

Package ‘AMR’

March 29, 2019

Version 0.6.1

Date 2019-03-28

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), 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 2019-03-29 12:00:03 UTC

R topics documented:

abname	3
age	4
age_groups	5
AMR	7
antibiotics	8
as.atc	10
as.mic	11
as.mo	12
as.rsi	17
atc_online_property	19
atc_property	20
availability	22
catalogue_of_life	23
catalogue_of_life_version	24
count	25
eucast_rules	28
filter_ab_class	31
first_isolate	33
freq	37
g.test	42
get_locale	45
ggplot_rsi	46
guess_ab_col	49
join	51
key_antibiotics	52
kurtosis	55
like	56
mdro	57
microorganisms	61
microorganisms.codes	63
microorganisms.old	64
mo_property	65
mo_source	69
p.symbol	71
portion	72
read.4D	75
resistance_predict	77
rsi	80
septic_patients	81
skewness	82
WHOCC	83
WHONET	83

abname	<i>Name of an antibiotic</i>
--------	------------------------------

Description

Convert antibiotic codes to a (trivial) antibiotic name or ATC code, or vice versa. This uses the data from [antibiotics](#).

Usage

```
abname(abcode, from = c("guess", "atc", "certe", "umcg"),
       to = "official", textbetween = " + ", tolower = FALSE)
```

Arguments

abcode	a code or name, like "AMOX", "AMCL" or "J01CA04"
from, to	type to transform from and to. See antibiotics for its column names. With from = "guess" the from will be guessed from "atc", "certe" and "umcg". When using to = "atc", the ATC code will be searched using as.atc .
textbetween	text to put between multiple returned texts
tolower	return output as lower case with function tolower .

Details

The [ab_property](#) functions are **faster and more concise**, but do not support concatenated strings, like `abname("AMCL+GENT")`.

WHOCC

This package contains **all ~500 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 comprehensive 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[antibiotics](#)**Examples**

```

abname("AMCL")
# "Amoxicillin and beta-lactamase inhibitor"

# It is quite flexible at default (having `from = "guess"`)
abname(c("amox", "J01CA04", "Trimox", "dispermox", "Amoxil"))
# "Amoxicillin" "Amoxicillin" "Amoxicillin" "Amoxicillin" "Amoxicillin"

# Multiple antibiotics can be combined with "+".
# The second antibiotic will be set to lower case when `tolower` was not set:
abname("AMCL+GENT", textbetween = "/")
# "amoxicillin and enzyme inhibitor/gentamicin"

abname(c("AMCL", "GENT"))
# "Amoxicillin and beta-lactamase inhibitor" "Gentamicin"

abname("AMCL", to = "trivial_nl")
# "Amoxicilline/clavulaanzuur"

abname("AMCL", to = "atc")
# "J01CR02"

# specific codes for University Medical Center Groningen (UMCG):
abname("J01CR02", from = "atc", to = "umcg")
# "AMCL"

# specific codes for Certe:
abname("J01CR02", from = "atc", to = "certe")
# "amcl"

```

age

Age in years of individuals

Description

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

Usage

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

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

Value

Integer (no decimals)

Read more on our website!

On our website <https://msberends.gitlab.io/AMR> you can find a [comprehensive 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

```
df <- data.frame(birth_date = Sys.Date() - runif(100) * 25000)
df$age <- age(df$birth_date)
```

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, 26-54, 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 (26-54), middle-aged adults (55-74) and elderly (75+).
- A character:
 - "children", 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, 95)`. This will split on 0-64, 65-74, 75-84, 85-94 and 95+.
- "fives", equivalent of: `1:24 * 5`. This will split on 0-4, 5-9, 10-14, 15-19 and so forth, until 120.
- "tens", equivalent of: `1:12 * 10`. This will split on 0-9, 10-19, 20-29 and so forth, until 120.

Value

Ordered `factor`

Read more on our website!

On our website <https://msberends.gitlab.io/AMR> you can find a [comprehensive 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, 1:12 * 10)
age_groups(ages, split_at = "tens")

# split into groups of five years
age_groups(ages, 1:24 * 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 ciprofloxacin per age group
library(dplyr)
septic_patients %>%
  mutate(first_isolate = first_isolate(.)) %>%
  filter(first_isolate == TRUE,
         mo == as.mo("E. coli")) %>%
  group_by(age_group = age_groups(age)) %>%
```

```
select(age_group,  
       cipr) %>%  
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 almost all 60,000 microbial (sub)species from the Catalogue of Life
- Calculating antimicrobial resistance
- 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 (not the translation from MIC to RSI values)
- Determining multi-drug resistant organisms (MDRO)
- 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 - rug.nl umcg.nl

[2] Certe Medical Diagnostics & Advice, Groningen, the Netherlands - certe.nl

Read more on our website!

On our website <https://msberends.gitlab.io/AMR> you can find [a comprehensive 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 ~500 antibiotics

Description

A data set containing all antibiotics with a J0 code and some other antimicrobial agents, with their DDDs. Except for trade names and abbreviations, all properties were downloaded from the WHO, see Source.

Usage

antibiotics

Format

A [data.frame](#) with 488 observations and 17 variables:

atc ATC code (Anatomical Therapeutic Chemical), like J01CR02
 ears_net EARS-Net code (European Antimicrobial Resistance Surveillance Network), like AMC
 certe Certe code, like amcl
 umcg UMCG code, like AMCL
 abbr Abbreviation as used by many countries, used internally by [as.atc](#)
 official Official name by the WHO, like "Amoxicillin and beta-lactamase inhibitor"
 official_nl Official name in the Netherlands, like "Amoxicilline met enzymremmer"
 trivial_nl Trivial name in Dutch, like "Amoxicilline/clavulaanzuur"
 trade_name Trade name as used by many countries (a total of 294), used internally by [as.atc](#)
 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
atc_group1 ATC group, like "Macrolides, lincosamides and streptogramins"
atc_group2 Subgroup of atc_group1, like "Macrolides"
useful_gramnegative FALSE if not useful according to EUCAST, NA otherwise (see Source)
useful_grampositive FALSE if not useful according to EUCAST, NA otherwise (see Source)

WHOCC

This package contains **all ~500 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.whooc.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 **comprehensive 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: https://www.whooc.no/atc_ddd_index/

Table antibiotic coding EARSS (from WHONET 5.3): http://www.madsonline.dk/Tutorials/landskoder_antibiotika_WM.pdf

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

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

See Also

[microorganisms](#)

`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.whooc.no/atc/structure_and_principles/

Value

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

WHOCC

This package contains **all ~500 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.whooc.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 comprehensive 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.atc("Erythrocin") # Trade name
as.atc("Eryzole")    # Trade name
as.atc("Pediamycin") # Trade name

# Use ab_* functions to get a specific property based on an ATC code
Cipro <- as.atc("cipro") # returns `J01MA02`
atc_official(Cipro)     # returns "Ciprofloxacin"
atc_umcg(Cipro)        # returns "CIPR", the code used in the UMG
```

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
```

Value

Ordered factor with new class `mic`

Read more on our website!

On our website <https://msberends.gitlab.io/AMR> you can find a [comprehensive 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

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()

clean_mo_history(...)
```

Arguments

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

Self-learning algorithm

The `as.mo()` function gains experience from previously determined microbial IDs and learns from it. This drastically improves both speed and reliability. Use `clean_mo_history()` to reset the algorithms. Only experience from your current AMR package version is used. This is done because in the future the taxonomic tree (which is included in this package) may change for any organism and it consequently has to rebuild its knowledge.

Usually, any guess after the first try runs 80-95% faster than the first try.

For now, learning only works per session. If R is closed or terminated, the algorithms reset. This will probably be resolved in a next version.

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 takes 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 negatives, like all Enterobacteriaceae and e.g. *Pseudomonas* and *Legionella*.

Group 2 probably contains all other microbial pathogens ever found in humans.

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. 2019 Mar 11. <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), 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 (~60,000 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 comprehensive 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 aureuz") # handles incorrect spelling
as.mo("MRSA") # Methicillin Resistant S. aureus
as.mo("VISA") # Vancomycin Intermediate S. aureus
as.mo("VRSA") # Vancomycin Resistant S. aureus

# 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

This transforms a vector 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)
```

```
is.rsi(x)
```

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

Arguments

x vector
 threshold maximum fraction of x that is allowed to fail transformation, see Examples

Details

The function `is.rsi.eligible` returns TRUE when a column contains only valid antimicrobial interpretations (S and/or I and/or R), and FALSE otherwise.

Value

Ordered factor with new class `rsi`

Read more on our website!

On our website <https://msberends.gitlab.io/AMR> you can find a **comprehensive 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

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(peni:rifa), 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 = 0.9) # 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 = "0",  
  url = "https://www.whooc.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 [comprehensive 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/abbreviations/

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"
```

atc_property

Property of an antibiotic

Description

Use these functions to return a specific property of an antibiotic from the [antibiotics](#) data set, based on their ATC code. Get such a code with [as.atc](#).

Usage

```
atc_property(x, property = "official")  
  
atc_official(x, language = NULL)  
  
atc_name(x, language = NULL)  
  
atc_trivial_nl(x)  
  
atc_certe(x)  
  
atc_umcg(x)  
  
atc_tradenames(x)
```

Arguments

x	a (vector of a) valid atc code or any text that can be coerced to a valid atc with as.atc
property	one of the column names of one of the antibiotics data set, like "atc" and "official"
language	language of the returned text, defaults to English ("en") and can be set with getOption("AMR_locale") . Either one of "en" (English) or "nl" (Dutch).

Value

A vector of values. In case of `atc_tradenames`, if `x` is of length one, a vector will be returned. Otherwise a [list](#), with `x` as names.

Read more on our website!

On our website <https://msberends.gitlab.io/AMR> you can find a [comprehensive 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

```
as.atc("amcl")          # J01CR02  
atc_name("amcl")        # Amoxicillin and beta-lactamase inhibitor  
atc_name("amcl", "nl")  # Amoxicilline met enzymremmer  
atc_trivial_nl("amcl")  # Amoxicilline/clavulaanzuur  
atc_certe("amcl")       # amcl  
atc_umcg("amcl")       # AMCL
```

`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

<code>tbl</code>	a <code>data.frame</code> or <code>list</code>
<code>width</code>	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 [comprehensive 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 (~60,000 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*, *Histoplasma*, *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 ~15,000 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/reproduction_of_microorganisms.R.

Read more on our website!

On our website <https://msberends.gitlab.io/AMR> you can find a **comprehensive 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_shortcode("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_phylum("C. elegans")
# [1] "Cyanobacteria" # Bacteria?!
mo_fullname("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. It also shows if the included version is their latest annual release. The Catalogue of Life releases their annual release in March each year.

Usage

```
catalogue_of_life_version()
```

Details

The list item `...$catalogue_of_life$is_latest_annual_release` is based on the system date. 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 (~60,000 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 comprehensive 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 dplyrs [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 = getOption("get_antibiotic_names",
  "official"), 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 `abname`. This can be set with `getOption("get_antibiotic_names")`.

`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. IR (susceptible vs. non-susceptible)

Details

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

`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(...)`.

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

Value

Integer

Read more on our website!

On our website <https://msberends.gitlab.io/AMR> you can find a [comprehensive 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$amox)
count_IR(septic_patients$amox)

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

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

# 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$amox)
portion_IR(septic_patients$amox) * n_rsi(septic_patients$amox)

library(dplyr)
septic_patients %>%
  group_by(hospital_id) %>%
  summarise(R = count_R(cipr),
            I = count_I(cipr),
            S = count_S(cipr),
            n1 = count_all(cipr), # the actual total; sum of all three
            n2 = n_rsi(cipr),     # 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$amcl) # S = 1057 (67.1%)
count_all(septic_patients$amcl) # n = 1576
```

```

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

with(septic_patients,
      count_S(amcl, gent))      # S = 1396 (92.0%)
with(septic_patients,
      n_rsi(amcl, gent))       # n = 1517

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

# It also supports grouping variables
septic_patients %>%
  select(hospital_id, amox, cizr) %>%
  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(tbl, col_mo = NULL, info = TRUE,
             rules = c("breakpoints", "expert", "other", "all"), verbose = FALSE,
             amcl = guess_ab_col(), amik = guess_ab_col(),
             amox = guess_ab_col(), ampi = guess_ab_col(),
             azit = guess_ab_col(), azlo = guess_ab_col(),
             aztr = guess_ab_col(), cefa = guess_ab_col(),
             cfep = guess_ab_col(), cfot = guess_ab_col(),
             cfox = guess_ab_col(), cfra = guess_ab_col(),
             cfta = guess_ab_col(), cftr = guess_ab_col(),
             cfur = guess_ab_col(), chlo = guess_ab_col(),
             cizr = guess_ab_col(), clar = guess_ab_col(),
             clin = guess_ab_col(), clox = guess_ab_col(),
             coli = guess_ab_col(), czol = guess_ab_col(),
             dapt = guess_ab_col(), doxy = guess_ab_col(),
             erta = guess_ab_col(), eryt = guess_ab_col(),
             fosf = guess_ab_col(), fusi = guess_ab_col(),
             gent = guess_ab_col(), imip = guess_ab_col(),

```

```

kana = guess_ab_col(), levo = guess_ab_col(),
linc = guess_ab_col(), line = guess_ab_col(),
mero = guess_ab_col(), mezl = guess_ab_col(),
mino = guess_ab_col(), moxi = guess_ab_col(),
nali = guess_ab_col(), neom = guess_ab_col(),
neti = guess_ab_col(), nitr = guess_ab_col(),
norf = guess_ab_col(), novo = guess_ab_col(),
oflo = guess_ab_col(), oxac = guess_ab_col(),
peni = guess_ab_col(), pipe = guess_ab_col(),
pita = guess_ab_col(), poly = guess_ab_col(),
pris = guess_ab_col(), qida = guess_ab_col(),
rifa = guess_ab_col(), roxi = guess_ab_col(),
siso = guess_ab_col(), teic = guess_ab_col(),
tetr = guess_ab_col(), tica = guess_ab_col(),
tige = guess_ab_col(), tobr = guess_ab_col(),
trim = guess_ab_col(), trsu = guess_ab_col(),
vanc = guess_ab_col()

```

```
EUCAST_rules(...)
```

```
interpretive_reading(...)
```

Arguments

tbl	table with antibiotic columns, like e.g. amox and amcl
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.
amcl, amik, amox, ampi, azit, azlo, aztr, cefa, cfep, cfot, cfox, cfra, cfta, cftr, cfur, chlo, cip	column name of an antibiotic, see Antibiotics
...	parameters that are passed on to eucast_rules

Value

The input of `tbl`, 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. `tica = NULL`). Non-existing columns will anyway be skipped with a warning.

Abbreviations of the column containing antibiotics in the form: **abbreviation:** generic name (*ATC code*)

amcl: amoxicillin+clavulanic acid (J01CR02), **amik:** amikacin (J01GB06), **amox:** amoxicillin (J01CA04), **ampi:** ampicillin (J01CA01), **azit:** azithromycin (J01FA10), **azlo:** azlocillin (J01CA09), **aztr:** aztreonam (J01DF01), **cefa:** cefaloridine (J01DB02), **cfep:** cefepime (J01DE01), **cfot:** cefotaxime (J01DD01), **cfos:** cefoxitin (J01DC01), **cfra:** cefradine (J01DB09), **cfta:** ceftazidime (J01DD02), **cftr:** ceftriaxone (J01DD04), **cfur:** cefuroxime (J01DC02), **chlo:** chloramphenicol (J01BA01), **cipr:** ciprofloxacin (J01MA02), **clar:** clarithromycin (J01FA09), **clin:** clindamycin (J01FF01), **clox:** flucloxacillin (J01CF05), **coli:** colistin (J01XB01), **czol:** ceftazidime (J01DB04), **dapt:** daptomycin (J01XX09), **doxy:** doxycycline (J01AA02), **erta:** ertapenem (J01DH03), **eryt:** erythromycin (J01FA01), **fosf:** fosfomicin (J01XX01), **fusi:** fusidic acid (J01XC01), **gent:** gentamicin (J01GB03), **imip:** imipenem (J01DH51), **kana:** kanamycin (J01GB04), **levo:** levofloxacin (J01MA12), **line:** linezolid (J01XX08), **mero:** meropenem (J01DH02), **mezl:** mezlocillin (J01CA10), **mino:** minocycline (J01AA08), **moxi:** moxifloxacin (J01MA14), **nali:** nalidixic acid (J01MB02), **neom:** neomycin (J01GB05), **neti:** netilmicin (J01GB07), **nitr:** nitrofurantoin (J01XE01), **norf:** norfloxacin (J01MA06), **novo:** novobiocin (an ATCvet code: QJ01XX95), **oflo:** ofloxacin (J01MA01), **peni:** (benzyl)penicillin (J01CE01), **pipe:** piperacillin (J01CA12), **pita:** piperacillin+tazobactam (J01CR05), **poly:** polymyxin B (J01XB02), **pris:** pristinamycin (J01FG01), **qida:** quinupristin/dalfopristin (J01FG02), **rifa:** rifampicin (J04AB02), **roxi:** roxithromycin (J01FA06), **siso:** sisomicin (J01GB08), **teic:** teicoplanin (J01XA02), **tetr:** tetracycline (J01AA07), **tica:** ticarcillin (J01CA13), **tige:** tigecycline (J01AA12), **tobr:** tobramycin (J01GB01), **trim:** trimethoprim (J01EA01), **trsu:** sulfamethoxazole and trimethoprim (J01EE01), **vanc:** vancomycin (J01XA01).

Read more on our website!

On our website <https://msberends.gitlab.io/AMR> you can find a **comprehensive 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"),
               vanc = "-",      # Vancomycin
               amox = "-",      # Amoxicillin
               coli = "-",      # Colistin
               cfta = "-",      # Ceftazidime
               cfur = "-",      # Cefuroxime
               peni = "S",      # Benzylpenicillin
               cfox = "S",      # Cefoxitin
               stringsAsFactors = FALSE)

a
#           mo vanc amox coli cfta cfur peni cfox
# 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 vanc amox coli cfta cfur peni cfox
# 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:
septic_patients %>%

```



```

filter_at(.vars = vars(c("gent", "tobr", "amik", "kana")),
         .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(tbl, 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(tbl, col_date = NULL, col_patient_id = NULL,
                    col_mo = NULL, ...)

filter_first_weighted_isolate(tbl, col_date = NULL,
                              col_patient_id = NULL, col_mo = NULL, col_keyantibiotics = NULL,
                              ...)

```

Arguments

tbl	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 <i>weighted</i> 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 function `filter_first_isolate` is essentially equal to:

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

The function `filter_first_weighted_isolate` is essentially equal to:

```
tbl %>%
  mutate(keyab = key_antibiotics()) %>%
  mutate(only_weighted_firsts = first_isolate(tbl,
                                             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 SIR (or vice versa) will lead to this. This is a reliable method and 30-35 times faster than method 2.

2. Using `type = "points"` and parameter `points_threshold`

A difference from I to SIR (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`, 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 **comprehensive 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(gent),          # gentamicin availability
            resistance = portion_IR(gent)) # gentamicin resistance
```

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

```
# Have a look at A and B.
# B is more reliable because every isolate is only counted once.
# Gentamicin resistance in hospital D appears to be 5.4% higher than
# when you (erroneously) would have used all isolates!
```

```
## OTHER EXAMPLES:
```

```
## Not run:
```

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

```
tbl$first_isolate <-
  first_isolate(tbl)
```

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

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

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

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

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

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

tbl$first_resp_isolate_weighed <-
  first_isolate(tbl,
                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 quasiquote 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

```
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 != ",", ", ", "."))
```

```

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 !=
     ",", ", ", "."))

top_freq(f, n)

header(f, property = NULL)

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

```

Arguments

<code>x</code>	vector of any class or a <code>data.frame</code> , <code>tibble</code> (may contain a grouping variable) or <code>table</code>
<code>...</code>	up to nine different columns of <code>x</code> when <code>x</code> is a <code>data.frame</code> or <code>tibble</code> , to calculate frequencies from - see Examples. Also supports quasiquotation.
<code>sort.count</code>	sort on count, i.e. frequencies. This will be <code>TRUE</code> at default for everything except when using grouping variables.
<code>nmax</code>	number of row to print. The default, 15, uses <code>getOption("max.print.freq")</code> . Use <code>nmax = 0</code> , <code>nmax = Inf</code> , <code>nmax = NULL</code> or <code>nmax = NA</code> to print all rows.
<code>na.rm</code>	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.
<code>row.names</code>	a logical value indicating whether row indices should be printed as <code>1:nrow(x)</code>
<code>markdown</code>	a logical value indicating whether the frequency table should be printed in markdown format. This will print all rows (except when <code>nmax</code> is defined) and is default behaviour in non-interactive R sessions (like when knitting RMarkdown files).
<code>digits</code>	how many significant digits are to be used for numeric values in the header (not for the items themselves, that depends on <code>getOption("digits")</code>)
<code>quote</code>	a logical value indicating whether or not strings should be printed with surrounding quotes
<code>header</code>	a logical value indicating whether an informative header should be printed
<code>title</code>	text to show above frequency table, at default to tries to coerce from the variables passed to <code>x</code>
<code>na</code>	a character string that should be used to show empty (NA) values (only useful when <code>na.rm = FALSE</code>)
<code>droplevels</code>	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 <code>prettyNum:</code> that help page explains the details.
big.mark	used for prettying (longish) numerical and complex sequences. Passed to <code>prettyNum:</code> that help page explains the details.
f	a frequency table
n	number of top n items to return, use <code>-n</code> 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 `"frequency_tbl"`) 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 **comprehensive 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 quasi-quotation 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 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$trim),
      freq(septic_patients$trsu))
```

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

<code>x</code>	a numeric vector or matrix. <code>x</code> and <code>y</code> can also both be factors.
<code>y</code>	a numeric vector; ignored if <code>x</code> is a matrix. If <code>x</code> is a factor, <code>y</code> should be a factor of the same length.
<code>p</code>	a vector of probabilities of the same length of <code>x</code> . An error is given if any entry of <code>p</code> is negative.
<code>rescale.p</code>	a logical scalar; if TRUE then <code>p</code> is rescaled (if necessary) to sum to 1. If <code>rescale.p</code> is FALSE, and <code>p</code> 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 $n = \text{sum}(x)$. 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, $(\text{observed} - \text{expected}) / \text{sqrt}(\text{expected})$.
stdres	standardized residuals, $(\text{observed} - \text{expected}) / \text{sqrt}(V)$, 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 comprehensive 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.
```

get_locale

Get language for AMR

Description

Determines the system language to be used for language-dependent output of AMR functions, like [mo_gramstain](#) and [mo_type](#).

Usage

```
get_locale()
```

Details

The system language can be overwritten with [getOption\("AMR_locale"\)](#).

Supported languages

Supported languages are "en" (English), "de" (German), "nl" (Dutch), "es" (Spanish), "it" (Italian), "fr" (French), and "pt" (Portuguese).

Read more on our website!

On our website <https://msberends.gitlab.io/AMR> you can find a [comprehensive 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](#).

ggplot_rsi

AMR plots with ggplot2

Description

Use these functions to create bar plots for antimicrobial resistance analysis. All functions rely on internal [ggplot](#) functions.

Usage

```
ggplot_rsi(data, position = NULL, x = "Antibiotic",
  fill = "Interpretation", facet = NULL, breaks = seq(0, 1, 0.1),
  limits = NULL, translate_ab = "official", fun = count_df,
  nrow = NULL, datalabels = TRUE, datalabels.size = 3,
  datalabels.colour = "grey15", ...)

geom_rsi(position = NULL, x = c("Antibiotic", "Interpretation"),
  fill = "Interpretation", translate_ab = "official", 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()

theme_rsi()

labels_rsi_count(position = NULL, x = "Antibiotic",
  datalabels.size = 3, datalabels.colour = "grey15")
```

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 into, using abname . Default behaviour is to translate to official names according to the WHO. Use <code>translate_ab = FALSE</code> to disable translation.
fun	function to transform data, either count_df (default) or portion_df
nrow	(when using facet) number of rows
datalabels	show datalabels using <code>labels_rsi_count</code> , will at default only be shown when <code>fun = count_df</code>
datalabels.size	size of the datalabels
datalabels.colour	colour of the datalabels
...	other parameters passed on to <code>geom_rsi</code>

Details

At default, the names of antibiotics will be shown on the plots using [abname](#). This can be set with the option `get_antibiotic_names` (a logical value), so change it e.g. to `FALSE` with options(`get_antibiotic_names = FALSE`)

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: green for S, yellow for I and 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 [comprehensive 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(amox, nitr, fosf, trim, cipr)) +
  geom_rsi()

# prettify the plot using some additional functions:
df <- septic_patients[, c("amox", "nitr", "fosf", "trim", "cipr")]
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(amox, nitr, fosf, trim, cipr) %>%
  ggplot_rsi()

# get only portions and no counts:
septic_patients %>%
  select(amox, nitr, fosf, trim, cipr) %>%
  ggplot_rsi(fun = portion_df)

# add other ggplot2 parameters as you like:
septic_patients %>%
  select(amox, nitr, fosf, trim, cipr) %>%
  ggplot_rsi(width = 0.5,
             colour = "black",
             size = 1,
             linetype = 2,
             alpha = 0.25)

# resistance of ciprofloxacin 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,
         cipr) %>%
  ggplot_rsi(x = "age_group")
```



```

# for colourblind mode, use divergent colours from the viridis package:
septic_patients %>%
  select(amox, nitr, fosf, trim, cipr) %>%
  ggplot_rsi() + scale_fill_viridis_d()

# it also supports groups (don't forget to use the group var on `x` or `facet`):
septic_patients %>%
  select(hospital_id, amox, nitr, fosf, trim, cipr) %>%
  group_by(hospital_id) %>%
  ggplot_rsi(x = hospital_id,
            facet = Antibiotic,
            nrow = 1) +
  labs(title = "AMR of Anti-UTI Drugs Per Hospital",
        x = "Hospital")

# genuine analysis: check 2 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)) %>%
  # determine first isolates
  mutate(first_isolate = first_isolate(.,
                                       col_date = "date",
                                       col_patient_id = "patient_id",
                                       col_mo = "mo")) %>%

  # filter on first isolates
  filter(first_isolate == TRUE) %>%
  # get short MO names (like "E. coli")
  mutate(mo = mo_shortname(mo, Becker = TRUE)) %>%
  # select this short name and some antiseptic drugs
  select(mo, cfur, gent, cipr) %>%
  # group by MO
  group_by(mo) %>%
  # plot the thing, putting MOs on the facet
  ggplot_rsi(x = Antibiotic,
            facet = mo,
            translate_ab = FALSE,
            nrow = 1) +
  labs(title = "AMR of Top Three Microorganisms In Blood Culture Isolates",
        subtitle = "Only First Isolates, CoNS grouped according to Becker et al. (2014)",
        x = "Microorganisms")

```

Description

This tries to find a column name in a data set based on information from the [antibiotics](#) data set. Also supports WHONET abbreviations. You can look for an antibiotic (trade) name or abbreviation and it will search the `data.frame` for any column containing a name or ATC code of that antibiotic.

Usage

```
guess_ab_col(tbl = NULL, col = NULL, verbose = FALSE)
```

Arguments

<code>tbl</code>	a <code>data.frame</code>
<code>col</code>	a character to look for
<code>verbose</code>	a logical to indicate whether additional info should be printed

Read more on our website!

On our website <https://msberends.gitlab.io/AMR> you can find a [comprehensive 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)
# using column `tetr` for col "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.atc("augmentin"))
# [1] "AMC_ED20"
```

join	<i>Join a table with microorganisms</i>
------	---

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

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

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 [comprehensive 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

This 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(tbl, col_mo = NULL, universal_1 = guess_ab_col(tbl,
  "amox"), universal_2 = guess_ab_col(tbl, "amcl"),
  universal_3 = guess_ab_col(tbl, "cfur"),
  universal_4 = guess_ab_col(tbl, "pita"),
  universal_5 = guess_ab_col(tbl, "cipr"),
  universal_6 = guess_ab_col(tbl, "trsu"),
  GramPos_1 = guess_ab_col(tbl, "vanc"), GramPos_2 = guess_ab_col(tbl,
  "teic"), GramPos_3 = guess_ab_col(tbl, "tetr"),
  GramPos_4 = guess_ab_col(tbl, "eryt"), GramPos_5 = guess_ab_col(tbl,
  "oxac"), GramPos_6 = guess_ab_col(tbl, "rifa"),
  GramNeg_1 = guess_ab_col(tbl, "gent"), GramNeg_2 = guess_ab_col(tbl,
  "tobr"), GramNeg_3 = guess_ab_col(tbl, "coli"),
  GramNeg_4 = guess_ab_col(tbl, "cfot"), GramNeg_5 = guess_ab_col(tbl,
  "cfta"), GramNeg_6 = guess_ab_col(tbl, "mero"), warnings = TRUE, ...)

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

```

Arguments

tbl	table with antibiotics coloms, like amox and amcl.
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
x, y	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 (column names): "amox", "amcl", "cfur", "pita", "cipr", "trsu" (until here is universal), "vanc", "teic", "tetr", "eryt", "oxac", "rifa".

At default, the antibiotics that are used for **Gram negative bacteria** are (column names): "amox", "amcl", "cfur", "pita", "cipr", "trsu" (until here is universal), "gent", "tobr", "coli", "cfot", "cfta", "mero".

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 SIR (or vice versa) will lead to this. This is a reliable method and 30-35 times faster than method 2.

2. Using type = "points" and parameter points_threshold

A difference from I to SIR (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, 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 **comprehensive 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 <- "SSRR.S.R..S"
strainB <- "SSIRSSSRSS"

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 [comprehensive 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	<i>Pattern Matching</i>
------	-------------------------

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 <code>as.character</code> to a character vector. Long vectors are supported.
pattern	character string containing a regular expression (or character string for <code>fixed = TRUE</code>) to be matched in the given character vector. Coerced by <code>as.character</code> 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 <code>regexpr</code> and <code>gregexpr</code> .

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 comprehensive 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(tbl, country = NULL, col_mo = NULL, info = TRUE,
  amcl = guess_ab_col(), amik = guess_ab_col(),
  amox = guess_ab_col(), ampi = guess_ab_col(),
  azit = guess_ab_col(), aztr = guess_ab_col(),
  cefa = guess_ab_col(), cfra = guess_ab_col(),
  cfep = guess_ab_col(), cfot = guess_ab_col(),
  cfox = guess_ab_col(), cfta = guess_ab_col(),
  cftr = guess_ab_col(), cfur = guess_ab_col(),
  chlo = guess_ab_col(), cipr = guess_ab_col(),
  clar = guess_ab_col(), clin = guess_ab_col(),
  clox = guess_ab_col(), coli = guess_ab_col(),
  czol = guess_ab_col(), dapt = guess_ab_col(),
  doxy = guess_ab_col(), erta = guess_ab_col(),
  eryt = guess_ab_col(), fosf = guess_ab_col(),
  fusi = guess_ab_col(), gent = guess_ab_col(),
  imip = guess_ab_col(), kana = guess_ab_col(),

```

```

levo = guess_ab_col(), linc = guess_ab_col(),
line = guess_ab_col(), mero = guess_ab_col(),
metr = guess_ab_col(), mino = guess_ab_col(),
moxi = guess_ab_col(), nali = guess_ab_col(),
neom = guess_ab_col(), neti = guess_ab_col(),
nitr = guess_ab_col(), novo = guess_ab_col(),
norf = guess_ab_col(), oflo = guess_ab_col(),
peni = guess_ab_col(), pipe = guess_ab_col(),
pita = guess_ab_col(), poly = guess_ab_col(),
qida = guess_ab_col(), rifa = guess_ab_col(),
roxi = guess_ab_col(), siso = guess_ab_col(),
teic = guess_ab_col(), tetr = guess_ab_col(),
tica = guess_ab_col(), tige = guess_ab_col(),
toibr = guess_ab_col(), trim = guess_ab_col(),
trsu = guess_ab_col(), vanc = guess_ab_col()

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

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

eucast_exceptional_phenotypes(tbl, country = "EUCAST", ...)

```

Arguments

tbl	table with antibiotic columns, like e.g. amox and amcl
country	country code to determine guidelines. EUCAST rules will be used when left empty, see Details. Should be or a code from the list of ISO 3166-1 alpha-2 country codes . Case-insensitive. Currently supported are de (Germany) and nl (the Netherlands).
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
amcl	column name of an antibiotic, see Antibiotics
amik	column name of an antibiotic, see Antibiotics
amox	column name of an antibiotic, see Antibiotics
ampi	column name of an antibiotic, see Antibiotics
azit	column name of an antibiotic, see Antibiotics
aztr	column name of an antibiotic, see Antibiotics
cefa	column name of an antibiotic, see Antibiotics
cfra	column name of an antibiotic, see Antibiotics
cfep	column name of an antibiotic, see Antibiotics
cfot	column name of an antibiotic, see Antibiotics
cfox	column name of an antibiotic, see Antibiotics
cfta	column name of an antibiotic, see Antibiotics

cftr	column name of an antibiotic, see Antibiotics
cfur	column name of an antibiotic, see Antibiotics
chlo	column name of an antibiotic, see Antibiotics
cipr	column name of an antibiotic, see Antibiotics
clar	column name of an antibiotic, see Antibiotics
clin	column name of an antibiotic, see Antibiotics
clox	column name of an antibiotic, see Antibiotics
coli	column name of an antibiotic, see Antibiotics
czol	column name of an antibiotic, see Antibiotics
dapt	column name of an antibiotic, see Antibiotics
doxy	column name of an antibiotic, see Antibiotics
erta	column name of an antibiotic, see Antibiotics
eryt	column name of an antibiotic, see Antibiotics
fosf	column name of an antibiotic, see Antibiotics
fusi	column name of an antibiotic, see Antibiotics
gent	column name of an antibiotic, see Antibiotics
imip	column name of an antibiotic, see Antibiotics
kana	column name of an antibiotic, see Antibiotics
levo	column name of an antibiotic, see Antibiotics
linc	column name of an antibiotic, see Antibiotics
line	column name of an antibiotic, see Antibiotics
mero	column name of an antibiotic, see Antibiotics
metr	column name of an antibiotic, see Antibiotics
mino	column name of an antibiotic, see Antibiotics
moxi	column name of an antibiotic, see Antibiotics
nali	column name of an antibiotic, see Antibiotics
neom	column name of an antibiotic, see Antibiotics
neti	column name of an antibiotic, see Antibiotics
nitr	column name of an antibiotic, see Antibiotics
novo	column name of an antibiotic, see Antibiotics
norf	column name of an antibiotic, see Antibiotics
oflo	column name of an antibiotic, see Antibiotics
peni	column name of an antibiotic, see Antibiotics
pipe	column name of an antibiotic, see Antibiotics
pita	column name of an antibiotic, see Antibiotics
poly	column name of an antibiotic, see Antibiotics
qida	column name of an antibiotic, see Antibiotics

rifa	column name of an antibiotic, see Antibiotics
roxi	column name of an antibiotic, see Antibiotics
siso	column name of an antibiotic, see Antibiotics
teic	column name of an antibiotic, see Antibiotics
tetr	column name of an antibiotic, see Antibiotics
tica	column name of an antibiotic, see Antibiotics
tige	column name of an antibiotic, see Antibiotics
tobr	column name of an antibiotic, see Antibiotics
trim	column name of an antibiotic, see Antibiotics
trsu	column name of an antibiotic, see Antibiotics
vanc	column name of an antibiotic, see Antibiotics
...	parameters that are passed on to methods

Details

When country will be left blank, guidelines will be taken from EUCAST Expert Rules Version 3.1 "Intrinsic Resistance and Exceptional Phenotypes Tables" (http://www.eucast.org/fileadmin/src/media/PDFs/EUCAST_files/Expert_Rules/Expert_rules_intrinsic_exceptional_V3.1.pdf).

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. `tica = NULL`). Non-existing columns will anyway be skipped with a warning.

Abbreviations of the column containing antibiotics in the form: **abbreviation**: generic name (*ATC code*)

amcl: amoxicillin+clavulanic acid (**J01CR02**), **amik**: amikacin (**J01GB06**), **amox**: amoxicillin (**J01CA04**), **ampi**: ampicillin (**J01CA01**), **azit**: azithromycin (**J01FA10**), **azlo**: azlocillin (**J01CA09**), **aztr**: aztreonam (**J01DF01**), **cefa**: cefaloridine (**J01DB02**), **cfep**: cefepime (**J01DE01**), **cfot**: cefotaxime (**J01DD01**), **cfox**: cefoxitin (**J01DC01**), **cfra**: cefradine (**J01DB09**), **cfta**: ceftazidime (**J01DD02**), **cftr**: ceftriaxone (**J01DD04**), **cfur**: cefuroxime (**J01DC02**), **chlo**: chloramphenicol (**J01BA01**), **cipr**: ciprofloxacin (**J01MA02**), **clar**: clarithromycin (**J01FA09**), **clin**: clindamycin (**J01FF01**), **clox**: flucloxacillin (**J01CF05**), **coli**: colistin (**J01XB01**), **czol**: cefazolin (**J01DB04**), **dapt**: daptomycin (**J01XX09**), **doxy**: doxycycline (**J01AA02**), **erta**: ertapenem (**J01DH03**), **eryt**: erythromycin (**J01FA01**), **fosf**: fosfomicin (**J01XX01**), **fusi**: fusidic acid (**J01XC01**), **gent**: gentamicin (**J01GB03**), **imip**: imipenem (**J01DH51**), **kana**: kanamycin (**J01GB04**), **levo**: levofloxacin (**J01MA12**), **line**: linezolid (**J01XX08**), **mero**: meropenem (**J01DH02**), **mezl**: mezlocillin (**J01CA10**), **mino**: minocycline (**J01AA08**), **moxi**: moxifloxacin (**J01MA14**), **nali**: nalidixic acid (**J01MB02**), **neom**: neomycin (**J01GB05**), **neti**: netilmicin (**J01GB07**), **nitr**: nitrofurantoin (**J01XE01**), **norf**: norfloxacin (**J01MA06**), **novo**: novobiocin (an ATCvet code:

QJ01XX95), **oflo**: ofloxacin (**J01MA01**), **peni**: (benzyl)penicillin (**J01CE01**), **pipe**: piperacillin (**J01CA12**), **pita**: piperacillin+tazobactam (**J01CR05**), **poly**: polymyxin B (**J01XB02**), **pris**: pristinamycin (**J01FG01**), **qida**: quinupristin/dalfopristin (**J01FG02**), **rifa**: rifampicin (**J04AB02**), **roxi**: roxithromycin (**J01FA06**), **siso**: sisomicin (**J01GB08**), **teic**: teicoplanin (**J01XA02**), **tetr**: tetracycline (**J01AA07**), **tica**: ticarcillin (**J01CA13**), **tige**: tigecycline (**J01AA12**), **tobr**: tobramycin (**J01GB01**), **trim**: trimethoprim (**J01EA01**), **trsu**: sulfamethoxazole and trimethoprim (**J01EE01**), **vanc**: vancomycin (**J01XA01**).

Read more on our website!

On our website <https://msberends.gitlab.io/AMR> you can find a [comprehensive 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 65,629 observations and 16 variables:

```
mo ID of microorganism as used by this package
col_id Catalogue of Life ID
fullname Full name, like "Echerichia coli"
kingdom Taxonomic kingdom of the microorganism
phylum Taxonomic phylum of the microorganism
class Taxonomic class of the microorganism
order Taxonomic order of the microorganism
family Taxonomic family of the microorganism
```

genus Taxonomic genus of the microorganism
 species Taxonomic species of the microorganism
 subspecies Taxonomic subspecies of the microorganism
 rank 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 species of *Streptococcus* (beta haemolytic groups A, B, C, D, F, G, H, K and unspecified)
- 2 species of *Staphylococcus* (coagulase-negative [CoNS] and coagulase-positive [CoPS])
- 3 other undefined (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 (~60,000 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 comprehensive 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), 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 5,171 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 (~60,000 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 comprehensive 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 16,911 observations and 4 variables:

col_id Catalogue of Life ID

tsn_new New Catalogue of Life ID

fullname Old 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 (~60,000 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 comprehensive 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

[3] Catalogue of Life: Annual Checklist (public online database), www.catalogueoflife.org.

See Also

[as.mo mo_property microorganisms](#)

mo_property	<i>Property of a microorganism</i>
-------------	------------------------------------

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_fullname(x, language = get_locale(), ...)  
mo_shortcode(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_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 <code>as.mo</code>
language	language of the returned text, defaults to system language (see <code>get_locale</code>) and can also be set with <code>getOption("AMR_locale")</code> . Use <code>language = NULL</code> or <code>language = ""</code> to prevent translation.
...	other parameters passed on to <code>as.mo</code>
open	browse the URL using <code>browseURL()</code>
property	one of the column names of one of the <code>microorganisms</code> 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_fullname("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.

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

Supported languages

Supported languages are "en" (English), "de" (German), "nl" (Dutch), "es" (Spanish), "it" (Italian), "fr" (French), and "pt" (Portuguese).

Catalogue of Life

This package contains the complete taxonomic tree of almost all microorganisms (~60,000 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. 2019 Mar 11. <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), 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 **comprehensive 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_fullname("E. coli")    # "Escherichia coli"
mo_shortcode("E. coli")   # "E. coli"

## other properties
mo_gramstain("E. coli")  # "Gram negative"
```

```

mo_type("E. coli")           # "Bacteria" (equal to kingdom)
mo_rank("E. coli")          # "species"
mo_url("E. coli")           # get the direct url to the Catalogue of Life

## 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_genus("doylei")          # "Campylobacter"
mo_species("doylei")        # "jejuni"
mo_fullname("doylei")       # "Campylobacter jejuni 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 (kingdom to subspecies)
mo_taxonomy("E. coli")

```

mo_source	<i>Use predefined reference data set</i>
-----------	--

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 separated 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 [comprehensive 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	<i>Symbol of a p value</i>
----------	----------------------------

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 [comprehensive 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	<i>Calculate resistance of isolates</i>
---------	---

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 = getOption("get_antibiotic_names",
  "official"), minimum = 30, as_percent = FALSE, combine_IR = FALSE)
```

Arguments

- | | |
|------------|---|
| ... | one or more vectors (or columns) with antibiotic interpretations. They will be transformed internally with <code>as.rsi</code> 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 <code>minimum</code> 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 abname . This can be set with <code>getOption("get_antibiotic_names")</code> .
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. IR (susceptible vs. non-susceptible)

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.*

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 variable with class "rsi".

The old [rsi](#) function is still available for backwards compatibility but is deprecated.

Value

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

Read more on our website!

On our website <https://msberends.gitlab.io/AMR> you can find [a comprehensive 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$amox)
portion_IR(septic_patients$amox)

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

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

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

septic_patients %>%
  group_by(hospital_id) %>%
  summarise(R = portion_R(cipr, as_percent = TRUE),
            I = portion_I(cipr, as_percent = TRUE),
            S = portion_S(cipr, as_percent = TRUE),
            n1 = count_all(cipr), # the actual total; sum of all three
            n2 = n_rsi(cipr),    # 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(amcl)      # S = 67.1%
septic_patients %>% count_all(amcl)     # n = 1576

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

septic_patients %>% portion_S(amcl, gent) # S = 92.0%
septic_patients %>% count_all(amcl, gent) # n = 1517

septic_patients %>%
  group_by(hospital_id) %>%
  summarise(cipro_p = portion_S(cipr, as_percent = TRUE),
            cipro_n = count_all(cipr),
            genta_p = portion_S(gent, as_percent = TRUE),
            genta_n = count_all(gent),
            combination_p = portion_S(cipr, gent, as_percent = TRUE),

```

```

        combination_n = count_all(cipr, gent))

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

# It also supports grouping variables
septic_patients %>%
  select(hospital_id, amox, cipr) %>%
  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(amox, metr), # amoxicillin with metronidazole
            n = count_all(amox, metr))

## End(Not run)

```

read.4D	<i>Read data from 4D database</i>
---------	-----------------------------------

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](#).)

<code>info</code>	a logical to indicate whether info about the import should be printed, defaults to TRUE in interactive sessions
<code>header</code>	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: <code>header</code> is set to TRUE if and only if the first row contains one fewer field than the number of columns.
<code>row.names</code>	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 <code>row.names</code> is missing, the rows are numbered. Using <code>row.names = NULL</code> forces row numbering. Missing or NULL <code>row.names</code> generate row names that are considered to be ‘automatic’ (and not preserved by as.matrix).
<code>sep</code>	the field separator character. Values on each line of the file are separated by this character. If <code>sep = ""</code> (the default for <code>read.table</code>) the separator is ‘white space’, that is one or more spaces, tabs, newlines or carriage returns.
<code>quote</code>	the set of quoting characters. To disable quoting altogether, use <code>quote = ""</code> . 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 <code>colClasses</code> is specified.
<code>dec</code>	the character used in the file for decimal points.
<code>na.strings</code>	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 <i>after</i> white space is stripped from the input, so <code>na.strings</code> values may need their own white space stripped in advance.
<code>skip</code>	integer: the number of lines of the data file to skip before beginning to read data.
<code>check.names</code>	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.
<code>strip.white</code>	logical. Used only when <code>sep</code> 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 <code>as.is</code> and <code>colClasses</code> , 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 <code>file</code> , 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 comprehensive 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(tbl, col_ab, col_date = NULL, year_min = NULL,
  year_max = NULL, year_every = 1, minimum = 30,
  model = "binomial", I_as_R = TRUE, preserve_measurements = TRUE,
  info = TRUE)
```

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

```
## S3 method for class 'resistance_predict'
plot(x,
     main = paste("Resistance prediction of", attributes(x)$ab), ...)

ggplot_rsi_predict(x, main = paste("Resistance prediction of",
                                  attributes(x)$ab), ribbon = TRUE, ...)
```

Arguments

<code>tbl</code>	a <code>data.frame</code> containing isolates.
<code>col_ab</code>	column name of <code>tbl</code> with antimicrobial interpretations (R, I and S)
<code>col_date</code>	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
<code>year_min</code>	lowest year to use in the prediction model, defaults to the lowest year in <code>col_date</code>
<code>year_max</code>	highest year to use in the prediction model, defaults to 10 years after today
<code>year_every</code>	unit of sequence between lowest year found in the data and <code>year_max</code>
<code>minimum</code>	minimal amount of available isolates per year to include. Years containing less observations will be estimated by the model.
<code>model</code>	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.
<code>I_as_R</code>	a logical to indicate whether values I should be treated as R
<code>preserve_measurements</code>	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.
<code>info</code>	a logical to indicate whether textual analysis should be printed with the name and summary of the statistical model.
<code>x</code>	the coordinates of points in the plot. Alternatively, a single plotting structure, function or <i>any R object with a plot method</i> can be provided.
<code>main</code>	title of the plot
<code>...</code>	parameters passed on to the <code>first_isolate</code> function
<code>ribbon</code>	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 **comprehensive 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 = "amox", 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("peni")
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 = "amox",
                    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

Calculate resistance of isolates

Description

This function is deprecated. Use the [portion](#) functions instead.

Usage

```
rsi(ab1, ab2 = NULL, interpretation = "IR", minimum = 30,
   as_percent = FALSE, ...)
```

Arguments

ab1, ab2	vector (or column) with antibiotic interpretations. It will be transformed internally with as.rsi if needed.
interpretation	antimicrobial interpretation to check for
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%".
...	deprecated parameters to support usage on older versions

septic_patients	<i>Data set with 2,000 blood culture isolates from septic patients</i>
-----------------	--

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 comprehensive 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	<i>Skewness of the sample</i>
----------	-------------------------------

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 comprehensive 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](#)

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 ~500 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 **comprehensive 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.atc("meropenem")
atc_name("J01DH02")

atc_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 abbreviations 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 [comprehensive 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

- *Topic **Becker**
 - as.mo, 12
- *Topic **Lancefield**
 - as.mo, 12
- *Topic **ab**
 - abname, 3
- *Topic **age_group**
 - age_groups, 5
- *Topic **age**
 - age_groups, 5
- *Topic **antibiotics**
 - abname, 3
 - count, 25
 - portion, 72
- *Topic **atc**
 - as.atc, 10
- *Topic **becker**
 - as.mo, 12
- *Topic **chi**
 - g.test, 42
- *Topic **datasets**
 - antibiotics, 8
 - microorganisms, 61
 - microorganisms.codes, 63
 - microorganisms.old, 64
 - septic_patients, 81
 - WHONET, 83
- *Topic **eucast**
 - eucast_rules, 28
- *Topic **filter_class**
 - filter_ab_class, 31
- *Topic **filter**
 - filter_ab_class, 31
- *Topic **first**
 - first_isolate, 33
- *Topic **frequency**
 - freq, 37
- *Topic **freq**
 - freq, 37
- *Topic **guess**
 - as.mo, 12
- *Topic **interpretive**
 - eucast_rules, 28
- *Topic **isolates**
 - count, 25
 - first_isolate, 33
 - portion, 72
- *Topic **isolate**
 - count, 25
 - first_isolate, 33
 - portion, 72
- *Topic **lancefield**
 - as.mo, 12
- *Topic **mic**
 - as.mic, 11
- *Topic **mo**
 - as.mo, 12
- *Topic **reading**
 - eucast_rules, 28
- *Topic **resistance**
 - count, 25
 - eucast_rules, 28
 - portion, 72
- *Topic **rsi_df**
 - portion, 72
- *Topic **rsi**
 - as.rsi, 17
 - count, 25
 - portion, 72
- *Topic **summarise**
 - freq, 37
- *Topic **summary**
 - freq, 37
- *Topic **susceptibility**
 - count, 25
 - portion, 72
 - %like%(like), 56
- ab_property, 3, 10

- abname, [3](#), [26](#), [47](#), [73](#), [81](#)
- age, [4](#), [5](#), [6](#)
- age_groups, [5](#), [5](#), [84](#)
- AMR, [7](#)
- anti_join_microorganisms (join), [51](#)
- antibiotics, [3](#), [4](#), [8](#), [10](#), [11](#), [20](#), [21](#), [26](#), [32](#), [47](#), [50](#), [73](#), [81](#), [84](#)
- as.atc, [3](#), [8](#), [10](#), [20](#), [21](#)
- as.character, [56](#)
- as.matrix, [76](#)
- as.mic, [11](#), [18](#)
- as.mo, [12](#), [24](#), [29](#), [34](#), [51](#), [53](#), [58](#), [61](#), [63–66](#), [69](#), [75](#), [81](#), [84](#)
- as.POSIXlt, [4](#)
- as.rsi, [12](#), [17](#), [26](#), [46](#), [47](#), [72](#), [73](#), [75](#), [80](#), [81](#), [84](#)
- atc, [21](#)
- atc (as.atc), [10](#)
- atc_certe (atc_property), [20](#)
- atc_name, [84](#)
- atc_name (atc_property), [20](#)
- atc_official (atc_property), [20](#)
- atc_online_ddd (atc_online_property), [19](#)
- atc_online_groups (atc_online_property), [19](#)
- atc_online_property, [19](#)
- atc_property, [20](#)
- atc_tradenames (atc_property), [20](#)
- atc_trivial_n1 (atc_property), [20](#)
- atc_umcg (atc_property), [20](#)
- availability, [22](#)
- boxplot.stats, [39](#)
- brmo (mdro), [57](#)
- browseURL, [66](#)
- catalogue_of_life, [23](#)
- catalogue_of_life_version, [16](#), [23](#), [24](#), [25](#), [62–64](#), [67](#)
- chisq.test, [42–44](#)
- clean_mo_history (as.mo), [12](#)
- Click here, [16](#), [23](#), [25](#), [62–64](#), [67](#)
- close, [76](#)
- connection, [76](#)
- count, [25](#), [73](#)
- count_all (count), [25](#)
- count_df, [46](#), [47](#)
- count_df (count), [25](#)
- count_I (count), [25](#)
- count_IR (count), [25](#)
- count_R (count), [25](#)
- count_S (count), [25](#)
- count_SI (count), [25](#)
- data.frame, [8](#), [38](#), [61](#), [63](#), [64](#), [81](#), [84](#)
- Encoding, [77](#)
- eucast_exceptional_phenotypes (mdro), [57](#)
- EUCAST_rules (eucast_rules), [28](#)
- eucast_rules, [28](#)
- facet_rsi (ggplot_rsi), [46](#)
- facet_wrap, [47](#)
- factor, [6](#)
- file, [75](#), [77](#)
- filter_1st_cephalosporins (filter_ab_class), [31](#)
- filter_2nd_cephalosporins (filter_ab_class), [31](#)
- filter_3rd_cephalosporins (filter_ab_class), [31](#)
- filter_4th_cephalosporins (filter_ab_class), [31](#)
- filter_ab_class, [31](#)
- filter_aminoglycosides (filter_ab_class), [31](#)
- filter_at, [32](#)
- filter_carbapenems (filter_ab_class), [31](#)
- filter_cephalosporins (filter_ab_class), [31](#)
- filter_first_isolate (first_isolate), [33](#)
- filter_first_weighted_isolate (first_isolate), [33](#)
- filter_fluoroquinolones (filter_ab_class), [31](#)
- filter_glycopeptides (filter_ab_class), [31](#)
- filter_macrolides (filter_ab_class), [31](#)
- filter_tetracyclines (filter_ab_class), [31](#)
- first_isolate, [33](#), [52–54](#), [73](#)
- fisher.test, [43](#), [44](#)
- fivenum, [39](#)
- freq, [37](#)
- frequency_tbl (freq), [37](#)
- full_join_microorganisms (join), [51](#)
- g.test, [42](#)

- geom_rsi (ggplot_rsi), 46
- geom_text, 47
- get_locale, 45, 66
- get_mo_source, 13
- get_mo_source (mo_source), 69
- getOption, 21, 26, 38, 45, 66, 73
- getwd, 75
- ggplot, 46
- ggplot_rsi, 46
- ggplot_rsi_predict
 - (resistance_predict), 77
- glm, 79
- grep, 56
- guess_ab_col, 29, 49, 53, 60
- header (freq), 37
- inner_join (join), 51
- inner_join_microorganisms (join), 51
- interpretive_reading (eucast_rules), 28
- is.atc (as.atc), 10
- is.mic (as.mic), 11
- is.mo (as.mo), 12
- is.rsi (as.rsi), 17
- join, 51, 51
- key_antibiotics, 34, 36, 52
- key_antibiotics_equal
 - (key_antibiotics), 52
- kurtosis, 55, 82
- labels_rsi_count (ggplot_rsi), 46
- left_join_microorganisms (join), 51
- like, 56
- list, 21
- lm, 79
- Long vectors, 56
- mad, 39
- make.names, 76
- max, 39
- mdro, 57
- mean, 39
- median, 39
- microorganisms, 9, 16, 24, 25, 51, 61, 63–67, 81
- microorganisms.codes, 63, 63
- microorganisms.old, 64
- min, 39
- mo, 29, 34, 53, 58
- mo (as.mo), 12
- mo_authors (mo_property), 65
- mo_class (mo_property), 65
- mo_failures (as.mo), 12
- mo_family (mo_property), 65
- mo_fullname (mo_property), 65
- mo_genus, 16, 69
- mo_genus (mo_property), 65
- mo_gramstain, 16, 45, 69
- mo_gramstain (mo_property), 65
- mo_kingdom (mo_property), 65
- mo_order (mo_property), 65
- mo_phylum (mo_property), 65
- mo_property, 13, 16, 63, 64, 65
- mo_rank (mo_property), 65
- mo_ref (mo_property), 65
- mo_renamed (as.mo), 12
- mo_shortname (mo_property), 65
- mo_source, 69
- mo_species (mo_property), 65
- mo_subspecies (mo_property), 65
- mo_taxonomy (mo_property), 65
- mo_type, 45, 84
- mo_type (mo_property), 65
- mo_uncertainties (as.mo), 12
- mo_url (mo_property), 65
- mo_year (mo_property), 65
- mrng (mdro), 57
- n_distinct, 26
- n_rsi (count), 25
- NA, 76
- options, 69
- p.symbol, 71
- plot.resistance_predict
 - (resistance_predict), 77
- portion, 26, 27, 72, 79, 80
- portion_df, 46, 47
- portion_df (portion), 72
- portion_I (portion), 72
- portion_IR, 22
- portion_IR (portion), 72
- portion_R (portion), 72
- portion_S (portion), 72
- portion_SI (portion), 72
- prettyNum, 39

`print.frequency_tbl (freq)`, 37

`quantile`, 39

`read.4D`, 75

`readRDS`, 69

regular expression, 56

`resistance_predict`, 77

`right_join_microorganisms (join)`, 51

`rsi`, 73, 80

`rsi_predict (resistance_predict)`, 77

`scale_brewer`, 47

`scale_continuous`, 47

`scale_rsi_colours (ggplot_rsi)`, 46

`scale_y_percent (ggplot_rsi)`, 46

`scan`, 76

`sd`, 39

`semi_join_microorganisms (join)`, 51

`septic_patients`, 81, 83

`set_mo_source`, 13, 63

`set_mo_source (mo_source)`, 69

skewness, 55, 82

`stdin`, 76

`summarise`, 25, 72

`summary`, 78

`table`, 38

`theme`, 47

`theme_rsi (ggplot_rsi)`, 46

`tibble`, 38

`tolower`, 3

`top_freq (freq)`, 37

`top_n`, 39

`url`, 76

WHOCC, 83

WHONET, 83