pneumoniae"))
left_join_microorganisms("B_KLBSL_PNE")

library(dplyr)
septic_patients %>% left_join_microorganisms()

df <- data.frame(date = seq(from = as.Date("2018-01-01"),
                            to = as.Date("2018-01-07"),
                            by = 1),
                 bacteria = as.mo(c("S. aureus", "MRSA", "MSSA", "STAAUR",
                                    "E. coli", "E. coli", "E. coli")),
                 stringsAsFactors = FALSE)
colnames(df)
df_joined <- left_join_microorganisms(df, "bacteria")
colnames(df_joined)
```

---

key_antibiotics *Key antibiotics for first* weighted *isolates*

Description

These function can be used to determine first isolates (see first_isolate). Using key antibiotics to determine first isolates is more reliable than without key antibiotics. These selected isolates will then be called first *weighted* isolates.

Usage

```
key_antibiotics(x, col_mo = NULL, universal_1 = guess_ab_col(x,
  "amoxicillin"), universal_2 = guess_ab_col(x,
  "amoxicillin/clavulanic acid"), universal_3 = guess_ab_col(x,
  "cefuroxime"), universal_4 = guess_ab_col(x,
  "piperacillin/tazobactam"), universal_5 = guess_ab_col(x,
  "ciprofloxacin"), universal_6 = guess_ab_col(x,
  "trimethoprim/sulfamethoxazole"), GramPos_1 = guess_ab_col(x,
  "vancomycin"), GramPos_2 = guess_ab_col(x, "teicoplanin"),
  GramPos_3 = guess_ab_col(x, "tetracycline"),
  GramPos_4 = guess_ab_col(x, "erythromycin"),
  GramPos_5 = guess_ab_col(x, "oxacillin"), GramPos_6 = guess_ab_col(x,
  "rifampin"), GramNeg_1 = guess_ab_col(x, "gentamicin"),
  GramNeg_2 = guess_ab_col(x, "tobramycin"),
  GramNeg_3 = guess_ab_col(x, "colistin"), GramNeg_4 = guess_ab_col(x,
```

```
  "cefotaxime"), GramNeg_5 = guess_ab_col(x, "ceftazidime"),
  GramNeg_6 = guess_ab_col(x, "meropenem"), warnings = TRUE, ...)

key_antibiotics_equal(y, z, type = c("keyantibiotics", "points"),
  ignore_I = TRUE, points_threshold = 2, info = FALSE)
```

Arguments

x	table with antibiotics coloms, like AMX or amox
col_mo	column name of the unique IDs of the microorganisms (see mo), defaults to the first column of class mo. Values will be coerced using as.mo.
universal_1, universal_2, universal_3, universal_4, universal_5, universal_6	column names of **broad-spectrum** antibiotics, case-insensitive. At default, the columns containing these antibiotics will be guessed with guess_ab_col.
GramPos_1, GramPos_2, GramPos_3, GramPos_4, GramPos_5, GramPos_6	column names of antibiotics for **Gram-positives**, case-insensitive. At default, the columns containing these antibiotics will be guessed with guess_ab_col.
GramNeg_1, GramNeg_2, GramNeg_3, GramNeg_4, GramNeg_5, GramNeg_6	column names of antibiotics for **Gram-negatives**, case-insensitive. At default, the columns containing these antibiotics will be guessed with guess_ab_col.
warnings	give warning about missing antibiotic columns, they will anyway be ignored
...	other parameters passed on to function
y, z	characters to compare
type	type to determine weighed isolates; can be "keyantibiotics" or "points", see Details
ignore_I	logical to determine whether antibiotic interpretations with "I" will be ignored when type = "keyantibiotics", see Details
points_threshold	points until the comparison of key antibiotics will lead to inclusion of an isolate when type = "points", see Details
info	print progress

Details

The function key_antibiotics returns a character vector with 12 antibiotic results for every isolate. These isolates can then be compared using key_antibiotics_equal, to check if two isolates have generally the same antibiogram. Missing and invalid values are replaced with a dot (".").
The first_isolate function only uses this function on the same microbial species from the same patient. Using this, an MRSA will be included after a susceptible *S. aureus* (MSSA) found within the same episode (see episode parameter of first_isolate). Without key antibiotic comparison it would not.

At default, the antibiotics that are used for **Gram-positive bacteria** are:
amoxicillin, amoxicillin/clavulanic acid, cefuroxime, piperacillin/tazobactam, ciprofloxacin, trimethoprim/sulfamethoxazole (until here is universal), vancomycin, teicoplanin, tetracycline, erythromycin, oxacillin, rifampin.

At default, the antibiotics that are used for **Gram-negative bacteria** are:
amoxicillin, amoxicillin/clavulanic acid, cefuroxime, piperacillin/tazobactam, ciprofloxacin, trimethoprim/sulfamethoxazole (until here is universal), gentamicin, tobramycin, colistin, cefotaxime, ceftazidime, meropenem.

The function `key_antibiotics_equal` checks the characters returned by `key_antibiotics` for equality, and returns a logical vector.

Key antibiotics

There are two ways to determine whether isolates can be included as first *weighted* isolates which will give generally the same results:

1. Using `type = "keyantibiotics"` and parameter `ignore_I`
Any difference from S to R (or vice versa) will (re)select an isolate as a first weighted isolate. With `ignore_I = FALSE`, also differences from I to S|R (or vice versa) will lead to this. This is a reliable method and 30-35 times faster than method 2. Read more about this in the `key_antibiotics` function.

2. Using `type = "points"` and parameter `points_threshold`
A difference from I to S|R (or vice versa) means 0.5 points, a difference from S to R (or vice versa) means 1 point. When the sum of points exceeds `points_threshold`, which default to 2, an isolate will be (re)selected as a first weighted isolate.

Read more on our website!

On our website https://msberends.gitlab.io/AMR you can find a tutorial about how to conduct AMR analysis, the complete documentation of all functions (which reads a lot easier than here in R) and an example analysis using WHONET data.

See Also

`first_isolate`

Examples

```
# septic_patients is a dataset available in the AMR package
?septic_patients

library(dplyr)
# set key antibiotics to a new variable
my_patients <- septic_patients %>%
  mutate(keyab = key_antibiotics(.)) %>%
  mutate(
    # now calculate first isolates
    first_regular = first_isolate(., col_keyantibiotics = FALSE),
    # and first WEIGHTED isolates
    first_weighted = first_isolate(., col_keyantibiotics = "keyab")
  )

# Check the difference, in this data set it results in 7% more isolates:
```

```
sum(my_patients$first_regular, na.rm = TRUE)
sum(my_patients$first_weighted, na.rm = TRUE)


# output of the `key_antibiotics` function could be like this:
strainA <- "SSSRR.S.R..S"
strainB <- "SSSIRSSSRSSS"

key_antibiotics_equal(strainA, strainB)
# TRUE, because I is ignored (as well as missing values)

key_antibiotics_equal(strainA, strainB, ignore_I = FALSE)
# FALSE, because I is not ignored and so the 4th value differs
```

`kurtosis` *Kurtosis of the sample*

Description

Kurtosis is a measure of the "tailedness" of the probability distribution of a real-valued random variable.

Usage

```
kurtosis(x, na.rm = FALSE)

## Default S3 method:
kurtosis(x, na.rm = FALSE)

## S3 method for class 'matrix'
kurtosis(x, na.rm = FALSE)

## S3 method for class 'data.frame'
kurtosis(x, na.rm = FALSE)
```

Arguments

`x`	a vector of values, a `matrix` or a `data frame`
`na.rm`	a logical value indicating whether `NA` values should be stripped before the computation proceeds.

Read more on our website!

On our website `https://msberends.gitlab.io/AMR` you can find a tutorial about how to conduct AMR analysis, the complete documentation of all functions (which reads a lot easier than here in R) and an example analysis using WHONET data.

See Also

skewness

---

like *Pattern Matching*

Description

Convenient wrapper around grep to match a pattern: `a %like% b`. It always returns a `logical` vector and is always case-insensitive. Also, `pattern` (`b`) can be as long as `x` (`a`) to compare items of each index in both vectors.

Usage

```
like(x, pattern)

x %like% pattern
```

Arguments

`x`	a character vector where matches are sought, or an object which can be coerced by `as.character` to a character vector. Long vectors are supported.
`pattern`	character string containing a regular expression (or character string for `fixed = TRUE`) to be matched in the given character vector. Coerced by `as.character` to a character string if possible. If a character vector of length 2 or more is supplied, the first element is used with a warning. Missing values are allowed except for `regexpr` and `gregexpr`.

Details

Using RStudio? This function can also be inserted from the Addins menu and can have its own Keyboard Shortcut like Ctrl+Shift+L or Cmd+Shift+L (see Tools > Modify Keyboard Shortcuts...).

Value

A `logical` vector

Read more on our website!

On our website `https://msberends.gitlab.io/AMR` you can find a tutorial about how to conduct AMR analysis, the complete documentation of all functions (which reads a lot easier than here in R) and an example analysis using WHONET data.

Source

Idea from the `like` function from the `data.table` package, but made it case insensitive at default and let it support multiple patterns.

See Also

grep

Examples

```
# simple test
a <- "This is a test"
b <- "TEST"
a %like% b
#> TRUE
b %like% a
#> FALSE

# also supports multiple patterns, length must be equal to x
a <- c("Test case", "Something different", "Yet another thing")
b <- c("case", "diff", "yet")
a %like% b
#> TRUE TRUE TRUE

# get frequencies of bacteria whose name start with 'Ent' or 'ent'
library(dplyr)
septic_patients %>%
  left_join_microorganisms() %>%
  filter(genus %like% '^ent') %>%
  freq(genus, species)
```

---

mdro *Determine multidrug-resistant organisms (MDRO)*

Description

Determine which isolates are multidrug-resistant organisms (MDRO) according to country-specific guidelines.

Usage

```
mdro(x, country = NULL, guideline = NULL, col_mo = NULL,
  info = TRUE, verbose = FALSE, ...)

brmo(..., country = "nl")

mrgn(x, country = "de", ...)

mdr_tb(x, guideline = "TB", ...)

eucast_exceptional_phenotypes(x, guideline = "EUCAST", ...)
```

Arguments

x	table with antibiotic columns, like e.g. AMX and AMC
country	country code to determine guidelines. Should be or a code from the list of ISO 3166-1 alpha-2 country codes. Case-insensitive.
guideline	a specific guideline to mention. For some countries this will be determined automatically, see Details. EUCAST guidelines will be used when left empty, see Details.
col_mo	column name of the unique IDs of the microorganisms (see mo), defaults to the first column of class mo. Values will be coerced using as.mo.
info	print progress
verbose	print additional info: missing antibiotic columns per parameter
...	column name of an antibiotic, see section Antibiotics

Details

When country is set, the parameter guideline will be ignored as these guidelines will be used:

- country = "nl": Rijksinstituut voor Volksgezondheid en Milieu "WIP-richtlijn BRMO (Bijzonder Resistente Micro-Organismen) [ZKH]" (link)

Please suggest your own country's specific guidelines by letting us know: https://gitlab.com/msberends/AMR/issues/new.

Other currently supported guidelines are:

- guideline = "eucast": EUCAST Expert Rules Version 3.1 "Intrinsic Resistance and Exceptional Phenotypes Tables" (link)
- guideline = "tb": World Health Organization "Companion handbook to the WHO guidelines for the programmatic management of drug-resistant tuberculosis" (link)

Value

Ordered factor with levels Negative < Positive, unconfirmed < Positive.

Antibiotics

To define antibiotics column names, leave as it is to determine it automatically with guess_ab_col or input a text (case-insensitive), or use NULL to skip a column (e.g. TIC = NULL to skip ticarcillin). Manually defined but non-existing columns will be skipped with a warning.

The following antibiotics are used for the functions eucast_rules and mdro. These are shown in the format '**antimicrobial ID**: name (*ATC code*)', sorted by name:

AMK: amikacin (J01GB06), **AMX**: amoxicillin (J01CA04), **AMC**: amoxicillin/clavulanic acid (J01CR02), **AMP**: ampicillin (J01CA01), **AZM**: azithromycin (J01FA10), **AZL**: azlocillin (J01CA09), **ATM**: aztreonam (J01DF01), **CAP**: capreomycin (J04AB30), **RID**: cefaloridine (J01DB02), **CZO**: cefazolin (J01DB04), **FEP**: cefepime (J01DE01), **CTX**: cefotaxime (J01DD01), **FOX**: cefoxitin (J01DC01), **CED**: cefradine (J01DB09), **CAZ**: ceftazidime (J01DD02), **CRO**: ceftriaxone (J01DD04), **CXM**: cefuroxime (J01DC02), **CHL**: chloramphenicol (J01BA01), **CIP**: ciprofloxacin (J01MA02),

CLR: clarithromycin (J01FA09), **CLI**: clindamycin (J01FF01), **COL**: colistin (J01XB01), **DAP**: daptomycin (J01XX09), **DOX**: doxycycline (J01AA02), **ETP**: ertapenem (J01DH03), **ERY**: erythromycin (J01FA01), **ETH**: ethambutol (J04AK02), **FLC**: flucloxacillin (J01CF05), **FOS**: fosfomycin (J01XX01), **FUS**: fusidic acid (J01XC01), **GAT**: gatifloxacin (J01MA16), **GEN**: gentamicin (J01GB03), **IPM**: imipenem (J01DH51), **INH**: isoniazid (J04AC01), **KAN**: kanamycin (J01GB04), **LVX**: levofloxacin (J01MA12), **LIN**: lincomycin (J01FF02), **LNZ**: linezolid (J01XX08), **MEM**: meropenem (J01DH02), **MTR**: metronidazole (J01XD01), **MEZ**: mezlocillin (J01CA10), **MNO**: minocycline (J01AA08), **MFX**: moxifloxacin (J01MA14), **NAL**: nalidixic acid (J01MB02), **NEO**: neomycin (J01GB05), **NET**: netilmicin (J01GB07), **NIT**: nitrofurantoin (J01XE01), **NOR**: norfloxacin (J01MA06), **NOV**: novobiocin (an ATCvet code: QJ01XX95), **OFX**: ofloxacin (J01MA01), **OXA**: oxacillin (J01CF04), **PEN**: penicillin G (J01CE01), **PIP**: piperacillin (J01CA12), **TZP**: piperacillin/tazobactam (J01CR05), **PLB**: polymyxin B (J01XB02), **PRI**: pristinamycin (J01FG01), **PZA**: pyrazinamide (J04AK01), **QDA**: quinupristin/dalfopristin (J01FG02), **RIB**: rifabutin (J04AB04), **RIF**: rifampicin (J04AB02), **RIF**: rifampin (J04AB02), **RFP**: rifapentine (J04AB05), **RXT**: roxithromycin (J01FA06), **SIS**: sisomicin (J01GB08), **TEC**: teicoplanin (J01XA02), **TCY**: tetracycline (J01AA07), **TIC**: ticarcillin (J01CA13), **TGC**: tigecycline (J01AA12), **TOB**: tobramycin (J01GB01), **TMP**: trimethoprim (J01EA01), **SXT**: trimethoprim/sulfamethoxazole (J01EE01), **VAN**: vancomycin (J01XA01).

Read more on our website!

On our website `https://msberends.gitlab.io/AMR` you can find a tutorial about how to conduct AMR analysis, the complete documentation of all functions (which reads a lot easier than here in R) and an example analysis using WHONET data.

Examples

```
library(dplyr)

septic_patients %>%
  mutate(EUCAST = mdro(.),
         BRMO = brmo(.))
```

`microorganisms` *Data set with ~65,000 microorganisms*

Description

A data set containing the microbial taxonomy of six kingdoms from the Catalogue of Life. MO codes can be looked up using `as.mo`.

Usage

```
microorganisms
```

Format

A `data.frame` with 67,906 observations and 16 variables:

`mo` ID of microorganism as used by this package

`col_id` Catalogue of Life ID

`fullname` Full name, like `"Escherichia coli"`

`kingdom`, `phylum`, `class`, `order`, `family`, `genus`, `species`, `subspecies` Taxonomic rank of the microorganism

`rank` Text of the taxonomic rank of the microorganism, like `"species"` or `"genus"`

`ref` Author(s) and year of concerning scientific publication

`species_id` ID of the species as used by the Catalogue of Life

`source` Either `"CoL"`, `"DSMZ"` (see source) or "manually added"

`prevalence` Prevalence of the microorganism, see `?as.mo`

Details

Manually added were:

- 9 entries of *Streptococcus* (beta haemolytic groups A, B, C, D, F, G, H, K and unspecified)
- 2 entries of *Staphylococcus* (coagulase-negative [CoNS] and coagulase-positive [CoPS])
- 3 entries of Trichomonas (Trichomonas vaginalis, and its family and genus)
- 3 other 'undefined' entries (unknown, unknown Gram negatives and unknown Gram positives)
- 8,830 species from the DSMZ (Deutsche Sammlung von Mikroorganismen und Zellkulturen) that are not in the Catalogue of Life

About the records from DSMZ (see source)

Names of prokaryotes are defined as being validly published by the International Code of Nomenclature of Bacteria. Validly published are all names which are included in the Approved Lists of Bacterial Names and the names subsequently published in the International Journal of Systematic Bacteriology (IJSB) and, from January 2000, in the International Journal of Systematic and Evolutionary Microbiology (IJSEM) as original articles or in the validation lists.

From: https://www.dsmz.de/support/bacterial-nomenclature-up-to-date-downloads/readme.html

Catalogue of Life

This package contains the complete taxonomic tree of almost all microorganisms (~65,000 species) from the authoritative and comprehensive Catalogue of Life (http://www.catalogueoflife.org). The Catalogue of Life is the most comprehensive and authoritative global index of species currently available.

Click here for more information about the included taxa. The Catalogue of Life releases updates annually; check which version was included in this package with `catalogue_of_life_version()`.

Read more on our website!

On our website https://msberends.gitlab.io/AMR you can find a tutorial about how to conduct AMR analysis, the complete documentation of all functions (which reads a lot easier than here in R) and an example analysis using WHONET data.

Source

Catalogue of Life: Annual Checklist (public online taxonomic database), http://www.catalogueoflife.org (check included annual version with catalogue_of_life_version()).

Leibniz Institute DSMZ-German Collection of Microorganisms and Cell Cultures, Germany, Prokaryotic Nomenclature Up-to-Date, http://www.dsmz.de/bacterial-diversity/prokaryotic-nomenclature-up-to-date (check included version with catalogue_of_life_version()).

See Also

as.mo, mo_property, microorganisms.codes

---

microorganisms.codes *Translation table for microorganism codes*

Description

A data set containing commonly used codes for microorganisms, from laboratory systems and WHONET. Define your own with set_mo_source.

Usage

```
microorganisms.codes
```

Format

A data.frame with 4,969 observations and 2 variables:

`certe` Commonly used code of a microorganism

`mo` ID of the microorganism in the microorganisms data set

Catalogue of Life

This package contains the complete taxonomic tree of almost all microorganisms (~65,000 species) from the authoritative and comprehensive Catalogue of Life (http://www.catalogueoflife.org). The Catalogue of Life is the most comprehensive and authoritative global index of species currently available.

Click here for more information about the included taxa. The Catalogue of Life releases updates annually; check which version was included in this package with catalogue_of_life_version().

Read more on our website!

On our website https://msberends.gitlab.io/AMR you can find a tutorial about how to conduct AMR analysis, the complete documentation of all functions (which reads a lot easier than here in R) and an example analysis using WHONET data.

See Also

as.mo microorganisms

---

microorganisms.old *Data set with previously accepted taxonomic names*

Description

A data set containing old (previously valid or accepted) taxonomic names according to the Catalogue of Life. This data set is used internally by as.mo.

Usage

```
microorganisms.old
```

Format

A data.frame with 21,342 observations and 4 variables:

`col_id` Catalogue of Life ID that was originally given

`col_id_new` New Catalogue of Life ID that responds to an entry in the microorganisms data set

`fullname` Old full taxonomic name of the microorganism

`ref` Author(s) and year of concerning scientific publication

Catalogue of Life

This package contains the complete taxonomic tree of almost all microorganisms (~65,000 species) from the authoritative and comprehensive Catalogue of Life (http://www.catalogueoflife.org). The Catalogue of Life is the most comprehensive and authoritative global index of species currently available.

Click here for more information about the included taxa. The Catalogue of Life releases updates annually; check which version was included in this package with catalogue_of_life_version().

Read more on our website!

On our website https://msberends.gitlab.io/AMR you can find a tutorial about how to conduct AMR analysis, the complete documentation of all functions (which reads a lot easier than here in R) and an example analysis using WHONET data.

Source

Catalogue of Life: Annual Checklist (public online taxonomic database), http://www.catalogueoflife.org (check included annual version with catalogue_of_life_version()).

See Also

as.mo mo_property microorganisms

---

mo_property *Property of a microorganism*

Description

Use these functions to return a specific property of a microorganism from the microorganisms data set. All input values will be evaluated internally with as.mo.

Usage

```
mo_name(x, language = get_locale(), ...)

mo_fullname(x, language = get_locale(), ...)

mo_shortname(x, language = get_locale(), ...)

mo_subspecies(x, language = get_locale(), ...)

mo_species(x, language = get_locale(), ...)

mo_genus(x, language = get_locale(), ...)

mo_family(x, language = get_locale(), ...)

mo_order(x, language = get_locale(), ...)

mo_class(x, language = get_locale(), ...)

mo_phylum(x, language = get_locale(), ...)

mo_kingdom(x, language = get_locale(), ...)

mo_type(x, language = get_locale(), ...)

mo_gramstain(x, language = get_locale(), ...)

mo_ref(x, ...)

mo_authors(x, ...)
```

```
mo_year(x, ...)

mo_rank(x, ...)

mo_taxonomy(x, language = get_locale(), ...)

mo_synonyms(x, ...)

mo_info(x, language = get_locale(), ...)

mo_url(x, open = FALSE, ...)

mo_property(x, property = "fullname", language = get_locale(), ...)
```

Arguments

x	any (vector of) text that can be coerced to a valid microorganism code with as.mo
language	language of the returned text, defaults to system language (see get_locale) and can also be set with getOption("AMR_locale"). Use language = NULL or language = "" to prevent translation.
...	other parameters passed on to as.mo
open	browse the URL using browseURL()
property	one of the column names of the microorganisms data set or "shortname"

Details

All functions will return the most recently known taxonomic property according to the Catalogue of Life, except for mo_ref, mo_authors and mo_year. This leads to the following results:

- mo_name("Chlamydia psittaci") will return "Chlamydophila psittaci" (with a warning about the renaming)
- mo_ref("Chlamydia psittaci") will return "Page, 1968" (with a warning about the renaming)
- mo_ref("Chlamydophila psittaci") will return "Everett et al., 1999" (without a warning)

The Gram stain - mo_gramstain() - will be determined on the taxonomic kingdom and phylum. According to Cavalier-Smith (2002) who defined subkingdoms Negibacteria and Posibacteria, only these phyla are Posibacteria: Actinobacteria, Chloroflexi, Firmicutes and Tenericutes. These bacteria are considered Gram positive - all other bacteria are considered Gram negative. Species outside the kingdom of Bacteria will return a value NA.

All output will be translated where possible.

The function mo_url() will return the direct URL to the online database entry, which also shows the scientific reference of the concerned species.

Value

- An `integer` in case of `mo_year`
- A `list` in case of `mo_taxonomy`
- A named `character` in case of `mo_url`
- A `character` in all other cases

Catalogue of Life

This package contains the complete taxonomic tree of almost all microorganisms (~65,000 species) from the authoritative and comprehensive Catalogue of Life (http://www.catalogueoflife.org). The Catalogue of Life is the most comprehensive and authoritative global index of species currently available.

Click here for more information about the included taxa. The Catalogue of Life releases updates annually; check which version was included in this package with `catalogue_of_life_version()`.

Source

[1] Becker K *et al.* **Coagulase-Negative Staphylococci**. 2014. Clin Microbiol Rev. 27(4): 870–926. https://dx.doi.org/10.1128/CMR.00109-13

[2] Becker K *et al.* **Implications of identifying the recently defined members of the *S. aureus* complex, *S. argenteus* and *S. schweitzeri*: A position paper of members of the ESCMID Study Group for staphylococci and Staphylococcal Diseases (ESGS).** 2019. Clin Microbiol Infect. https://doi.org/10.1016/j.cmi.2019.02.028

[3] Lancefield RC **A serological differentiation of human and other groups of hemolytic streptococci**. 1933. J Exp Med. 57(4): 571–95. https://dx.doi.org/10.1084/jem.57.4.571

[4] Catalogue of Life: Annual Checklist (public online taxonomic database), http://www.catalogueoflife.org (check included annual version with `catalogue_of_life_version()`).

Read more on our website!

On our website https://msberends.gitlab.io/AMR you can find a tutorial about how to conduct AMR analysis, the complete documentation of all functions (which reads a lot easier than here in R) and an example analysis using WHONET data.

See Also

`microorganisms`

Examples

```
## taxonomic tree
mo_kingdom("E. coli")        # "Bacteria"
mo_phylum("E. coli")         # "Proteobacteria"
mo_class("E. coli")          # "Gammaproteobacteria"
mo_order("E. coli")          # "Enterobacteriales"
mo_family("E. coli")         # "Enterobacteriaceae"
mo_genus("E. coli")          # "Escherichia"
```

```
mo_species("E. coli")         # "coli"
mo_subspecies("E. coli")      # ""

## colloquial properties
mo_name("E. coli")            # "Escherichia coli"
mo_fullname("E. coli")        # "Escherichia coli", same as mo_name()
mo_shortname("E. coli")       # "E. coli"

## other properties
mo_gramstain("E. coli")       # "Gram-negative"
mo_type("E. coli")            # "Bacteria" (equal to kingdom, but may be translated)
mo_rank("E. coli")            # "species"
mo_url("E. coli")             # get the direct url to the online database entry
mo_synonyms("E. coli")        # get previously accepted taxonomic names

## scientific reference
mo_ref("E. coli")             # "Castellani et al., 1919"
mo_authors("E. coli")         # "Castellani et al."
mo_year("E. coli")            # 1919


# Abbreviations known in the field
mo_genus("MRSA")              # "Staphylococcus"
mo_species("MRSA")            # "aureus"
mo_shortname("MRSA")          # "S. aureus"
mo_gramstain("MRSA")          # "Gram-positive"

mo_genus("VISA")              # "Staphylococcus"
mo_species("VISA")            # "aureus"


# Known subspecies
mo_name("doylei")             # "Campylobacter jejuni doylei"
mo_genus("doylei")            # "Campylobacter"
mo_species("doylei")          # "jejuni"
mo_subspecies("doylei")       # "doylei"

mo_fullname("K. pneu rh")     # "Klebsiella pneumoniae rhinoscleromatis"
mo_shortname("K. pneu rh")    # "K. pneumoniae"


# Becker classification, see ?as.mo
mo_fullname("S. epi")                     # "Staphylococcus epidermidis"
mo_fullname("S. epi", Becker = TRUE)      # "Coagulase-negative Staphylococcus (CoNS)"
mo_shortname("S. epi")                    # "S. epidermidis"
mo_shortname("S. epi", Becker = TRUE)     # "CoNS"

# Lancefield classification, see ?as.mo
mo_fullname("S. pyo")                     # "Streptococcus pyogenes"
mo_fullname("S. pyo", Lancefield = TRUE)  # "Streptococcus group A"
mo_shortname("S. pyo")                    # "S. pyogenes"
mo_shortname("S. pyo", Lancefield = TRUE) # "GAS" ('Group A streptococci')
```

```
# language support for German, Dutch, Spanish, Portuguese, Italian and French
mo_gramstain("E. coli", language = "de")  # "Gramnegativ"
mo_gramstain("E. coli", language = "nl")  # "Gram-negatief"
mo_gramstain("E. coli", language = "es")  # "Gram negativo"

# mo_type is equal to mo_kingdom, but mo_kingdom will remain official
mo_kingdom("E. coli")                     # "Bacteria" on a German system
mo_type("E. coli")                        # "Bakterien" on a German system
mo_type("E. coli")                        # "Bacteria" on an English system

mo_fullname("S. pyogenes",
            Lancefield = TRUE,
            language = "de")              # "Streptococcus Gruppe A"
mo_fullname("S. pyogenes",
            Lancefield = TRUE,
            language = "nl")              # "Streptococcus groep A"


# get a list with the complete taxonomy (from kingdom to subspecies)
mo_taxonomy("E. coli")
# get a list with the taxonomy, the authors and the URL to the online database
mo_info("E. coli")
```

`mo_source` *Use predefined reference data set*

Description

These functions can be used to predefine your own reference to be used in `as.mo` and consequently all `mo_*` functions like `mo_genus` and `mo_gramstain`.

This is **the fastest way** to have your organisation (or analysis) specific codes picked up and translated by this package.

Usage

```
set_mo_source(path)

get_mo_source()
```

Arguments

`path`	location of your reference file, see Details

Details

The reference file can be a text file seperated with commas (CSV) or tabs or pipes, an Excel file (either 'xls' or 'xlsx' format) or an R object file (extension '.rds'). To use an Excel file, you need to have the `readxl` package installed.

set_mo_source will check the file for validity: it must be a data.frame, must have a column named "mo" which contains values from microorganisms$mo and must have a reference column with your own defined values. If all tests pass, set_mo_source will read the file into R and export it to "~/.mo_source.rds". This compressed data file will then be used at default for MO determination (function as.mo and consequently all mo_* functions like mo_genus and mo_gramstain). The location of the original file will be saved as option with options(mo_source = path). Its timestamp will be saved with options(mo_source_datetime = ...).

get_mo_source will return the data set by reading "~/.mo_source.rds" with readRDS. If the original file has changed (the file defined with path), it will call set_mo_source to update the data file automatically.

Reading an Excel file (.xlsx) with only one row has a size of 8-9 kB. The compressed file used by this package will have a size of 0.1 kB and can be read by get_mo_source in only a couple of microseconds (a millionth of a second).

How it works

Imagine this data on a sheet of an Excel file (mo codes were looked up in the 'microorganisms' data set). The first column contains the organisation specific codes, the second column contains an MO code from this package:

```
  |         A          |      B      |
--|--------------------|-------------|
1 | Organisation XYZ   | mo          |
2 | lab_mo_ecoli       | B_ESCHR_COL |
3 | lab_mo_kpneumoniae | B_KLBSL_PNE |
4 |                    |             |
```

We save it as 'home/me/ourcodes.xlsx'. Now we have to set it as a source:

```
set_mo_source("home/me/ourcodes.xlsx")
# Created mo_source file '~/.mo_source.rds' from 'home/me/ourcodes.xlsx'.
```

It has now created a file "~/.mo_source.rds" with the contents of our Excel file, but only the first column with foreign values and the 'mo' column will be kept.

And now we can use it in our functions:

```
as.mo("lab_mo_ecoli")
[1] B_ESCHR_COL

mo_genus("lab_mo_kpneumoniae")
[1] "Klebsiella"

# other input values still work too
as.mo(c("Escherichia coli", "E. coli", "lab_mo_ecoli"))
[1] B_ESCHR_COL  B_ESCHR_COL  B_ESCHR_COL
```

If we edit the Excel file to, let's say, this:

```
  |         A          |      B      |
--|--------------------|-------------|
1 | Organisation XYZ   | mo          |
2 | lab_mo_ecoli       | B_ESCHR_COL |
3 | lab_mo_kpneumoniae | B_KLBSL_PNE |
4 | lab_Staph_aureus   | B_STPHY_AUR |
5 |                    |             |
```

...any new usage of an MO function in this package will update your data:

```
as.mo("lab_mo_ecoli")
# Updated mo_source file '~/.mo_source.rds' from 'home/me/ourcodes.xlsx'.
[1] B_ESCHR_COL

mo_genus("lab_Staph_aureus")
[1] "Staphylococcus"
```

To remove the reference completely, just use any of these:

```
set_mo_source("")
set_mo_source(NULL)
# Removed mo_source file '~/.mo_source.rds'.
```

Read more on our website!

On our website https://msberends.gitlab.io/AMR you can find a tutorial about how to conduct AMR analysis, the complete documentation of all functions (which reads a lot easier than here in R) and an example analysis using WHONET data.

`p.symbol` *Symbol of a p value*

Description

Return the symbol related to the p value: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1. Values above `p = 1` will return `NA`.

Usage

```
p.symbol(p, emptychar = " ")
```

Arguments

`p`	p value
`emptychar`	text to show when `p > 0.1`

Value

Text

Read more on our website!

On our website https://msberends.gitlab.io/AMR you can find a tutorial about how to conduct AMR analysis, the complete documentation of all functions (which reads a lot easier than here in R) and an example analysis using WHONET data.

`portion` *Calculate resistance of isolates*

Description

These functions can be used to calculate the (co-)resistance of microbial isolates (i.e. percentage of S, SI, I, IR or R). All functions support quasiquotation with pipes, can be used in `dplyrs` `summarise` and support grouped variables, see *Examples*.

`portion_R` and `portion_IR` can be used to calculate resistance, `portion_S` and `portion_SI` can be used to calculate susceptibility.

Usage

```
portion_R(..., minimum = 30, as_percent = FALSE,
  also_single_tested = FALSE)

portion_IR(..., minimum = 30, as_percent = FALSE,
  also_single_tested = FALSE)

portion_I(..., minimum = 30, as_percent = FALSE,
  also_single_tested = FALSE)

portion_SI(..., minimum = 30, as_percent = FALSE,
  also_single_tested = FALSE)

portion_S(..., minimum = 30, as_percent = FALSE,
  also_single_tested = FALSE)

portion_df(data, translate_ab = "name", language = get_locale(),
  minimum = 30, as_percent = FALSE, combine_SI = TRUE,
  combine_IR = FALSE)

rsi_df(data, translate_ab = "name", language = get_locale(),
  minimum = 30, as_percent = FALSE, combine_SI = TRUE,
  combine_IR = FALSE)
```

Arguments

...	one or more vectors (or columns) with antibiotic interpretations. They will be transformed internally with as.rsi if needed. Use multiple columns to calculate (the lack of) co-resistance: the probability where one of two drugs have a resistant or susceptible result. See Examples.
minimum	the minimum allowed number of available (tested) isolates. Any isolate count lower than minimum will return NA with a warning. The default number of 30 isolates is advised by the Clinical and Laboratory Standards Institute (CLSI) as best practice, see Source.
as_percent	a logical to indicate whether the output must be returned as a hundred fold with % sign (a character). A value of 0.123456 will then be returned as "12.3%".
also_single_tested	a logical to indicate whether (in combination therapies) also observations should be included where not all antibiotics were tested, but at least one of the tested antibiotics contains a target interpretation (e.g. S in case of portion_S and R in case of portion_R). **This would lead to selection bias in almost all cases.**
data	a data.frame containing columns with class rsi (see as.rsi)
translate_ab	a column name of the antibiotics data set to translate the antibiotic abbreviations to, using ab_property
language	language of the returned text, defaults to system language (see get_locale) and can also be set with getOption("AMR_locale"). Use language = NULL or language = "" to prevent translation.
combine_SI	a logical to indicate whether all values of S and I must be merged into one, so the output only consists of S+I vs. R (susceptible vs. resistant). This used to be the parameter combine_IR, but this now follows the redefinition by EUCAST about the interpretion of I (increased exposure) in 2019, see section 'Interpretation of S, I and R' below. Default is TRUE.
combine_IR	a logical to indicate whether all values of I and R must be merged into one, so the output only consists of S vs. I+R (susceptible vs. non-susceptible). This is outdated, see parameter combine_SI.

Details

Remember that you should filter your table to let it contain only first isolates! Use first_isolate to determine them in your data set.

These functions are not meant to count isolates, but to calculate the portion of resistance/susceptibility. Use the count functions to count isolates. *Low counts can influence the outcome - these* portion *functions may camouflage this, since they only return the portion albeit being dependent on the* minimum *parameter.*

The function portion_df takes any variable from data that has an "rsi" class (created with as.rsi) and calculates the portions R, I and S. The resulting *tidy data* (see Source) data.frame will have three rows (S/I/R) and a column for each group and each variable with class "rsi".

The function rsi_df works exactly like portion_df, but adds the number of isolates.

Value

Double or, when `as_percent = TRUE`, a character.

Interpretation of S, I and R

In 2019, EUCAST has decided to change the definitions of susceptibility testing categories S, I and R as shown below. Results of several consultations on the new definitions are available on the EUCAST website under "Consultations".

- **S** - Susceptible, standard dosing regimen: A microorganism is categorised as "Susceptible, standard dosing regimen", when there is a high likelihood of therapeutic success using a standard dosing regimen of the agent.
- **I** - Susceptible, increased exposure: A microorganism is categorised as "Susceptible, Increased exposure" when there is a high likelihood of therapeutic success because exposure to the agent is increased by adjusting the dosing regimen or by its concentration at the site of infection.
- **R** - Resistant: A microorganism is categorised as "Resistant" when there is a high likelihood of therapeutic failure even when there is increased exposure.

Exposure is a function of how the mode of administration, dose, dosing interval, infusion time, as well as distribution and excretion of the antimicrobial agent will influence the infecting organism at the site of infection.

Source: http://www.eucast.org/newsiandr/.

This AMR package honours this new insight.

Read more on our website!

On our website https://msberends.gitlab.io/AMR you can find a tutorial about how to conduct AMR analysis, the complete documentation of all functions (which reads a lot easier than here in R) and an example analysis using WHONET data.

Source

M39 Analysis and Presentation of Cumulative Antimicrobial Susceptibility Test Data, 4th Edition, 2014, *Clinical and Laboratory Standards Institute (CLSI)*. https://clsi.org/standards/products/microbiology/documents/m39/.

Wickham H. **Tidy Data.** The Journal of Statistical Software, vol. 59, 2014. http://vita.had.co.nz/papers/tidy-data.html

See Also

`count_*` to count resistant and susceptible isolates.

Examples

```
# septic_patients is a data set available in the AMR package. It is true, genuine data.
?septic_patients

# Calculate resistance
portion_R(septic_patients$AMX)
portion_IR(septic_patients$AMX)

# Or susceptibility
portion_S(septic_patients$AMX)
portion_SI(septic_patients$AMX)

# Do the above with pipes:
library(dplyr)
septic_patients %>% portion_R(AMX)
septic_patients %>% portion_IR(AMX)
septic_patients %>% portion_S(AMX)
septic_patients %>% portion_SI(AMX)

septic_patients %>%
  group_by(hospital_id) %>%
  summarise(p = portion_S(CIP),
            n = n_rsi(CIP)) # n_rsi works like n_distinct in dplyr

septic_patients %>%
  group_by(hospital_id) %>%
  summarise(R = portion_R(CIP, as_percent = TRUE),
            I = portion_I(CIP, as_percent = TRUE),
            S = portion_S(CIP, as_percent = TRUE),
            n1 = count_all(CIP),  # the actual total; sum of all three
            n2 = n_rsi(CIP),      # same - analogous to n_distinct
            total = n())          # NOT the number of tested isolates!

# Calculate co-resistance between amoxicillin/clav acid and gentamicin,
# so we can see that combination therapy does a lot more than mono therapy:
septic_patients %>% portion_S(AMC)       # S = 71.4%
septic_patients %>% count_all(AMC)       # n = 1879

septic_patients %>% portion_S(GEN)       # S = 74.0%
septic_patients %>% count_all(GEN)       # n = 1855

septic_patients %>% portion_S(AMC, GEN)  # S = 92.3%
septic_patients %>% count_all(AMC, GEN)  # n = 1798


septic_patients %>%
  group_by(hospital_id) %>%
  summarise(cipro_p = portion_S(CIP, as_percent = TRUE),
            cipro_n = count_all(CIP),
            genta_p = portion_S(GEN, as_percent = TRUE),
            genta_n = count_all(GEN),
            combination_p = portion_S(CIP, GEN, as_percent = TRUE),
```

```
            combination_n = count_all(CIP, GEN))

# Get portions S/I/R immediately of all rsi columns
septic_patients %>%
  select(AMX, CIP) %>%
  portion_df(translate = FALSE)

# It also supports grouping variables
septic_patients %>%
  select(hospital_id, AMX, CIP) %>%
  group_by(hospital_id) %>%
  portion_df(translate = FALSE)


## Not run:

# calculate current empiric combination therapy of Helicobacter gastritis:
my_table %>%
  filter(first_isolate == TRUE,
         genus == "Helicobacter") %>%
  summarise(p = portion_S(AMX, MTR),  # amoxicillin with metronidazole
            n = count_all(AMX, MTR))

## End(Not run)
```

`read.4D` *Read data from 4D database*

Description

This function is only useful for the MMB department of the UMCG. Use this function to **import data by just defining the `file` parameter**. It will automatically transform birth dates and calculate patients age, translate the column names to English, transform the MO codes with `as.mo` and transform all antimicrobial columns with `as.rsi`.

Usage

```
read.4D(file, info = interactive(), header = TRUE, row.names = NULL,
  sep = "\t", quote = "\"'", dec = ",", na.strings = c("NA", "",
  "."), skip = 2, check.names = TRUE, strip.white = TRUE,
  fill = TRUE, blank.lines.skip = TRUE, stringsAsFactors = FALSE,
  fileEncoding = "UTF-8", encoding = "UTF-8")
```

Arguments

`file`	the name of the file which the data are to be read from. Each row of the table appears as one line of the file. If it does not contain an *absolute* path, the file name is *relative* to the current working directory, `getwd()`. Tilde-expansion is performed where supported. This can be a compressed file (see `file`).

Alternatively, file can be a readable text-mode connection (which will be opened for reading if necessary, and if so closed (and hence destroyed) at the end of the function call). (If stdin() is used, the prompts for lines may be somewhat confusing. Terminate input with a blank line or an EOF signal, Ctrl-D on Unix and Ctrl-Z on Windows. Any pushback on stdin() will be cleared before return.)

file can also be a complete URL. (For the supported URL schemes, see the 'URLs' section of the help for url.)

info — a logical to indicate whether info about the import should be printed, defaults to TRUE in interactive sessions

header — a logical value indicating whether the file contains the names of the variables as its first line. If missing, the value is determined from the file format: header is set to TRUE if and only if the first row contains one fewer field than the number of columns.

row.names — a vector of row names. This can be a vector giving the actual row names, or a single number giving the column of the table which contains the row names, or character string giving the name of the table column containing the row names.

If there is a header and the first row contains one fewer field than the number of columns, the first column in the input is used for the row names. Otherwise if row.names is missing, the rows are numbered.

Using row.names = NULL forces row numbering. Missing or NULL row.names generate row names that are considered to be 'automatic' (and not preserved by as.matrix).

sep — the field separator character. Values on each line of the file are separated by this character. If sep = "" (the default for read.table) the separator is 'white space', that is one or more spaces, tabs, newlines or carriage returns.

quote — the set of quoting characters. To disable quoting altogether, use quote = "". See scan for the behaviour on quotes embedded in quotes. Quoting is only considered for columns read as character, which is all of them unless colClasses is specified.

dec — the character used in the file for decimal points.

na.strings — a character vector of strings which are to be interpreted as NA values. Blank fields are also considered to be missing values in logical, integer, numeric and complex fields. Note that the test happens *after* white space is stripped from the input, so na.strings values may need their own white space stripped in advance.

skip — integer: the number of lines of the data file to skip before beginning to read data.

check.names — logical. If TRUE then the names of the variables in the data frame are checked to ensure that they are syntactically valid variable names. If necessary they are adjusted (by make.names) so that they are, and also to ensure that there are no duplicates.

strip.white — logical. Used only when sep has been specified, and allows the stripping of leading and trailing white space from unquoted character fields (numeric fields are always stripped). See scan for further details (including the exact meaning of 'white space'), remembering that the columns may include the row names.

`fill` logical. If `TRUE` then in case the rows have unequal length, blank fields are implicitly added. See 'Details'.

`blank.lines.skip` logical: if `TRUE` blank lines in the input are ignored.

`stringsAsFactors` logical: should character vectors be converted to factors? Note that this is overridden by `as.is` and `colClasses`, both of which allow finer control.

`fileEncoding` character string: if non-empty declares the encoding used on a file (not a connection) so the character data can be re-encoded. See the 'Encoding' section of the help for `file`, the 'R Data Import/Export Manual' and 'Note'.

`encoding` encoding to be assumed for input strings. It is used to mark character strings as known to be in Latin-1 or UTF-8 (see `Encoding`): it is not used to re-encode the input, but allows R to handle encoded strings in their native encoding (if one of those two). See 'Value' and 'Note'.

Details

Column names will be transformed, but the original column names are set as a "label" attribute and can be seen in e.g. RStudio Viewer.

Read more on our website!

On our website `https://msberends.gitlab.io/AMR` you can find a tutorial about how to conduct AMR analysis, the complete documentation of all functions (which reads a lot easier than here in R) and an example analysis using WHONET data.

`resistance_predict` *Predict antimicrobial resistance*

Description

Create a prediction model to predict antimicrobial resistance for the next years on statistical solid ground. Standard errors (SE) will be returned as columns `se_min` and `se_max`. See Examples for a real live example.

Usage

```
resistance_predict(x, col_ab, col_date = NULL, year_min = NULL,
  year_max = NULL, year_every = 1, minimum = 30,
  model = "binomial", I_as_S = TRUE, preserve_measurements = TRUE,
  info = TRUE, ...)

rsi_predict(x, col_ab, col_date = NULL, year_min = NULL,
  year_max = NULL, year_every = 1, minimum = 30,
  model = "binomial", I_as_S = TRUE, preserve_measurements = TRUE,
  info = TRUE, ...)
```

```
## S3 method for class 'resistance_predict'
plot(x,
  main = paste("Resistance Prediction of", x_name), ...)

ggplot_rsi_predict(x, main = paste("Resistance Prediction of", x_name),
  ribbon = TRUE, ...)
```

Arguments

x	a data.frame containing isolates.
col_ab	column name of x with antimicrobial interpretations (R, I and S)
col_date	column name of the date, will be used to calculate years if this column doesn't consist of years already, defaults to the first column of with a date class
year_min	lowest year to use in the prediction model, dafaults to the lowest year in col_date
year_max	highest year to use in the prediction model, defaults to 10 years after today
year_every	unit of sequence between lowest year found in the data and year_max
minimum	minimal amount of available isolates per year to include. Years containing less observations will be estimated by the model.
model	the statistical model of choice. Defaults to a generalised linear regression model with binomial distribution, assuming that a period of zero resistance was followed by a period of increasing resistance leading slowly to more and more resistance. See Details for valid options.
I_as_S	a logical to indicate whether values I should be treated as S
preserve_measurements	a logical to indicate whether predictions of years that are actually available in the data should be overwritten by the original data. The standard errors of those years will be NA.
info	a logical to indicate whether textual analysis should be printed with the name and summary of the statistical model.
...	parameters passed on to functions
main	title of the plot
ribbon	a logical to indicate whether a ribbon should be shown (default) or error bars

Details

Valid options for the statistical model are:

- "binomial" or "binom" or "logit": a generalised linear regression model with binomial distribution
- "loglin" or "poisson": a generalised log-linear regression model with poisson distribution
- "lin" or "linear": a linear regression model

Value

`data.frame` with extra class `"resistance_predict"` with columns:

- `year`
- `value`, the same as `estimated` when `preserve_measurements = FALSE`, and a combination of `observed` and `estimated` otherwise
- `se_min`, the lower bound of the standard error with a minimum of `0` (so the standard error will never go below 0%)
- `se_max` the upper bound of the standard error with a maximum of `1` (so the standard error will never go above 100%)
- `observations`, the total number of available observations in that year, i.e. S + I + R
- `observed`, the original observed resistant percentages
- `estimated`, the estimated resistant percentages, calculated by the model

Furthermore, the model itself is available as an attribute: `attributes(x)$model`, see Examples.

Read more on our website!

On our website https://msberends.gitlab.io/AMR you can find a tutorial about how to conduct AMR analysis, the complete documentation of all functions (which reads a lot easier than here in R) and an example analysis using WHONET data.

See Also

The `portion` function to calculate resistance,
`lm` `glm`

Examples

```
x <- resistance_predict(septic_patients, col_ab = "AMX", year_min = 2010)
plot(x)
ggplot_rsi_predict(x)

# use dplyr so you can actually read it:
library(dplyr)
x <- septic_patients %>%
  filter_first_isolate() %>%
  filter(mo_genus(mo) == "Staphylococcus") %>%
  resistance_predict("PEN")
plot(x)


# get the model from the object
mymodel <- attributes(x)$model
summary(mymodel)


# create nice plots with ggplot2 yourself
if (!require(ggplot2)) {
```

```
  data <- septic_patients %>%
    filter(mo == as.mo("E. coli")) %>%
    resistance_predict(col_ab = "AMX",
                       col_date = "date",
                       info = FALSE,
                       minimum = 15)

  ggplot(data,
         aes(x = year)) +
    geom_col(aes(y = value),
             fill = "grey75") +
    geom_errorbar(aes(ymin = se_min,
                      ymax = se_max),
                  colour = "grey50") +
    scale_y_continuous(limits = c(0, 1),
                       breaks = seq(0, 1, 0.1),
                       labels = paste0(seq(0, 100, 10), "%")) +
    labs(title = expression(paste("Forecast of amoxicillin resistance in ",
                                  italic("E. coli"))),
         y = "%IR",
         x = "Year") +
    theme_minimal(base_size = 13)
}
```

`rsi_translation` *Data set for RSI interpretation*

Description

Data set to interpret MIC and disk diffusion to RSI values. Included guidelines are CLSI (2011-2019) and EUCAST (2011-2019). Use `as.rsi` to transform MICs or disks measurements to RSI values.

Usage

```
rsi_translation
```

Format

A `data.frame` with 11,559 observations and 9 variables:

`guideline` Name of the guideline

`mo` Microbial ID, see `as.mo`

`ab` Antibiotic ID, see `as.ab`

`ref_tbl` Info about where the guideline rule can be found

`S_mic` Lowest MIC value that leads to "S"

`R_mic` Highest MIC value that leads to "R"

`dose_disk` Dose of the used disk diffusion method

`S_disk` Lowest number of millimeters that leads to "S"

`R_disk` Highest number of millimeters that leads to "R"

Read more on our website!

On our website https://msberends.gitlab.io/AMR you can find a tutorial about how to conduct AMR analysis, the complete documentation of all functions (which reads a lot easier than here in R) and an example analysis using WHONET data.

`septic_patients` *Data set with 2,000 blood culture isolates from septic patients*

Description

An anonymised data set containing 2,000 microbial blood culture isolates with their full antibiograms found in septic patients in 4 different hospitals in the Netherlands, between 2001 and 2017. It is true, genuine data. This `data.frame` can be used to practice AMR analysis. For examples, please read the tutorial on our website.

Usage

```
septic_patients
```

Format

A `data.frame` with 2,000 observations and 49 variables:

`date` date of receipt at the laboratory

`hospital_id` ID of the hospital, from A to D

`ward_icu` logical to determine if ward is an intensive care unit

`ward_clinical` logical to determine if ward is a regular clinical ward

`ward_outpatient` logical to determine if ward is an outpatient clinic

`age` age of the patient

`gender` gender of the patient

`patient_id` ID of the patient, first 10 characters of an SHA hash containing irretrievable information

`mo` ID of microorganism created with `as.mo`, see also `microorganisms`

`peni:rifa` 40 different antibiotics with class `rsi` (see `as.rsi`); these column names occur in `antibiotics` data set and can be translated with `abname`

Read more on our website!

On our website https://msberends.gitlab.io/AMR you can find a tutorial about how to conduct AMR analysis, the complete documentation of all functions (which reads a lot easier than here in R) and an example analysis using WHONET data.

skewness *Skewness of the sample*

Description

Skewness is a measure of the asymmetry of the probability distribution of a real-valued random variable about its mean.

When negative: the left tail is longer; the mass of the distribution is concentrated on the right of the figure. When positive: the right tail is longer; the mass of the distribution is concentrated on the left of the figure.

Usage

```
skewness(x, na.rm = FALSE)

## Default S3 method:
skewness(x, na.rm = FALSE)

## S3 method for class 'matrix'
skewness(x, na.rm = FALSE)

## S3 method for class 'data.frame'
skewness(x, na.rm = FALSE)
```

Arguments

x	a vector of values, a matrix or a data frame
na.rm	a logical value indicating whether NA values should be stripped before the computation proceeds.

Read more on our website!

On our website https://msberends.gitlab.io/AMR you can find a tutorial about how to conduct AMR analysis, the complete documentation of all functions (which reads a lot easier than here in R) and an example analysis using WHONET data.

See Also

kurtosis

translate *Translate strings from AMR package*

Description

For language-dependent output of AMR functions, like mo_fullname and mo_type.

Usage

```
get_locale()
```

Details

Strings will be translated to foreign languages if they are defined in a local translation file. Additions to this file can be suggested at our repository. The file can be found here: https://gitlab.com/msberends/AMR/blob/master/data-raw/translations.tsv.

Please suggest your own translations by creating a new issue on our repository.

This file will be read by all functions where a translated output can be desired, like all mo_property functions (mo_fullname, mo_type, etc.).

The system language will be used at default, if supported, using get_locale. The system language can be overwritten with getOption("AMR_locale").

Read more on our website!

On our website https://msberends.gitlab.io/AMR you can find a tutorial about how to conduct AMR analysis, the complete documentation of all functions (which reads a lot easier than here in R) and an example analysis using WHONET data.

Examples

```
# The 'language' parameter of below functions
# will be set automatically to your system language
# with get_locale()

# English
mo_fullname("CoNS", language = "en")
#> "Coagulase-negative Staphylococcus (CoNS)"

# German
mo_fullname("CoNS", language = "de")
#> "Koagulase-negative Staphylococcus (KNS)"

# Dutch
mo_fullname("CoNS", language = "nl")
#> "Coagulase-negatieve Staphylococcus (CNS)"

# Spanish
mo_fullname("CoNS", language = "es")
```

```
#> "Staphylococcus coagulasa negativo (SCN)"

# Italian
mo_fullname("CoNS", language = "it")
#> "Staphylococcus negativo coagulasi (CoNS)"

# Portuguese
mo_fullname("CoNS", language = "pt")
#> "Staphylococcus coagulase negativo (CoNS)"
```

WHOCC *WHOCC: WHO Collaborating Centre for Drug Statistics Methodology*

Description

All antimicrobial drugs and their official names, ATC codes, ATC groups and defined daily dose (DDD) are included in this package, using the WHO Collaborating Centre for Drug Statistics Methodology.

WHOCC

This package contains **all ~450 antimicrobial drugs** and their Anatomical Therapeutic Chemical (ATC) codes, ATC groups and Defined Daily Dose (DDD) from the World Health Organization Collaborating Centre for Drug Statistics Methodology (WHOCC, https://www.whocc.no) and the Pharmaceuticals Community Register of the European Commission (http://ec.europa.eu/health/documents/community-register/html/atc.htm).

These have become the gold standard for international drug utilisation monitoring and research.

The WHOCC is located in Oslo at the Norwegian Institute of Public Health and funded by the Norwegian government. The European Commission is the executive of the European Union and promotes its general interest.

Read more on our website!

On our website https://msberends.gitlab.io/AMR you can find a tutorial about how to conduct AMR analysis, the complete documentation of all functions (which reads a lot easier than here in R) and an example analysis using WHONET data.

Examples

```
as.ab("meropenem")
ab_name("J01DH02")

ab_tradenames("flucloxacillin")
```

WHONET *Data set with 500 isolates - WHONET example*

Description

This example data set has the exact same structure as an export file from WHONET. Such files can be used with this package, as this example data set shows. The data itself was based on our `septic_patients` data set.

Usage

```
WHONET
```

Format

A `data.frame` with 500 observations and 53 variables:

`Identification number` ID of the sample

`Specimen number` ID of the specimen

`Organism` Name of the microorganism. Before analysis, you should transform this to a valid microbial class, using `as.mo`.

`Country` Country of origin

`Laboratory` Name of laboratory

`Last name` Last name of patient

`First name` Initial of patient

`Sex` Gender of patient

`Age` Age of patient

`Age category` Age group, can also be looked up using `age_groups`

`Date of admission` Date of hospital admission

`Specimen date` Date when specimen was received at laboratory

`Specimen type` Specimen type or group

`Specimen type (Numeric)` Translation of `"Specimen type"`

`Reason` Reason of request with Differential Diagnosis

`Isolate number` ID of isolate

`Organism type` Type of microorganism, can also be looked up using `mo_type`

`Serotype` Serotype of microorganism

`Beta-lactamase` Microorganism produces beta-lactamase?

`ESBL` Microorganism produces extended spectrum beta-lactamase?

`Carbapenemase` Microorganism produces carbapenemase?

`MRSA screening test` Microorganism is possible MRSA?

`Inducible clindamycin resistance` Clindamycin can be induced?

`Comment` Other comments

`Date of data entry` Date this data was entered in WHONET

`AMP_ND10:CIP_EE` 27 different antibiotics. You can lookup the abbreviatons in the `antibiotics` data set, or use e.g. `atc_name("AMP")` to get the official name immediately. Before analysis, you should transform this to a valid antibiotic class, using `as.rsi`.

Read more on our website!

On our website `https://msberends.gitlab.io/AMR` you can find a tutorial about how to conduct AMR analysis, the complete documentation of all functions (which reads a lot easier than here in R) and an example analysis using WHONET data.

Index