Package ‘AMR’

April 15, 2020

Version 1.1.0
Date 2020-04-15
Title Antimicrobial Resistance Analysis

Depends R (>= 3.1.0)
Imports backports, cleaner, crayon (>= 1.3.0), data.table (>= 1.9.0), dplyr (>= 0.7.0), ggplot2, knitr (>= 1.0.0), microbenchmark, pillar, R6, rlang (>= 0.3.1), tidyr (>= 1.0.0), vctrs (>= 0.2.4)
Suggests covr (>= 3.0.1), curl, readxl, rmarkdown, rstudioapi, rvest (>= 0.3.2), testthat (>= 1.0.2), xml2 (>= 1.0.0)

VignetteBuilder knitr


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

License GPL-2 | file LICENSE

Encoding UTF-8

LazyData true

RoxygenNote 7.1.0

NeedsCompilation no

Author Matthijs S. Berends [aut, cre]
(https://orcid.org/0000-0001-7620-1800),
Christian F. Luz [aut, ctb] (https://orcid.org/0000-0001-5809-5995),
Alexander W. Friedrich [aut, ths]
(https://orcid.org/0000-0003-4881-038X),
Bhanu N. M. Sinha [aut, ths] (https://orcid.org/0000-0003-1634-0010),
Casper J. Albers [aut, ths] (https://orcid.org/0000-0002-9213-6743),
Corinna Glasner [aut, ths] (https://orcid.org/0000-0003-1241-1328),
Judith M. Fonville [ctb],
Erwin E. A. Hassing [ctb],
Eric H. L. C. M. Hazenberg [ctb],
Annick Lenglet [ctb],
Bart C. Meijer [ctb],
Sofia Ny [ctb],
Dennis Souverein [ctb]

Maintainer Matthijs S. Berends <m.s.berends@umcg.nl>
Repository CRAN
Date/Publication 2020-04-15 14:00:19 UTC

R topics documented:

ab_property ................................................. 3
age .......................................................... 5
age_groups .................................................. 7
AMR ......................................................... 9
antibiotics .................................................. 10
as.ab ......................................................... 13
as.disk ....................................................... 15
as.mic ....................................................... 16
as.mo ......................................................... 17
as.rsi ......................................................... 23
atc_online_property ......................................... 27
availability .................................................. 29
bug_drug_combinations ...................................... 30
catalogue_of_life ........................................... 32
catalogue_of_life_version ................................... 34
count ......................................................... 35
eucast_rules ................................................ 39
example_isolates ............................................ 43
example_isolates_unclean .................................... 44
filter_ab_class ............................................. 45
first_isolate ............................................... 47
g.test ....................................................... 51
ggplot_pca ................................................ 55
ggplot_rsi ................................................ 57
guess_ab_col ............................................... 62
join .......................................................... 64
key_antibiotics ............................................. 65
kurtosis ..................................................... 69
lifecycle .................................................... 70
like .......................................................... 71
mdro ......................................................... 73
microorganisms ............................................. 77
microorganisms.codes ....................................... 79
microorganisms.old ......................................... 80
Description

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

Usage

ab_name(x, language = get_locale(), tolower = FALSE, ...)
ab_atc(x, ...)
ab_cid(x, ...)
ab_synonyms(x, ...)
ab_tradenames(x, ...)
ab_group(x, language = get_locale(), ...)
ab_atc_group1(x, language = get_locale(), ...)
ab_atc_group2(x, language = get_locale(), ...)
ab_loinc(x, ...)
ab_ddd(x, administration = "oral", units = FALSE, ...)
ab_info(x, language = get_locale(), ...)
ab_property(x, property = "name", language = get_locale(), ...)
Arguments

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

Details

All output will be translated where possible.

Value

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

Maturing lifecycle

The lifecycle of this function is maturing. The unlying code of a maturing function has been roughed out, but finer details might still change. We will strive to maintain backward compatibility, but the function needs wider usage and more extensive testing in order to optimise the unlying code.

Source

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

WHONET 2019 software: [http://www.whonet.org/software.html](http://www.whonet.org/software.html)


Read more on our website!

On our website [https://msberends.gitlab.io/AMR](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

# all properties:
ab_name("AMX")  # "Amoxicillin"
ab_atc("AMX")   # J01CA04 (ATC code from the WHO)
ab_cid("AMX")   # 33613 (Compound ID from PubChem)
ab_synonyms("AMX") # a list with brand names of amoxicillin
ab_tradenames("AMX") # same
ab_group("AMX")  # "Beta-lactams/penicillins"
ab_atc_group1("AMX") # "Beta-lactam antibacterials, penicillins"
ab_atc_group2("AMX") # "Penicillins with extended spectrum"

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

# defined daily doses (DDD)
ab_ddd("AMX", "oral")  # 1
ab_ddd("AMX", "oral", units = TRUE) # "g"
ab_ddd("AMX", "iv")    # 1
ab_ddd("AMX", "iv", units = TRUE) # "g"

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

# all ab_* functions use as.ab() internally, so you can go from 'any' to 'any':
ab_atc("AMP")    # ATC code of AMP (ampicillin)
ab_group("J01CA01") # Drug group of ampicillins ATC code
ab_loinc("ampicillin") # LOINC codes of ampicillin
ab_name("21066-6") # "Ampicillin" (using LOINC)
ab_name(6249)     # "Ampicillin" (using CID)
ab_name("J01CA01") # "Ampicillin" (using ATC)

# spelling from different languages and dyslexia are no problem
ab_atc("ceftriaxon")
ab_atc("cephtriaxone")
ab_atc("cephthriaxone")
ab_atc("seephthriaaksone")

---

**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(), exact = FALSE, na.rm = FALSE)

Arguments

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

Value

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

Stable lifecycle

The lifecycle of this function is stable. In a stable function, we are largely happy with the underlying code, and major changes are unlikely. This means that the underlying code will generally evolve by adding new arguments; we will avoid removing arguments or changing the meaning of existing arguments.

If the underlying code needs breaking changes, they will occur gradually. To begin with, the function or argument will be deprecated; it will continue to work but will emit a message informing you of the change. Next, typically after at least one newly released version on CRAN, the message will be transformed to an error.

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

To split ages into groups, use the age_groups() function.

Examples

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

df
### Description

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

### Usage

```r
age_groups(x, split_at = c(12, 25, 55, 75), na.rm = FALSE)
```

### Arguments

- **x**: age, e.g. calculated with `age()`.
- **split_at**: values to split `x` at, defaults to age groups 0-11, 12-24, 25-54, 55-74 and 75+. See Details.
- **na.rm**: a logical to indicate whether missing values should be removed.

### Details

To split ages, the input for the `split_at` parameter can be:

- A numeric vector. A vector of e.g. `c(10, 20)` will split on 0-9, 10-19 and 20+. A value of only 50 will split on 0-49 and 50+. The default is to split on young children (0-11), youth (12-24), young adults (25-54), middle-aged adults (55-74) and elderly (75+).

- A character:
  - "children" or "kids", equivalent of: `c(0, 1, 2, 4, 6, 13, 18)`. This will split on 0, 1, 2-3, 4-5, 6-12, 13-17 and 18+.
  - "elderly" or "seniors", equivalent of: `c(65, 75, 85)`. This will split on 0-64, 65-74, 75-84, 85+.
  - "fives", equivalent of: `1:20 * 5`. This will split on 0-4, 5-9, 10-14, ..., 90-94, 95-99, 100+.
  - "tens", equivalent of: `1:10 * 10`. This will split on 0-9, 10-19, 20-29, ..., 80-89, 90-99, 100+.

### Value

Ordered `factor`

### Stable lifecycle

The lifecycle of this function is **stable**. In a stable function, we are largely happy with the underlying code, and major changes are unlikely. This means that the underlying code will generally evolve by adding new arguments; we will avoid removing arguments or changing the meaning of existing arguments.
If the underlying code needs breaking changes, they will occur gradually. To begin with, the function or argument will be deprecated; it will continue to work but will emit a message informing you of the change. Next, typically after at least one newly released version on CRAN, the message will be transformed to an error.

**Read more on our website!**

On our website [https://msberends.gitlab.io/AMR](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**

To determine ages, based on one or more reference dates, use the `age()` function.

**Examples**

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

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

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

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

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

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

## Not run:
# resistance of ciprofloxacin per age group
library(dplyr)
example_isolates %>%
  filter_first_isolate() %>%
  filter(mo == as.m0("E. coli")) %>%
  group_by(age_group = age_groups(age)) %>%
  select(age_group, CIP) %>%
ggplot_rsi(x = "age_group")

## End(Not run)
```
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 the taxonomy of microorganisms, since the package contains all microbial (sub)species from the Catalogue of Life
- Interpreting raw MIC and disk diffusion values, based on the latest CLSI or EUCAST guidelines
- Determining first isolates to be used for AMR analysis
- Calculating antimicrobial resistance
- Determining multi-drug resistance (MDR) / multi-drug resistant organisms (MDRO)
- Calculating (empirical) susceptibility of both mono therapy and combination therapies
- 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, EARS-Net code, ATC code, PubChem code, defined daily dose or trade name)
- Plotting antimicrobial resistance
- Getting SNOMED codes of a microorganism, or get its name associated with a SNOMED code
- Getting LOINC codes of an antibiotic, or get its name associated with a LOINC code
- Machine reading the EUCAST and CLSI guidelines from 2011-2020 to translate MIC values and disk diffusion diameters to R/SI
- Principal component analysis for AMR
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
The Netherlands

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

antibiotics

Data sets with 557 antimicrobials

Description

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

Usage

antibiotics

antivirals

Format

For the antibiotics data set: a data.frame with 455 observations and 14 variables::

- ab
  Antibiotic ID as used in this package (like AMC), using the official EARS-Net (European Antimicrobial Resistance Surveillance Network) codes where available
- atc
  ATC code (Anatomical Therapeutic Chemical) as defined by the WHOCC, like J01CR02
- cid
  Compound ID as found in PubChem
antibiotics

- **name**
  Official name as used by WHONET/EARS-Net or the WHO
- **group**
  A short and concise group name, based on WHONET and WHOCC definitions
- **atc_group1**
  Official pharmacological subgroup (3rd level ATC code) as defined by the WHOCC, like "Macrolides, lincosamides and streptogramins"
- **atc_group2**
  Official chemical subgroup (4th level ATC code) as defined by the WHOCC, like "Macrolides"
- **abbr**
  List of abbreviations as used in many countries, also for antibiotic susceptibility testing (AST)
- **synonyms**
  Synonyms (often trade names) of a drug, as found in PubChem based on their compound ID
- **oral_ddd**
  Defined Daily Dose (DDD), oral treatment
- **oral_units**
  Units of oral_ddd
- **iv_ddd**
  Defined Daily Dose (DDD), parenteral treatment
- **iv_units**
  Units of iv_ddd
- **loinc**
  All LOINC codes (Logical Observation Identifiers Names and Codes) associated with the name of the antimicrobial agent. Use ab_loinc() to retrieve them quickly, see ab_property().

For the **antivirals** data set: a data.frame with 102 observations and 9 variables::

- **atc**
  ATC code (Anatomical Therapeutic Chemical) as defined by the WHOCC
- **cid**
  Compound ID as found in PubChem
- **name**
  Official name as used by WHONET/EARS-Net or the WHO
- **atc_group**
  Official pharmacological subgroup (3rd level ATC code) as defined by the WHOCC
- **synonyms**
  Synonyms (often trade names) of a drug, as found in PubChem based on their compound ID
- **oral_ddd**
  Defined Daily Dose (DDD), oral treatment
- **oral_units**
  Units of oral_ddd
- **iv_ddd**
  Defined Daily Dose (DDD), parenteral treatment
- **iv_units**
  Units of iv_ddd

An object of class data.frame with 102 rows and 9 columns.
Details

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

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

Direct download:
These data sets are available as ‘flat files’ for use even without R - you can find the files here:


Files in R format (which data structure) can be found here:

- https://gitlab.com/msberends/AMR/raw/master/data/antibiotics.rda

WHOCC


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.

NOTE: The WHOCC copyright does not allow use for commercial purposes, unlike any other info from this package. See https://www.whocc.no/copyright_disclaimer/.

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 (WHOCC):
https://www.whocc.no/atc_ddd_index/

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


See Also

microorganisms
as.ab

Transform to antibiotic ID

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

Usage
as.ab(x, ...)

is.ab(x)

Arguments
x character vector to determine to antibiotic ID
...
arguments passed on to internal functions

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

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

Source
World Health Organization (WHO) Collaborating Centre for Drug Statistics Methodology: https://www.whocc.no/atc_ddd_index/
WHONET 2019 software: http://www.whonet.org/software.html

Maturing lifecycle
The lifecycle of this function is maturing. The underlying code of a maturing function has been roughed out, but finer details might still change. We will strive to maintain backward compatibility, but the function needs wider usage and more extensive testing in order to optimise the underlying code.
WHOCC


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.

**NOTE:** The WHOCC copyright does not allow use for commercial purposes, unlike any other info from this package. See [https://www.whocc.no/copyright_disclaimer/](https://www.whocc.no/copyright_disclaimer/).

Read more on our website!

On our website [https://msberends.gitlab.io/AMR](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

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

# spelling from different languages and dyslexia are no problem
ab_atc("ceftriaxon")
ab_atc("cephtriaxone") # small spelling error
ab_atc("cephthiaxone") # or a bit more severe
ab_atc("seephthriaaksone") # and even this works

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

Description

This transforms a vector to a new class disk, which is a growth zone size (around an antibiotic disk) in millimetres between 6 and 50.

Usage

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

is.disk(x)

Arguments

x

vector

na.rm

a logical indicating whether missing values should be removed

Details

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

Value

An integer with additional new class disk

Stable lifecycle

The lifecycle of this function is stable. In a stable function, we are largely happy with the underlying code, and major changes are unlikely. This means that the underlying code will generally evolve by adding new arguments; we will avoid removing arguments or changing the meaning of existing arguments.

If the underlying code needs breaking changes, they will occur gradually. To begin with, the function or argument will be deprecated; it will continue to work but will emit a message informing you of the change. Next, typically after at least one newly released version on CRAN, the message will be transformed to an error.

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

# transform existing disk zones to the 'disk' class
library(dplyr)

df <- data.frame(microorganism = "E. coli",
                 AMP = 20,
                 CIP = 14,
                 GEN = 18,
                 TOB = 16)

df <- df %>% mutate_at(vars(AMP:TOB), as.disk)

df

# interpret disk values, see ?as.rsi
as.rsi(x = as.disk(18),
       mo = "Strep pneu", # 'mo' will be coerced with as.mo()
       ab = "ampicillin", # and 'ab' with as.ab()
       guideline = "EUCAST")

as.rsi(df)

as.mic

Class 'mic'

Description

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

Usage

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

is.mic(x)

Arguments

x vector

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

Details

To interpret MIC values as RSI values, use as.rsi() on MIC values. It supports guidelines from EUCAST and CLSI.

Value

Ordered factor with new class mic
Stable lifecycle

The lifecycle of this function is stable. In a stable function, we are largely happy with the underlying code, and major changes are unlikely. This means that the underlying code will generally evolve by adding new arguments; we will avoid removing arguments or changing the meaning of existing arguments.

If the underlying code needs breaking changes, they will occur gradually. To begin with, the function or argument will be deprecated; it will continue to work but will emit a message informing you of the change. Next, typically after at least one newly released version on CRAN, the message will be transformed to an error.

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

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

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

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

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

---

**as.mo**

*Transform to microorganism ID*
Description

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

Usage

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

`is.mo(x)`

`mo_failures()`

`mo_uncertainties()`

`mo_renamed()`

Arguments

- `x` a character vector or a `data.frame` with one or two columns
- `Becker` a logical to indicate whether Staphylococci should be categorised into coagulase-negative Staphylococci ("CoNS") and coagulase-positive Staphylococci ("CoPS") instead of their own species, according to Karsten Becker et al. (1,2). Note that this does not include species that were newly named after these publications, like S. caeli.
  
  This excludes Staphylococcus aureus at default, use Becker = "all" to also categorise S. aureus as "CoPS".
- `Lancefield` a logical to indicate whether beta-haemolytic Streptococci should be categorised into Lancefield groups instead of their own species, according to Rebecca C. Lancefield (3). These Streptococci will be categorised in their first group, e.g. Streptococcus dysgalactiae will be group C, although officially it was also categorised into groups G and L.
  
  This excludes Enterococci at default (who are in group D), use Lancefield = "all" to also categorise all Enterococci as group D.
- `allow_uncertain` a number between 0 (or "none") and 3 (or "all"), or TRUE (= 2) or FALSE (= 0) to indicate whether the input should be checked for less probable results, please 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 microorganism ID from this package (class: mo) typically looks like these examples:

<table>
<thead>
<tr>
<th>Code</th>
<th>Full name</th>
</tr>
</thead>
<tbody>
<tr>
<td>B_KLBSL</td>
<td>Klebsiella</td>
</tr>
<tr>
<td>B_KLBSL_PNMN</td>
<td>Klebsiella pneumoniae</td>
</tr>
<tr>
<td>B_KLBSL_PNMN_RHNS</td>
<td>Klebsiella pneumoniae rhinoscleromatis</td>
</tr>
</tbody>
</table>

| | | | ---> subspecies, a 4-5 letter acronym |
| | | ----> species, a 4-5 letter acronym    |
| | ---> genus, a 5-7 letter acronym        |

----> taxonomic kingdom: A (Archaea), AN (Animalia), B (Bacteria), C (Chromista), F (Fungi), P (Protozoa)

Values that cannot be coerced will be considered 'unknown' and will get the MO code UNKNOWN. Use the mo_* 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).
The as.mo() function uses several coercion rules for fast and logical results. It assesses the input matching criteria in the following order:

1. Human pathogenic prevalence: the function starts with more prevalent microorganisms, followed by less prevalent ones;
2. Taxonomic kingdom: the function starts with determining Bacteria, then Fungi, then Protozoa, then others;
3. Breakdown of input values to identify possible matches.

This will lead to the effect that e.g. "E. coli" (a microorganism highly prevalent in humans) will return the microbial ID of Escherichia coli and not Entamoeba coli (a microorganism less prevalent in humans), although the latter would alphabetically come first.

Coping with uncertain results:
In addition, the as.mo() function can differentiate four levels of uncertainty to guess valid results:

- Uncertainty level 0: no additional rules are applied;
- Uncertainty level 1: allow previously accepted (but now invalid) taxonomic names and minor spelling errors;
- Uncertainty level 2: allow all of level 1, strip values between brackets, inverse the words of the input, strip off text elements from the end keeping at least two elements;
- Uncertainty level 3: allow all of level 1 and 2, strip off text elements from the end, allow any part of a taxonomic name.
This leads to e.g.:

• "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* \( (B_{\text{STRPT\_GRPB}}) \) 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* \( (B_{\text{STPHY\_AURS}}) \) 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* \( (B_{\text{NESSR\_GNRR}}) \) needs review.

The level of uncertainty can be set using the argument allow_uncertain. The default is allow_uncertain = TRUE, which is equal to uncertainty level 2. Using allow_uncertain = FALSE is equal to uncertainty level 0 and will skip all rules. You can also use e.g. as.mo(...,allow_uncertain = 1) to only allow up to level 1 uncertainty.

There are three helper functions that can be run after then as.mo() function:

• Use mo_uncertainties() to get a data.frame with all values that were coerced to a valid value, but with uncertainty. The output contains a score, that is calculated as \( (n - 0.5 \times L) / n \), where \( n \) is the number of characters of the returned full name of the microorganism, and \( L \) is the Levenshtein distance between that full name and the user input.

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

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

**Microbial prevalence of pathogens in humans:**

The intelligent rules consider the prevalence of microorganisms in humans grouped into three groups, which is available as the prevalence columns in the microorganisms and microorganisms.old data sets. The grouping into prevalence groups is based on experience from several microbiological laboratories in the Netherlands in conjunction with international reports on pathogen prevalence.

Group 1 (most prevalent microorganisms) consists of all microorganisms where the taxonomic class is Gammaproteobacteria or where the taxonomic genus is *Enterococcus*, *Staphylococcus* or *Streptococcus*. This group consequently contains all common Gram-negative bacteria, such as *Pseudomonas* and *Legionella* and all species within the order Enterobacteriales.

Group 2 consists of all microorganisms where the taxonomic phylum is Proteobacteria, Firmicutes, Actinobacteria or Sarcomastigophora, or where the taxonomic genus is *Aspergillus*, *Bacteroides*, *Candida*, *Capncycphaga*, *Chryseobacterium*, *Cryptococcus*, *Elisabethkingia*, *Flavobacterium*, *Fusobacterium*, *Giardia*, *Leptotrichia*, *Mycoplasma*, *Prevotella*, *Rhodotorula*, *Treponema*, *Trichophyton* or *Ureaplasma*.

Group 3 (least prevalent microorganisms) consists of all other microorganisms.

**Value**

A character vector with class mo

**Source**


**Stable lifecycle**

The lifecycle of this function is stable. In a stable function, we are largely happy with the unlying code, and major changes are unlikely. This means that the unlying code will generally evolve by adding new arguments; we will avoid removing arguments or changing the meaning of existing arguments.

If the unlying code needs breaking changes, they will occur gradually. To begin with, the function or argument will be deprecated; it will continue to work but will emit an message informing you of the change. Next, typically after at least one newly released version on CRAN, the message will be transformed to an error.

**Catalogue of Life**

This package contains the complete taxonomic tree of almost all microorganisms (~70,000 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. Check which version of the Catalogue of Life 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_AURS", 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("Zthafilokkookus oureuz")  # handles incorrect spelling
as.mo("MRSA")  # Methicillin Resistant S. aureus
as.mo("VISA")  # Vancomycin Intermediate S. aureus
as.mo("VRSA")  # Vancomycin Resistant S. aureus
as.mo(22242419)  # Catalogue of Life ID
as.mo(115329001)  # SNOMED CT code

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

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

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

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

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

# and can even contain 2 columns, which is convenient for genus/species combinations:
df$mo <- df %>%
  select(genus, species) %>%
  as.mo()

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

## End(Not run)
as.rsi

Class 'rsi'

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

Usage
as.rsi(x, ...)

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

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

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

is.rsi(x)

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

Arguments

x vector of values (for class mic: an MIC value in mg/L, for class disk: a disk diffusion radius in millimetres)

... parameters passed on to methods

mo any (vector of) text that can be coerced to a valid microorganism code with as.mo()
ab  any (vector of) text that can be coerced to a valid antimicrobial code with \texttt{as.ab()}

guideline defaults to the latest included EUCAST guideline, run \texttt{unique(rsi_translation$guideline)} for all options

uti (Urinary Tract Infection) A vector with \texttt{logicals} (TRUE or FALSE) to specify whether a UTI specific interpretation from the guideline should be chosen. For using \texttt{as.rsi()} on a \texttt{data.frame}, this can also be a column containing \texttt{logicals} or when left blank, the data set will be search for a 'specimen' and rows containing 'urin' in that column will be regarded isolates from a UTI. See \texttt{Examples}.

col_mo column name of the IDs of the microorganisms (see \texttt{as.mo()}), defaults to the first column of class \texttt{mo}. Values will be coerced using \texttt{as.mo()}.  

threshold maximum fraction of invalid antimicrobial interpretations of \(x\), please see \texttt{Examples}

Details

Run \texttt{unique(rsi_translation$guideline)} for a list of all supported guidelines. The repository of this package contains this machine readable version of these guidelines.

These guidelines are machine readable, since \texttt{.}

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

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

Value

Ordered factor with new class \texttt{rsi}

Interpretation of R and S/I

In 2019, the European Committee on Antimicrobial Susceptibility Testing (EUCAST) has decided to change the definitions of susceptibility testing categories R and S/I as shown below (http://www.eucast.org/newsiandr/).

- **R = Resistant**
  A microorganism is categorised as \textit{Resistant} when there is a high likelihood of therapeutic failure even when there is increased exposure. Exposure is a function of how the mode of administration, dose, dosing interval, infusion time, as well as distribution and excretion of the antimicrobial agent will influence the infecting organism at the site of infection.

- **S = Susceptible**
  A microorganism is categorised as \textit{Susceptible, standard dosing regimen}, when there is a high likelihood of therapeutic success using a standard dosing regimen of the agent.

- **I = Increased exposure, but still susceptible**
  A microorganism is categorised as \textit{Susceptible, Increased exposure} when there is a high likelihood of therapeutic success because exposure to the agent is increased by adjusting the dosing regimen or by its concentration at the site of infection.
This AMR package honours this new insight. Use `susceptibility()` (equal to `proportion_SI()`) to determine antimicrobial susceptibility and `count_susceptible()` (equal to `count_SI()`) to count susceptible isolates.

**Stable lifecycle**

The lifecycle of this function is **stable**. In a stable function, we are largely happy with the unlying code, and major changes are unlikely. This means that the unlying code will generally evolve by adding new arguments; we will avoid removing arguments or changing the meaning of existing arguments.

If the unlying code needs breaking changes, they will occur gradually. To begin with, the function or argument will be deprecated; it will continue to work but will emit a message informing you of the change. Next, typically after at least one newly released version on CRAN, the message will be transformed to an error.

**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**

```r
# For INTERPRETING disk diffusion and MIC values -----------------------

# a whole data set, even with combined MIC values and disk zones
df <- data.frame(microorganism = "E. coli",
                 AMP = as.mic(8),
                 CIP = as.mic(0.256),
                 GEN = as.disk(18),
                 TOB = as.disk(16),
                 NIT = as.mic(32))

as.rsi(df)

# the dplyr way
library(dplyr)

df %>%
  mutate_at(vars(AMP:TOB), as.rsi, mo = "E. coli")

df %>%
  mutate_at(vars(AMP:TOB), as.rsi, mo = .$microorganism)

# to include information about urinary tract infections (UTI)
data.frame(mo = "E. coli",
           NIT = c("<= 2", 32),
```


as.rsi

```r
from_the_bladder = c(TRUE, FALSE)) %>%
as.rsi(uti = "from_the_bladder")

data.frame(mo = "E. coli",
    NIT = c("<= 2", 32),
    specimen = c("urine", "blood")) %>%
as.rsi() # automatically determines urine isolates

df %>%
    mutate_at(vars(AMP:NIT), as.rsi, mo = "E. coli", uti = TRUE)

# for single values
as.rsi(x = as.mic(2),
    mo = as.mo("S. pneumoniae"),
    ab = "AMP",
    guideline = "EUCAST")
as.rsi(x = as.disk(18),
    mo = "Strep pneu", # `mo` will be coerced with as.mo()
    ab = "ampicillin", # and `ab` with as.ab()
    guideline = "EUCAST")

# For CLEANING existing R/SI values -------------------------------

as.rsi(c("S", "I", "R", "A", "B", "C"))
as.rsi("<= 0.002; S") # will return "S"

rsi_data <- as.rsi(c(rep("S", 474), rep("I", 36), rep("R", 370)))
is.rsi(rsi_data)
plot(rsi_data) # for percentages
barchart(rsi_data) # for frequencies
freq(rsi_data) # frequency table with informative header

library(dplyr)
example_isolates %>%
    mutate_at(vars(PEN:RIF), as.rsi)

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

# note: from dplyr 1.0.0 on, this will be:
# example_isolates %>%
# mutate(across(is.rsi.eligible, as.rsi))

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

```r
atc_online_property(
  atc_code,
  property,
  administration = "O",
  url = "https://www.whocc.no/atc_ddd_index/?code=%s&showdescription=no"
)
```

```r
atc_online_groups(atc_code, ...)
```

```r
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)

**Questioning lifecycle**

The lifecycle of this function is questioning. We are no longer convinced that this function is the optimal approach (but we do not know yet what a better approach would be), or whether this function should be in our AMR package at all.

**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] "ANTIMICROBIALS FOR SYSTEMIC USE"
# [2] "ANTIBACTERIALS FOR SYSTEMIC USE"
# [3] "BETA-LACTAM ANTIBACTERIALS, PENICILLINS"
# [4] "Penicillins with extended spectrum"
```
availability  

Check availability of columns

Description

Easy check for data availability of all columns in a data set. This makes it easy to get an idea of which antimicrobial combinations can be used for calculation with e.g. `susceptibility()` and `resistance()`.

Usage

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

Arguments

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

Details

The function returns a `data.frame` with columns "resistant" and "visual_resistance". The values in that columns are calculated with `resistance()`.

Value

`data.frame` with column names of `tbl` as row names

Stable lifecycle

The lifecycle of this function is stable. In a stable function, we are largely happy with the underlying code, and major changes are unlikely. This means that the underlying code will generally evolve by adding new arguments; we will avoid removing arguments or changing the meaning of existing arguments.

If the underlying code needs breaking changes, they will occur gradually. To begin with, the function or argument will be deprecated; it will continue to work but will emit a message informing you of the change. Next, typically after at least one newly released version on CRAN, the message will be transformed to an error.

Read more on our website!

On our website [https://msberends.gitlab.io/AMR](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(example_isolates)

library(dplyr)
example_isolates %>% availability()

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

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

---

**bug_drug_combinations**  Determine bug-drug combinations

Description

Determine antimicrobial resistance (AMR) of all bug-drug combinations in your data set where at least 30 (default) isolates are available per species. Use *format()* on the result to prettify it to a publicable/printable format, see Examples.

Usage

```r
bug_drug_combinations(x, col_mo = NULL, FUN = mo_shortname, ...)
```

## S3 method for class 'bug_drug_combinations'

```r
format(
  x,
  translate_ab = "name (ab, atc)",
  language = get_locale(),
  minimum = 30,
  combine_SI = TRUE,
  combine_IR = FALSE,
  add_ab_group = TRUE,
  remove_intrinsic_resistant = FALSE,
  decimal.mark =getOption("OutDec"),
  big.mark = ifelse(decimal.mark == ",", ".", ",", ","),
  ...
)
```

Arguments

- **x**: data with antibiotic columns, like e.g. AMX and AMC
- **col_mo**: column name of the IDs of the microorganisms (see `as.mo()`), defaults to the first column of class `mo`. Values will be coerced using `as.mo()`.
fun_drug_combinations

FUN
the function to call on the mo column to transform the microorganism IDs, defaults to mo_shortname()

... arguments passed on to FUN

translate_ab
a character of length 1 containing column names of the antibiotics data set

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

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.

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

combine_IR
logical to indicate whether values R and I should be summed

add_ab_group
logical to indicate where the group of the antimicrobials must be included as a first column

remove_intrinsic_resistant
logical to indicate that rows with 100% resistance for all tested antimicrobials must be removed from the table

decimal.mark
the character to be used to indicate the numeric decimal point.

big.mark
character; if not empty used as mark between every big.interval decimals before (hence big) the decimal point.

Details

The function format() calculates the resistance per bug-drug combination. Use combine_IR = FALSE (default) to test R vs. S+I and combine_IR = TRUE to test R+I vs. S.

The language of the output can be overwritten with options(AMR_locale), please see translate.

Value

The function bug_drug_combinations() returns a data.frame with columns "mo", "ab", "S", "I", "R" and "total".

Stable lifecycle

The lifecycle of this function is stable. In a stable function, we are largely happy with the unlying code, and major changes are unlikely. This means that the unlying code will generally evolve by adding new arguments; we will avoid removing arguments or changing the meaning of existing arguments.

If the unlying code needs breaking changes, they will occur gradually. To begin with, the function or argument will be deprecated; it will continue to work but will emit an message informing you of the change. Next, typically after at least one newly released version on CRAN, the message will be transformed to an error.
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


Examples

```r
x <- bug_drug_combinations(example_isolates)
x
format(x, translate_ab = "name (atc)")

# Use FUN to change to transformation of microorganism codes
x <- bug_drug_combinations(example_isolates,
    FUN = mo_gramstain)

x <- bug_drug_combinations(example_isolates,
    FUN = function(x) ifelse(x == "B_ESCHR_COLI",
        "E. coli",
        "Others"))
```

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 (~70,000 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. Check which version of the Catalogue of Life was included in this package with catalogue_of_life_version().
Included taxa

Included are:

- All ~61,000 (sub)species from the kingdoms of Archaea, Bacteria, Chromista and Protozoa
- All ~8,500 (sub)species from these orders of the kingdom of Fungi: Eurotiales, Microascales, Mucorales, Onygenales, Pneumocystales, Saccharomycetales, Schizosaccharomycetales and Tremellales. The kingdom of Fungi is a very large taxon with almost 300,000 different (sub)species, of which most are not microbial (but rather macroscopic, like mushrooms). Because of this, not all fungi fit the scope of this package and including everything would tremendously slow down our algorithms too. By only including the aforementioned taxonomic orders, the most relevant fungi are covered (like all species of Aspergillus, Candida, Cryptococcus, Histplasma, Pneumocystis, Saccharomyces and Trichophyton).
- All ~150 (sub)species from ~100 other relevant genera from the kingdom of Animalia (like Strongyloides and Taenia)
- All ~23,000 previously accepted names of all included (sub)species (these were 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 1.9 million species. The Catalogue of Life is used to support the major biodiversity and conservation information services such as the Global Biodiversity Information Facility (GBIF), Encyclopedia of Life (EoL) and the International Union for Conservation of Nature Red List. It is recognised by the Convention on Biological Diversity as a significant component of the Global Taxonomy Initiative and a contribution to Target 1 of the Global Strategy for Plant Conservation.

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

Read more on our website!

On our website https://msberends.gitlab.io/AMR you can find a 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_shortname("Chlamydia psittaci")
# Note: 'Chlamydia psittaci' (Page, 1968) was renamed
# 'Chlamydophila psittaci' (Everett et al., 1999)
# [1] "C. psittaci"

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

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

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

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

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

catalogue_of_life_version

Version info of included Catalogue of Life

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

Usage
catalogue_of_life_version()

Details
For DSMZ, see microorganisms.

Value
a list, which prints in pretty format

Catalogue of Life
This package contains the complete taxonomic tree of almost all microorganisms (~70,000 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. Check which version of the Catalogue of Life was included in this package with 

```r
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

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

## count

### Count available isolates

**Description**

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

- `count_resistant()` should be used to count resistant isolates, `count_susceptible()` should be used to count susceptible isolates.

**Usage**

- `count_resistant(..., only_all_tested = FALSE)`
- `count_susceptible(..., only_all_tested = FALSE)`
- `count_R(..., only_all_tested = FALSE)`
- `count_IR(..., only_all_tested = FALSE)`
- `count_I(..., only_all_tested = FALSE)`
- `count_SI(..., only_all_tested = FALSE)`
- `count_S(..., only_all_tested = FALSE)`
- `count_all(..., only_all_tested = FALSE)`
n_rsi(..., only_all_tested = FALSE)

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

Arguments

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

only_all_tested (for combination therapies, i.e. using more than one variable for ...): a logical to indicate that isolates must be tested for all antibiotics, see section Combination therapy below

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

translate_ab a column name of the antibiotics data set to translate the antibiotic abbreviations to, using ab_property() language language of the returned text, defaults to system language (see get_locale()) and can also be set with getOption("AMR_locale"). Use language = NULL or language = "" to prevent translation.

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

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

Details

These functions are meant to count isolates. Use the resistance()/susceptibility() functions to calculate microbial resistance/susceptibility.

The function count_resistant() is equal to the function count_R(). The function count_susceptible() is equal to the function count_SI().

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

The function count_df() takes any variable from data that has an rsi class (created with as.rsi()) and counts the number of S’s, I’s and R’s. The function rsi_df() works exactly like count_df(), but adds the percentage of S, I and R.
Value

An integer

Stable lifecycle

The lifecycle of this function is stable. In a stable function, we are largely happy with the underlying code, and major changes are unlikely. This means that the underlying code will generally evolve by adding new arguments; we will avoid removing arguments or changing the meaning of existing arguments.

If the underlying code needs breaking changes, they will occur gradually. To begin with, the function or argument will be deprecated; it will continue to work but will emit a message informing you of the change. Next, typically after at least one newly released version on CRAN, the message will be transformed to an error.

Interpretation of R and S/I

In 2019, the European Committee on Antimicrobial Susceptibility Testing (EUCAST) has decided to change the definitions of susceptibility testing categories R and S/I as shown below (http://www.eucast.org/newsandr/).

- **R = Resistant**
  A microorganism is categorised as *Resistant* when there is a high likelihood of therapeutic failure even when there is increased exposure. Exposure is a function of how the mode of administration, dose, dosing interval, infusion time, as well as distribution and excretion of the antimicrobial agent will influence the infecting organism at the site of infection.

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

- **I = Increased exposure, but still susceptible**
  A microorganism is categorised as *Susceptible, Increased exposure* when there is a high likelihood of therapeutic success because exposure to the agent is increased by adjusting the dosing regimen or by its concentration at the site of infection.

This AMR package honours this new insight. Use `susceptibility()` (equal to `proportion_SI()`) to determine antimicrobial susceptibility and `count_susceptible()` (equal to `count_SI()`) to count susceptible isolates.

Combination therapy

When using more than one variable for ... (= combination therapy)), use `only_all_tested` to only count isolates that are tested for all antibiotics/variables that you test them for. See this example for two antibiotics, Drug A and Drug B, about how `susceptibility()` works to calculate the %SI:

<table>
<thead>
<tr>
<th>only_all_tested = FALSE</th>
<th>only_all_tested = TRUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Drug A</td>
<td>Drug B</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Drug A</th>
<th>Drug B</th>
<th>include as numerator</th>
<th>denominator</th>
<th>include as numerator</th>
<th>denominator</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---
Please note that, in combination therapies, for `only_all_tested = TRUE` applies that:

\[
\text{count}_S() + \text{count}_I() + \text{count}_R() = \text{count}_\text{all}()
\]

\[
\text{proportion}_S() + \text{proportion}_I() + \text{proportion}_R() = 1
\]

and that, in combination therapies, for `only_all_tested = FALSE` applies that:

\[
\text{count}_S() + \text{count}_I() + \text{count}_R() \geq \text{count}_\text{all}()
\]

\[
\text{proportion}_S() + \text{proportion}_I() + \text{proportion}_R() \geq 1
\]

Using `only_all_tested` has no impact when only using one antibiotic as input.

Read more on our website!

On our website [https://msberends.gitlab.io/AMR](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

`proportion_*` to calculate microbial resistance and susceptibility.

Examples

```r
# example_isolates is a data set available in the AMR package.
?example_isolates

count_resistant(example_isolates$AMX) # counts "R"
count_susceptible(example_isolates$AMX) # counts "S" and "I"
count_all(example_isolates$AMX) # counts "S", "I" and "R"

# be more specific
count_S(example_isolates$AMX)
count_SI(example_isolates$AMX)
count_I(example_isolates$AMX)
count_IR(example_isolates$AMX)
count_R(example_isolates$AMX)

# Count all available isolates
```
count_all(example_isolates$AMX)
n_rsi(example_isolates$AMX)

# n_rsi() is an alias of count_all().
# Since it counts all available isolates, you can
# calculate back to count e.g. susceptible isolates.
# These results are the same:
count_susceptible(example_isolates$AMX)
susceptibility(example_isolates$AMX) * n_rsi(example_isolates$AMX)

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

# Count co-resistance between amoxicillin/clav acid and gentamicin,
# so we can see that combination therapy does a lot more than mono therapy.
# Please mind that `susceptibility()` calculates percentages right away instead.
example_isolates %>% count_susceptible(AMC) # 1433
example_isolates %>% count_all(AMC) # 1879

example_isolates %>% count_susceptible(GEN) # 1399
example_isolates %>% count_all(GEN) # 1855

example_isolates %>% count_susceptible(AMC, GEN) # 1764
example_isolates %>% count_all(AMC, GEN) # 1936

# Get number of S+I vs. R immediately of selected columns
example_isolates %>%
  select(AMX, CIP) %>%
count_df(translate = FALSE)

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

eucast_rules

Apply 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.
To improve the interpretation of the antibiogram before EUCAST rules are applied, some non-EUCAST rules are applied at default, see Details.

Usage

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

Arguments

x  data with antibiotic columns, like e.g. AMX and AMC

col_mo  column name of the IDs of the microorganisms (see as.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 turn Verbose mode on and off (default is off). In Verbose mode, the function does not apply rules to the data, but instead returns a data set in logbook form with extensive info about which rows and columns would be effected and in which way.

...  column name of an antibiotic, please see section Antibiotics below

Details

Note: This function does not translate MIC values to RSI values. Use as.rsi() for that.

Note: When ampicillin (AMP, J01CA01) is not available but amoxicillin (AMX, J01CA04) is, the latter will be used for all rules where there is a dependency on ampicillin. These drugs are interchangeable when it comes to expression of antimicrobial resistance.

Before further processing, some non-EUCAST rules are applied to improve the efficacy of the EUCAST rules. These non-EUCAST rules, that are applied to all isolates, are:

- Inherit amoxicillin (AMX) from ampicillin (AMP), where amoxicillin (AMX) is unavailable;
- Inherit ampicillin (AMP) from amoxicillin (AMX), where ampicillin (AMP) is unavailable;
- Set amoxicillin (AMX) = R where amoxicillin/clavulanic acid (AMC) = R;
- Set piperacillin (PIP) = R where piperacillin/tazobactam (TZP) = R;
- Set trimethoprim (TMP) = R where trimethoprim/sulfamethoxazole (SXT) = R;
- Set amoxicillin/clavulanic acid (AMC) = S where amoxicillin (AMX) = S;
- Set piperacillin/tazobactam (TZP) = S where piperacillin (PIP) = S;
- Set trimethoprim/sulfamethoxazole (SXT) = S where trimethoprim (TMP) = S.
To not use these rules, please use `eucast_rules(..., rules = c("breakpoints","expert"))`.


### Value

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

### Antibiotics

To define antibiotics column names, leave as is to determine it automatically with `guess_ab_col()` or input a text (case-insensitive), or use `NULL` to skip a column (e.g. `TIC = NULL` to skip ticarcillin).

Manually defined but non-existing columns will be skipped with a warning.

The following antibiotics are used for the functions `eucast_rules()` and `mdro()`.

These are shown below in the format `antimicrobial ID: name (ATC code)`, sorted by name:

- **AMK**: amikacin (J01GB06)
- **AMX**: amoxicillin (J01CA04)
- **AMC**: amoxicillin/clavulanic acid (J01CR02)
- **AMP**: ampicillin (J01CA01)
- **SAM**: ampicillin/sulbactam (J01CR01)
- **AZM**: azithromycin (J01FA10)
- **AZL**: azlocillin (J01CA09)
- **ATM**: aztreonam (J01DF01)
- **CAP**: capreomycin (J04AB30)
- **RID**: cefaloridine (J01DB02)
- **CZO**: cefazolin (J01DB04)
- **CTX**: cefotaxime (J01DD01)
- **CTT**: cefotetan (J01DC05)
- **FOX**: cefoxitin (J01DC01)
- **CPT**: ceftaroline (J01DI02)
- **CAZ**: ceftazidime (J01DD02)
- **CRO**: ceftriaxone (J01DD04)
- **CXM**: cefuroxime (J01DC02)
- **CED**: cephradine (J01DB09)
- **CHL**: chloramphenicol (J01BA01)
- **CIP**: ciprofloxacin (J01MA02)
- **CLR**: clarithromycin (J01FA09)
- **CLI**: clindamycin (J01FF01)
- **COL**: colistin (J01XB01)
- **DAP**: daptomycin (J01XX08)
- **DOR**: doripenem (J01DH04)
- **DOX**: doxycycline (J01AA02)
- **ETP**: ertapenem (J01DH03)
- **ERY**: erythromycin (J01FA01)
- **ETH**: ethambutol (J04AK02)
- **FLC**: flucloxacillin (J01CF05)
- **FOS**: fosfomycin (J01XX01)
- **FUS**: fusidic acid (J01XC01)
- **GAT**: gatifloxacin (J01MA01)
- **GEN**: gentamicin (J01GB03)
- **GEH**: gentamicin-high (no ATC code)
- **IMI**: imipenem (J01DH51)
- **INH**: isoniazid (J04AC01)
- **KAN**: kanamycin (J01GB04)
- **LEV**: levofloxacin (J01MA02)
- **LIN**: lincomycin (J01FA01)
- **LNZ**: linezolid (J01XX08)
- **MEM**: meropenem (J01DH02)
- **MTR**: metronidazole (J01XD01)
- **MEZ**: mezlocillin (J01CA10)
- **MNB**: minocycline (J01MB02)
- **MXF**: moxifloxacin (J01MA14)
- **NAL**: nalidixic acid (J01MB02)
- **NEO**: neomycin (J01GB05)
- **NET**: netilmicin (J01GB07)
- **NIT**: nitrofurantoin (J01XE01)
- **NOX**: novobiocin (J01XX05)
- **OFX**: ofloxacin (J01MA01)
- **OXA**: oxacillin (J01CF04)
- **PEN**: penicillin G (J01CE01)
- **PIP**: piperacillin (J01CA12)
- **TZIP**: piperacillin/tazobactam (J01CR05)
- **PLB**: polymyxin B (J01XB02)
- **PRI**: pristinamycin (J01FG01)
- **PZA**: pyrazinamide (J04AK01)
- **QDA**: quinupristin/dalfopristin (J01FG02)
- **RIB**: rifabutin (J04AB04)
- **rifampicin (J04AB02)
- **RFP**: rifapentine (J04AB05)
- **RXT**: roxithromycin (J01FA06)
- **SIS**: sisomicin (J01GB08)
- **STH**: streptomycin-high (no ATC code)
- **TEC**: teicoplanin (J01XA02)
- **TLY**: telavancin (J01XX01)
- **TIC**: ticarcillin (J01CA13)
- **TCC**: ticarcillin/clavulanic acid (J01CR03)
- **TGC**: tigecycline (J01AA12)
- **TOB**: tobramycin (J01GB01)
- **TMP**: trimethoprim (J01EA01)
- **SXT**: trimethoprim/sulfamethoxazole (J01EE01)
- **VAN**: vancomycin (J01XA01).

### Maturing lifecycle

The lifecycle of this function is maturing. The unlying code of a maturing function has been roughed out, but finer details might still change. We will strive to maintain backward compatibility, but the function needs wider usage and more extensive testing in order to optimise the unlying code.
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://doi.org/10.1111/j.1469-0691.2011.03703.x
  http://www.eucast.org/fileadmin/src/media/PDFs/EUCAST_files/Breakpoint_tables/v_9.0_Breakpoint_Tables.xlsx

Examples

```r
a <- data.frame(mo = c("Staphylococcus aureus", "Enterococcus faecalis", "Escherichia coli", "Klebsiella pneumoniae", "Pseudomonas aeruginosa"), VN = "-", AMX = "-", COL = "-", CAZ = "-", CXM = "-", PEN = "S", FOX = "S", stringsAsFactors = FALSE)
a
# mo VN AMX COL CAZ CXM PEN FOX
# 1 Staphylococcus aureus - - - - - S S
# 2 Enterococcus faecalis - - - - - S S
# 3 Escherichia coli - - - - - S S
# 4 Klebsiella pneumoniae - - - - - S S
# 5 Pseudomonas aeruginosa - - - - - S S

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

b
# mo VN AMX COL CAZ CXM PEN FOX
```

example_isolates

# 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 data.frame
# with 18 rows, containing all details about the transformations:
c <- eucast_rules(a, verbose = TRUE)

example_isolates  Data set with 2,000 example isolates

Description
A data set containing 2,000 microbial isolates with their full antibiograms. The data set reflects reality and can be used to practice AMR analysis. For examples, please read the tutorial on our website.

Usage
example_isolates

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
example_isolates_unclean

Data set with unclean data

Description

A data set containing 3,000 microbial isolates that are not cleaned up and consequently not ready for AMR analysis. This data set can be used for practice.

Usage

example_isolates_unclean

Format

A data.frame with 3,000 observations and 8 variables:

- patient_id
  ID of the patient
- date
  date of receipt at the laboratory
- hospital
  ID of the hospital, from A to C
- bacteria
  info about microorganism that can be transformed with as.mo(), see also microorganisms
- AMX:GEN
  4 different antibiotics that have to be transformed with 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.
**filter_ab_class**  
*Filter isolates on result in antibiotic class*

**Description**

Filter isolates on results in specific antibiotic variables based on their antibiotic class. This makes it easy to filter on isolates that were tested for e.g. any aminoglycoside.

**Usage**

```r
filter_ab_class(x, ab_class, result = NULL, scope = "any", ...)  
filter_aminoglycosides(x, result = NULL, scope = "any", ...)  
filter_carbapenems(x, result = NULL, scope = "any", ...)  
filtercephalosporins(x, result = NULL, scope = "any", ...)  
filter_1stcephalosporins(x, result = NULL, scope = "any", ...)  
filter_2ndcephalosporins(x, result = NULL, scope = "any", ...)  
filter_3rdcephalosporins(x, result = NULL, scope = "any", ...)  
filter_4thcephalosporins(x, result = NULL, scope = "any", ...)  
filter_5thcephalosporins(x, result = NULL, scope = "any", ...)  
filter_fluoroquinolones(x, result = NULL, scope = "any", ...)  
filter_glycopeptides(x, result = NULL, scope = "any", ...)  
filter_macrolides(x, result = NULL, scope = "any", ...)  
filter_tetracyclines(x, result = NULL, scope = "any", ...)  
```

**Arguments**

- **x**  
  A data set

- **ab_class**  
  An antimicrobial class, like "carbapenems", as can be found in `antibiotics$group`

- **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` from the `dplyr` package
Details

The group column in antibiotics data set will be searched for ab_class (case-insensitive). If no results are found, the atc_group1 and atc_group2 columns will be searched. Next, x will be checked for column names with a value in any abbreviations, codes or official names found in the antibiotics data set.

Stable lifecycle

The lifecycle of this function is stable. In a stable function, we are largely happy with the underlying code, and major changes are unlikely. This means that the underlying code will generally evolve by adding new arguments; we will avoid removing arguments or changing the meaning of existing arguments.

If the underlying code needs breaking changes, they will occur gradually. To begin with, the function or argument will be deprecated; it will continue to work but will emit a message informing you of the change. Next, typically after at least one newly released version on CRAN, the message will be transformed to an error.

Examples

```r
library(dplyr)

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

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

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

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

# filter on isolates that show resistance to any aminoglycoside and any fluoroquinolone
example_isolates %>%
  filter_aminoglycosides("R") %>%
  filter_fluoroquinolones("R")

# filter on isolates that show resistance to all aminoglycosides and all fluoroquinolones
example_isolates %>%
  filter_aminoglycosides("R", "all") %>%
  filter_fluoroquinolones("R", "all")
```
**Determine first (weighted) isolates**

**Description**

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

**Usage**

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

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

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

x a data.frame containing isolates.
col_date column name of the result date (or date that is was received on the lab), defaults to the first column of with a date class
col_patient_id column name of the unique IDs of the patients, defaults to the first column that starts with 'patient' or 'patid' (case insensitive)
col_mo column name of the IDs of the microorganisms (see as.mo()), defaults to the first column of class mo. Values will be coerced using as.mo().
col_testcode column name of the test codes. Use col_testcode = NULL to not exclude certain test codes (like test codes for screening). In that case testcodes_exclude will be ignored.
col_specimen column name of the specimen type or group
col_icu column name of the logicals (TRUE/FALSE) whether a ward or department is an Intensive Care Unit (ICU)
col_keyantibiotics column name of the key antibiotics to determine first weighted isolates, see key_antibiotics(). Defaults to the first column that starts with 'key' followed by 'ab' or 'antibiotics' (case insensitive). Use col_keyantibiotics = FALSE to prevent this.
episode_days episode in days after which a genus/species combination will be determined as 'first isolate' again. The default of 365 days is based on the guideline by CLSI, see Source.
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
include_unknown logical to determine whether 'unknown' microorganisms should be included too, i.e. microbial code "UNKNOWN", which defaults to FALSE. For WHONET users, this means that all records with organism code "con" (contamination) will be excluded at default. Isolates with a microbial ID of NA will always be excluded as first isolate.
...
parameters passed on to the first_isolate() function
WHY THIS IS SO IMPORTANT

To conduct an analysis of antimicrobial resistance, you should only include the first isolate of every patient per episode (ref). 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.

All isolates with a microbial ID of NA will be excluded as first isolate.

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

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

The function filter_first_weighted_isolate() is essentially equal to:

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

Value

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. Read more about this in the key_antibiotics() function.

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, which default to 2, an isolate will be (re)selected as a first weighted isolate.
Stable lifecycle

The lifecycle of this function is stable. In a stable function, we are largely happy with the unlying code, and major changes are unlikely. This means that the unlying code will generally evolve by adding new arguments; we will avoid removing arguments or changing the meaning of existing arguments.

If the unlying code needs breaking changes, they will occur gradually. To begin with, the function or argument will be deprecated; it will continue to work but will emit a message informing you of the change. Next, typically after at least one newly released version on CRAN, the message will be transformed to an error.

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:


See Also

key_antibiotics()

Examples

# 'example_isolates' is a dataset available in the AMR package.
# See ?example_isolates.

library(dplyr)
# Filter on first isolates:
example_isolates %>%
  mutate(first_isolate = first_isolate(.)) %>%
  filter(first_isolate == TRUE)

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

B <- example_isolates %>%
  filter_first_weighted_isolate() %>% # the 1st isolate filter
  group_by(hospital_id) %>%
  summarise(count = n_rsi(GEN), # gentamicin availability
            resistance = resistance(GEN)) # gentamicin resistance
Have a look at A and B.
B is more reliable because every isolate is counted only once.
Gentamicin resistance in hospital D appears to be 3.7% higher than
when you (erroneously) would have used all isolates for analysis.

## OTHER EXAMPLES:

## Not run:

Short-hand versions:
example_isolates %>%
  filter_first_isolate()

example_isolates %>%
  filter_first_weighted_isolate()

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

x$first_isolate <- first_isolate(x)

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

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

## End(Not run)

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**

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

**Arguments**

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

Details

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

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

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

In the contingency table case simulation is done by random sampling from the set of all contingency tables with given marginals, and works only if the marginals are strictly positive. Note that this is not the usual sampling situation assumed for a chi-squared test (like the G-test) but rather that for Fisher’s exact test.

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

**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:

$$ G = 2 \times \sum (x \times \log(x/E)) $$

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

```r
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.

**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, $(\text{observed} - \text{expected}) / \sqrt{\text{expected}}$.
- **stdres** standardized residuals, $(\text{observed} - \text{expected}) / \sqrt{\text{V}}$, 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).

**Questioning lifecycle**
The lifecycle of this function is questioning. We are no longer convinced that this function is the optimal approach (but we do not know yet what a better approach would be), or whether this function should be in our AMR package at all.
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

The code for this function is identical to that of `chisq.test()`, except that:

- The calculation of the statistic was changed to \( 2 \times \sum(x \times \log(x/E)) \)
- 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


See Also

`chisq.test()`

Examples

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

x <- c(772, 1611, 737)
G <- g.test(x, p = c(1, 2, 1) / 4)
# G$p.value = 0.12574.
# There is no significant difference from a 1:2:1 ratio.
# Meaning: resistance controlled by a single gene with two co-dominant
# alleles, is plausible.

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

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

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

---

**ggplot_pca**  

**PCA biplot with ggplot2**

---

**Description**

Produces a ggplot2 variant of a so-called biplot for PCA (principal component analysis), but is more flexible and more appealing than the base R `biplot()` function.

**Usage**

```r
ggplot_pca(
  x,
  choices = 1:2,
  scale = TRUE,
  pc.biplot = TRUE,
  labels = NULL,
  labels_textsize = 3,
  labels_text_placement = 1.5,
  groups = NULL,
  ellipse = TRUE,
  ellipse_prob = 0.68,
  ellipse_size = 0.5,
  ellipse_alpha = 0.5,
  points_size = 2,
  points_alpha = 0.25,
  arrows = TRUE,
  arrows_colour = "darkblue",
  arrows_size = 0.5,
  arrows_textsize = 3,
  arrows_alpha = 0.75,
  base_textsize = 10,
  ...
)
```

**Arguments**

- `x` an object returned by `pca()`, `prcomp()` or `princomp()`
- `choices` length 2 vector specifying the components to plot. Only the default is a biplot in the strict sense.
scale
The variables are scaled by $\lambda^\text{scale}$ and the observations are scaled by $\lambda^{(1-\text{scale})}$ where $\lambda$ are the singular values as computed by \texttt{princomp}. Normally $0 \leq \text{scale} \leq 1$, and a warning will be issued if the specified \text{scale} is outside this range.

\texttt{pc.biplot}
If true, use what Gabriel (1971) refers to as a "principal component biplot", with $\lambda = 1$ and observations scaled up by $\sqrt{n}$ and variables scaled down by $\sqrt{n}$. Then inner products between variables approximate covariances and distances between observations approximate Mahalanobis distance.

\texttt{labels}
an optional vector of labels for the observations. If set, the labels will be placed below their respective points. When using the \texttt{pca()} function as input for \texttt{x}, this will be determined automatically based on the attribute \texttt{non_numeric_cols}, see \texttt{pca()}.

\texttt{labels.textsize}
the size of the text used for the labels

\texttt{labels.text_placement}
adjustment factor the placement of the variable names (>=1 means further away from the arrow head)

\texttt{groups}
an optional vector of groups for the labels, with the same length as \texttt{labels}. If set, the points and labels will be coloured according to these groups. When using the \texttt{pca()} function as input for \texttt{x}, this will be determined automatically based on the attribute \texttt{non_numeric_cols}, see \texttt{pca()}.

\texttt{ellipse}
a logical to indicate whether a normal data ellipse should be drawn for each group (set with \texttt{groups})

\texttt{ellipse.prob}
statistical size of the ellipse in normal probability

\texttt{ellipse.size}
the size of the ellipse line

\texttt{ellipse.alpha}
the alpha (transparency) of the ellipse line

\texttt{points.size}
the size of the points

\texttt{points.alpha}
the alpha (transparency) of the points

\texttt{arrows}
a logical to indicate whether arrows should be drawn

\texttt{arrows.colour}
the colour of the arrow and their text

\texttt{arrows.size}
the size (thickness) of the arrow lines

\texttt{arrows.textsize}
the size of the text at the end of the arrows

\texttt{arrows.alpha}
the alpha (transparency) of the arrows and their text

\texttt{base.textsize}
the text size for all plot elements except the labels and arrows

... Parameters passed on to functions

**Details**
The colours for labels and points can be changed by adding another scale layer for colour, like \texttt{scale_colour_viridis_d()} or \texttt{scale_colour_brewer()}.
Maturing lifecycle

The lifecycle of this function is maturing. The underlying code of a maturing function has been roughed out, but finer details might still change. We will strive to maintain backward compatibility, but the function needs wider usage and more extensive testing in order to optimise the underlying code.

Source

The ggplot_pca() function is based on the ggbiplot() function from the ggbiplot package by Vince Vu, as found on GitHub: https://github.com/vqv/ggbiplot (retrieved: 2 March 2020, their latest commit: 7325e88; 12 February 2015).

As per their GPL-2 licence that demands documentation of code changes, the changes made based on the source code were:

1. Rewritten code to remove the dependency on packages plyr, scales and grid
2. Parametrised more options, like arrow and ellipse settings
3. Added total amount of explained variance as a caption in the plot
4. Cleaned all syntax based on the lintr package and added integrity checks
5. Updated documentation

Examples

```r
# 'example_isolates' is a dataset available in the AMR package.
# See ?example_isolates.

# See ?pca for more info about Principal Component Analysis (PCA).
library(dplyr)
pca_model <- example_isolates %>%
  filter(mo_genus(mo) == "Staphylococcus") %>%
  group_by(species = mo_shortname(mo)) %>%
  summarise_if (is.rsi, resistance) %>%
  pca(FLC, AMC, CXM, GEN, TOB, TMP, SXT, CIP, TEC, TCY, ERY)

# old
biplot(pca_model)

# new
ggplot_pca(pca_model)
```

Description

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

ggplot_rsi(
  data,
  position = NULL,
  x = "antibiotic",
  fill = "interpretation",
  facet = NULL,
  breaks = seq(0, 1, 0.1),
  limits = NULL,
  translate_ab = "name",
  combine_SI = TRUE,
  combine_IR = FALSE,
  language = get_locale(),
  nrow = NULL,
  colours = c(S = "#61a8ff", SI = "#61a8ff", I = "#61f7ff", IR = "#ff6961", R = "#ff6961"),
  datalabels = TRUE,
  datalabels.size = 2.5,
  datalabels.colour = "gray15",
  title = NULL,
  subtitle = NULL,
  caption = NULL,
  x.title = "Antimicrobial",
  y.title = "Proportion",
  ...
)

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

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

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

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

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

Arguments

data a data.frame with column(s) of class rsi (see as.rsi())
position position adjustment of bars, either "fill", "stack" or "dodge"
x variable to show on x axis, either "antibiotic" (default) or "interpretation" or a grouping variable
fill variable to categorise using the plots legend, either "antibiotic" (default) or "interpretation" or a grouping variable
facet variable to split plots by, either "interpretation" (default) or "antibiotic" or a grouping variable
breaks numeric vector of positions
limits numeric vector of length two providing limits of the scale, use NA to refer to the existing minimum or maximum
translate_ab a column name of the antibiotics data set to translate the antibiotic abbreviations to, using ab_property()
combine_SI a logical to indicate whether all values of S and I must be merged into one, so the output only consists of S+I vs. R (susceptible vs. resistant). This used to be the parameter combine_IR, but this now follows the redefinition by EUCAST about the interpretation of I (increased exposure) in 2019, see section 'Interpretation of S, I and R' below. Default is TRUE.
combine_IR a logical to indicate whether all values of I and R must be merged into one, so the output only consists of S vs. I+R (susceptible vs. non-susceptible). This is outdated, see parameter combine_SI.
language language of the returned text, defaults to system language (see get_locale()) and can also be set with getOption("AMR_locale"). Use language = NULL or language = "" to prevent translation.
nrow (when using facet) number of rows
colours a named vector with colours for the bars. The names must be one or more of: S, SI, I, IR, R or be FALSE to use default [ggplot2]::ggplot() colours.
datalabels show datalabels using labels_rsi_count()
datalabels.size size of the datalabels
datalabels.colour colour of the datalabels
title    text to show as title of the plot
subtitle    text to show as subtitle of the plot
caption    text to show as caption of the plot
x.title    text to show as x axis description
y.title    text to show as y axis description
...    other parameters passed on to geom_rsi()

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

The functions:
geom_rsi() will take any variable from the data that has an rsi class (created with as.rsi()) using rsi_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 ggplot2::facet_wrap().
scale_y_percent() transforms the y axis to a 0 to 100% range using ggplot2::scale_continuous().
scale_rsi_colours() sets colours to the bars: pastel blue for S, pastel turquoise for I and pastel red for R, using ggplot2::scale_brewer().
theme_rsi() is a [ggplot2 theme]ggplot2::theme() with minimal distraction.
labels_rsi_count() print datalabels on the bars with percentage and amount of isolates using ggplot2::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.

Maturing lifecycle
The lifecycle of this function is maturing. The underlying code of a maturing function has been roughed out, but finer details might still change. We will strive to maintain backward compatibility, but the function needs wider usage and more extensive testing in order to optimise the underlying code.

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(example_isolates %>% select(AMX, NIT, FOS, TMP, CIP)) +
  geom_rsi()

# prettify the plot using some additional functions:
df <- example_isolates %>% select(AMX, NIT, FOS, TMP, CIP)
ggplot(df) +
ggplot_rsi()

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

# get only proportions and no counts:
exmaple_isolates %>%
select(AMX, NIT, FOS, TMP, CIP) %>%
ggplot_rsi(datalabels = FALSE)

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

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

## Not run:

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

# for colourblind mode, use divergent colours from the viridis package:
exmaple_isolates %>%
select(AMX, NIT, FOS, TMP, CIP) %>%
ggplot_rsi() + scale_fill_viridis_d()
# a shorter version which also adjusts data label colours:
exmaple_isolates %>%
select(AMX, NIT, FOS, TMP, CIP) %>%
ggplot_rsi(colours = FALSE)
# it also supports groups (don't forget to use the group var on `x` or `facet'
):
example_isolates %>%
  select(hospital_id, AMX, NIT, FOS, TMP, CIP) %>%
  group_by(hospital_id) %>%
  ggplot_rsi(x = "hospital_id",
             facet = "antibiotic",
             nrow = 1,
             title = "AMR of Anti-UTI Drugs Per Hospital",
             x.title = "Hospital",
             datalabels = FALSE)

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

## End(Not run)

---

**guess_ab_col**

**Guess antibiotic column**

**Description**

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

**Usage**

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

Arguments

- **x**: a `data.frame`
- **search_string**: a text to search `x` for, will be checked with `as.ab()` if this value is not a column in `x`
- **verbose**: a logical to indicate whether additional info should be printed

Details

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

Value

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

Maturing lifecycle

The **lifecycle** of this function is **maturing**. The unlying code of a maturing function has been roughed out, but finer details might still change. We will strive to maintain backward compatibility, but the function needs wider usage and more extensive testing in order to optimise the unlying code.

Read more on our website!

On our website [https://msberends.gitlab.io/AMR](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

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

guess_ab_col(df, "amoxicillin")
# [1] "amox"

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

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

# WHONET codes
df <- data.frame(AMP_ND10 = "R",
                 AMC_ED20 = "S")

guess_ab_col(df, "ampicillin")
# [1] "AMP_ND10"

guess_ab_col(df, "J01CR02")
# [1] "AMC_ED20"

guess_ab_col(df, as.ab("augmentin"))
```
# Longer names take precedence:

```r
df <- data.frame(AMP_ED2 = "S",
                 AMP_ED20 = "S")
guess_ab_col(df, "ampicillin")
# [1] "AMP_ED20"
```

### join

Join a table with `microorganisms`

**Description**

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

**Usage**

```r
inner_join_microorganisms(x, by = NULL, suffix = c("2", ","), ...)
left_join_microorganisms(x, by = NULL, suffix = c("2", ","), ...)
right_join_microorganisms(x, by = NULL, suffix = c("2", ","), ...)
full_join_microorganisms(x, by = NULL, suffix = c("2", ","), ...)
semi_join_microorganisms(x, by = NULL, ...)
anti_join_microorganisms(x, by = NULL, ...)
```

**Arguments**

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

**Details**

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

The lifecycle of this function is stable. In a stable function, we are largely happy with the unlying code, and major changes are unlikely. This means that the unlying code will generally evolve by adding new arguments; we will avoid removing arguments or changing the meaning of existing arguments.

If the unlying code needs breaking changes, they will occur gradually. To begin with, the function or argument will be deprecated; it will continue to work but will emit a message informing you of the change. Next, typically after at least one newly released version on CRAN, the message will be transformed to an error.

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

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

```r
colnames(df)
df_joined <- left_join_microorganisms(df, "bacteria")
colnames(df_joined)
```

key_antibiotics  | Key antibiotics for first weighted isolates

Description

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

Usage

```r
key_antibiotics(
  x, col_mo = NULL,
```
universal_1 = guess_ab_col(x, "amoxicillin"),
universal_2 = guess_ab_col(x, "amoxicillin/clavulanic acid"),
universal_3 = guess_ab_col(x, "cefuroxime"),
universal_4 = guess_ab_col(x, "piperacillin/tazobactam"),
universal_5 = guess_ab_col(x, "ciprofloxacin"),
universal_6 = guess_ab_col(x, "trimethoprim/sulfamethoxazole"),
GramPos_1 = guess_ab_col(x, "vancomycin"),
GramPos_2 = guess_ab_col(x, "teicoplanin"),
GramPos_3 = guess_ab_col(x, "tetracycline"),
GramPos_4 = guess_ab_col(x, "erythromycin"),
GramPos_5 = guess_ab_col(x, "oxacillin"),
GramPos_6 = guess_ab_col(x, "rifampin"),
GramNeg_1 = guess_ab_col(x, "gentamicin"),
GramNeg_2 = guess_ab_col(x, "tobramycin"),
GramNeg_3 = guess_ab_col(x, "colistin"),
GramNeg_4 = guess_ab_col(x, "cefotaxime"),
GramNeg_5 = guess_ab_col(x, "ceftazidime"),
GramNeg_6 = guess_ab_col(x, "meropenem"),
warnings = TRUE,
...
}

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

Arguments

x
  table with antibiotics coloms, like AMX or amox

col_mo
  column name of the IDs of the microorganisms (see as.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
**key_antibiotics**

```r
y, z characters to compare
type type to determine weighed isolates; can be "keyantibiotics" or "points", see Details
ignore_I logical to determine whether antibiotic interpretations with "I" will be ignored when type = "keyantibiotics", see Details
points_threshold points until the comparison of key antibiotics will lead to inclusion of an isolate when type = "points", see Details
info print progress
```

**Details**

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

At default, the antibiotics that are used for **Gram-positive bacteria** are:

- Amoxicillin
- Amoxicillin/clavulanic acid
- Cefuroxime
- Piperacillin/tazobactam
- Ciprofloxacin
- Trimethoprim/sulfamethoxazole
- Vancomycin
- Teicoplanin
- Tetracycline
- Erythromycin
- Oxacillin
- Rifampin

At default the antibiotics that are used for **Gram-negative bacteria** are:

- Amoxicillin
- Amoxicillin/clavulanic acid
- Cefuroxime
- Piperacillin/tazobactam
- Ciprofloxacin
- Trimethoprim/sulfamethoxazole
- Gentamicin
key_antibiotics

- Tobramycin
- Colistin
- Cefotaxime
- Ceftazidime
- Meropenem

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

Stable lifecycle

The lifecycle of this function is stable. In a stable function, we are largely happy with the underlying code, and major changes are unlikely. This means that the underlying code will generally evolve by adding new arguments; we will avoid removing arguments or changing the meaning of existing arguments.

If the underlying code needs breaking changes, they will occur gradually. To begin with, the function or argument will be deprecated; it will continue to work but will emit a message informing you of the change. Next, typically after at least one newly released version on CRAN, the message will be transformed to an error.

Key antibiotics

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

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

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

Read more on our website!

On our website https://msberends.gitlab.io/AMR you can find a 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

# `example_isolates` is a dataset available in the AMR package.
# See ?example_isolates.

library(dplyr)
# set key antibiotics to a new variable
my_patients <- example_isolates %>%
  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 <- "SSSRRS.R..S"
strainB <- "SSSRRR$RRRSSS"

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.

Questioning lifecycle

The *lifecycle* of this function is *questioning*. We are no longer convinced that this function is the optimal approach (but we do not know yet what a better approach would be), or whether this function should be in our *AMR* package at all.

Read more on our website!

On our website [https://msberends.gitlab.io/AMR](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()**

---

**lifecycle**

*Lifecycles of functions in the AMR package*

Description

Our functions are categorised using the *lifecycle circle* of the *tidyverse* as found on [www.tidyverse.org/lifecycle](http://www.tidyverse.org/lifecycle).

This page contains a section for every lifecycle (with text borrowed from the aforementioned *tidyverse* website), so they can be used in the manual pages of our functions.

Experimental lifecycle

The *lifecycle* of this function is *experimental*. An experimental function is in the very early stages of development. The unlying code might be changing frequently as we rapidly iterate and explore variations in search of the best fit. Experimental functions might be removed without deprecation, so you are generally best off waiting until a function is more mature before you use it in production code. Experimental functions will not be included in releases we submit to CRAN, since they have not yet matured enough.

Maturing lifecycle

The *lifecycle* of this function is *maturing*. The unlying code of a maturing function has been roughed out, but finer details might still change. We will strive to maintain backward compatibility, but the function needs wider usage and more extensive testing in order to optimise the unlying code.
Stable lifecycle

The lifecycle of this function is **stable**. In a stable function, we are largely happy with the underlying code, and major changes are unlikely. This means that the underlying code will generally evolve by adding new arguments; we will avoid removing arguments or changing the meaning of existing arguments.

If the underlying code needs breaking changes, they will occur gradually. To begin with, the function or argument will be deprecated; it will continue to work but will emit a message informing you of the change. Next, typically after at least one newly released version on CRAN, the message will be transformed to an error.

Retired lifecycle

The lifecycle of this function is **retired**. A retired function is no longer under active development, and (if appropriate) a better alternative is available. We will only make the necessary changes to ensure that retired functions remain available. No new arguments will be added, and only the most critical bugs will be fixed.

Archived lifecycle

The lifecycle of this function is **archived**. The development of an archived function has ended, and it is no longer available in future package versions.

Dormant lifecycle

The lifecycle of this function is **dormant**. A dormant function is currently not under active development and has not reached a stable phase. We might return to it in the future. As with experimental functions, you are best off waiting until a function is more mature before you use it in production code.

Questioning lifecycle

The lifecycle of this function is **questioning**. We are no longer convinced that this function is the optimal approach (but we do not know yet what a better approach would be), or whether this function should be in our AMR package at all.

---

**like**

*Pattern Matching*

**Description**

Convenient wrapper around `grep()` to match a pattern: `x %like% pattern`. It always returns a logical vector and is always case-insensitive (use `x %like_case% pattern` for case-sensitive matching). Also, `pattern` can be as long as `x` to compare items of each index in both vectors, or they both can have the same length to iterate over all cases.
Usage

like(x, pattern, ignore.case = TRUE)

x %like% pattern

x %like_case% pattern

Arguments

x               a character vector where matches are sought, or an object which can be coerced by `as.character()` to a character vector.

pattern        a character string containing a regular expression (or character string for `fixed = TRUE`) to be matched in the given character vector. Coerced by `as.character()` to a character string if possible. If a character vector of length 2 or more is supplied, the first element is used with a warning.

ignore.case    if FALSE, the pattern matching is case sensitive and if TRUE, case is ignored during matching.

Details

When running a regular expression fails, these functions try again with base::grepl(..., perl = TRUE).

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

Stable lifecycle

The lifecycle of this function is stable. In a stable function, we are largely happy with the underlying code, and major changes are unlikely. This means that the underlying code will generally evolve by adding new arguments; we will avoid removing arguments or changing the meaning of existing arguments.

If the underlying code needs breaking changes, they will occur gradually. To begin with, the function or argument will be deprecated; it will continue to work but will emit a message informing you of the change. Next, typically after at least one newly released version on CRAN, the message will be transformed to an error.

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.
mdro

Determine multidrug-resistant organisms (MDRO)

Description

Determine which isolates are multidrug-resistant organisms (MDRO) according to international and national guidelines.

Usage

mdro(
  x,
  guideline = "CMI2012",
  col_mo = NULL,
  info = interactive(),
  pct_required_classes = 0.5,
  combine_SI = TRUE,
)
mdro

verbose = FALSE,
...
)

brmo(x, guideline = "BRMO", ...)
mrgn(x, guideline = "MRGN", ...)
mdr_tb(x, guideline = "TB", ...)
mdr_cmi2012(x, guideline = "CMI2012", ...)
eucast_exceptional_phenotypes(x, guideline = "EUCAST", ...)

Arguments

x data with antibiotic columns, like e.g. AMX and AMC
guideline a specific guideline to follow. When left empty, the publication by Magiorakos et al. (2012, Clinical Microbiology and Infection) will be followed, please see Details.
col_mo column name of the IDs of the microorganisms (see as.mo()), defaults to the first column of class mo. Values will be coerced using as.mo().
info a logical to indicate whether progress should be printed to the console
pct_required_classes minimal required percentage of antimicrobial classes that must be available per isolate, rounded down. For example, with the default guideline, 17 antimicrobial classes must be available for S. aureus. Setting this pct_required_classes argument to 0.5 (default) means that for every S. aureus isolate at least 8 different classes must be available. Any lower number of available classes will return NA for that isolate.
combine_SI a logical to indicate whether all values of S and I must be merged into one, so resistance is only considered when isolates are R, not I. As this is the default behaviour of the mdro() function, it follows the redefinition by EUCAST about the interpretation of I (increased exposure) in 2019, see section 'Interpretation of S, I and R' below. When using combine_SI = FALSE, resistance is considered when isolates are R or I.
verbose a logical to turn Verbose mode on and off (default is off). In Verbose mode, the function does not return the MDRO results, but instead returns a data set in logbook form with extensive info about which isolates would be MDRO-positive, or why they are not.
...

column name of an antibiotic, please see section Antibiotics below

Details

For the pct_required_classes argument, values above 1 will be divided by 100. This is to support both fractions (0.75 or 3/4) and percentages (75).
Currently supported guidelines are (case-insensitive):
multidrug-resistant, extensively drug-resistant and pandrug-resistant bacteria: an international expert proposal for interim standard definitions for acquired resistance.” Clinical Microbiology and Infection (2012) (link)

- guideline = "EUCAST"
The European international guideline - EUCAST Expert Rules Version 3.1 "Intrinsic Resistance and Exceptional Phenotypes Tables" (link)

- guideline = "TB"
The international guideline for multi-drug resistant tuberculosis - World Health Organization "Companion handbook to the WHO guidelines for the programmatic management of drug-resistant tuberculosis" (link)

- guideline = "MRGN"

- guideline = "BRMO"
The Dutch national guideline - Rijksinstituut voor Volksgezondheid en Milieu "WIP-richtlijn BRMO (Bijzonder Resistente Micro-Organismen) (ZKH)" (link)

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

Note: Every test that involves the Enterobacteriaceae family, will internally be performed using its newly named order Enterobacterales, since the Enterobacteriaceae family has been taxonomically reclassified by Adeolu et al. in 2016. Before that, Enterobacteriaceae was the only family under the Enterobacteriales (with an i) order. All species under the old Enterobacteriaceae family are still under the new Enterobacterales (without an i) order, but divided into multiple families. The way tests are performed now by this mdro() function makes sure that results from before 2016 and after 2016 are identical.

Value

- CMI 2012 paper - function mdr_cmi2012() or mdro():
  Ordered factor with levels Negative < Multi-drug-resistant (MDR) < Extensively drug-resistant (XDR) < Pandrug-resistant (PDR)
- TB guideline - function mdr_tb() or mdro(..., guideline = "TB"):
  Ordered factor with levels Negative < Mono-resistant < Poly-resistant < Multi-drug-resistant < Extensively drug-resistant
- German guideline - function mrgn() or mdro(..., guideline = "MRGN"):
  Ordered factor with levels Negative < 3MRGN < 4MRGN
- Everything else:
  Ordered factor with levels Negative < Positive, unconfirmed < Positive. The value "Positive, unconfirmed" means that, according to the guideline, it is not entirely sure if the isolate is multi-drug resistant and this should be confirmed with additional (e.g. molecular) tests

Maturing lifecycle

The lifecycle of this function is maturing. The unlying code of a maturing function has been roughed out, but finer details might still change. We will strive to maintain backward compatibility, but the function needs wider usage and more extensive testing in order to optimise the unlying code.
Antibiotics

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

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

AMK: amikacin (J01GB06), AMX: amoxicillin (J01CA04), AMC: amoxicillin/clavulanic acid (J01CR02), AMP: ampicillin (J01CA01), SAM: ampicillin/sulbactam (J01CR01), AZM: azithromycin (J01FA10), AZL: azlocillin (J01CA09), ATM: aztreonam (J01DF01), CAP: capreomycin (J04AB03), RID: cefaloridine (J01DB02), CZO: cefazolin (J01DB04), FEP: cefepime (J01DE01), CTX: cefotaxime (J01DD01), CTT: cefotetan (J01DC05), FOX: cefoxitin (J01DC01), CPT: ceftaroline (J01DI02), CAZ: ceftazidime (J01DD02), CRO: ceftriaxone (J01DD04), CXM: cefuroxime (J01DC02), CED: cephradine (J01DB09), CHL: chloramphenicol (J01BA01), CIP: ciprofloxacin (J01MA02), CLR: clarithromycin (J01FA09), CLI: clindamycin (J01FF01), COL: colistin (J01XB01), DAP: daptomycin (J01XX09), DOR: doripenem (J01DH04), DOX: doxycycline (J01AA02), ETP: ertapenem (J01DH03), ERY: erythromycin (J01FA01), ETH: ethambutol (J04AK02), FLC: flucloxacillin (J01CF05), FOS: fosfomycin (J01XX01), FUS: fusidic acid (J01XC01), GAT: gatifloxacin (J01MA16), GEN: gentamicin (J01GB03), GEH: gentamicin-high (no ATC code), IPM: imipenem (J01DH51), INH: isoniazid (J04AC01), KAN: kanamycin (J01GB04), LVX: levofloxacin (J01MA12), LIN: lincomycin (J01FF02), LNZ: linezolid (J01XX08), MEM: meropenem (J01DH02), MTR: metronidazole (J01XD01), MEZ: mezlocillin (J01CA10), MNO: minocycline (J01AA08), MFX: moxifloxacin (J01MA14), NAL: nalidixic acid (J01MB02), NEO: neomycin (J01GB05), NET: netilmicin (J01GB07), NIT: nitrofurantoin (J01XE01), NOR: norfloxacin (J01MA06), NOV: novobiocin (J01XX95), OFX: ofloxacin (J01MA01), OXA: oxacillin (J01CF04), PEN: penicillin G (J01CE01), PIP: piperacillin (J01CA12), TZP: piperacillin/tazobactam (J01CR05), PLB: polymyxin B (J01XB02), PRI: pristinamycin (J01FG01), PZA: pyrazinamide (J04AK01), QDA: quinupristin/dalfopristin (J01FG02), RIB: rifabutin (J04AB04), RIF: rifampicin (J04AB02), RFP: rifapentine (J04AB05), RXT: roxithromycin (J01FA06), SIS: sisomicin (J01GB08), STH: streptomycin-high (no ATC code), TEC: teicoplanin (J01XA02), TLV: telavancin (J01XA03), TCY: tetracycline (J01AA07), TIC: ticarcillin (J01CA13), TCC: ticarcillin/clavulanic acid (J01CR03), TGC: tigecycline (J01AA12), TOB: tobramycin (J01GB01), TMP: trimethoprim (J01EA01), SXT: trimethoprim/sulfamethoxazole (J01EE01), VAN: vancomycin (J01XA01).

Interpretation of R and S/I

In 2019, the European Committee on Antimicrobial Susceptibility Testing (EUCAST) has decided to change the definitions of susceptibility testing categories R and S/I as shown below (http://www.eucast.org/news/änder/)..

- **R** = Resistant
  A microorganism is categorised as Resistant when there is a high likelihood of therapeutic failure even when there is increased exposure. Exposure is a function of how the mode of administration, dose, dosing interval, infusion time, as well as distribution and excretion of the antimicrobial agent will influence the infecting organism at the site of infection.

- **S** = Susceptible
  A microorganism is categorised as Susceptible, standard dosing regimen, when there is a high likelihood of therapeutic success using a standard dosing regimen of the agent.
• **I = Increased exposure, but still susceptible**
  
  A microorganism is categorised as *Susceptible, Increased exposure* when there is a high likelihood of therapeutic success because exposure to the agent is increased by adjusting the dosing regimen or by its concentration at the site of infection.

  This AMR package honours this new insight. Use `susceptibility()` (equal to `proportion_SI()`) to determine antimicrobial susceptibility and `count_SUS()` (equal to `count_SI()`) to count susceptible isolates.

**Read more on our website!**

On our website [https://msberends.gitlab.io/AMR](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**

Please see *Details* for the list of publications used for this function.

**Examples**

```r
library(dplyr)

example_isolates %>%
  mdro() %>%
  freq()

example_isolates %>%
  mutate(EUCAST = eucast_exceptional_phenotypes(.),
         BRMO = brmo(.),
         MRGN = mrgn(.))
```

---

### microorganisms

**Data set with 69,447 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 69,447 observations and 17 variables:

- `mo`  
  ID of microorganism as used by this package
- `col_id`  
  Catalogue of Life ID
- `fullname`  
  Full name, like "Escherichia coli"
- `kingdom, phylum, class, order, family, genus, species, subspecies`  
  Taxonomic rank of the microorganism
- `rank`  
  Text of the taxonomic rank of the microorganism, like "species" or "genus"
- `ref`  
  Author(s) and year of concerning scientific publication
- `species_id`  
  ID of the species as used by the Catalogue of Life
- `source`  
  Either "CoL", "DSMZ" (see Source) or "manually added"
- `prevalence`  
  Prevalence of the microorganism, see `as.mo()`
- `snomed`  
  SNOMED code of the microorganism. Use `mo_snomed()` to retrieve it quickly, see `mo_property()`.

Details

Manually added were:

- 11 entries of *Streptococcus* (beta-haemolytic: groups A, B, C, D, F, G, H, K and unspecified; other: viridans, milleri)
- 2 entries of *Staphylococcus* (coagulase-negative (CoNS) and coagulase-positive (CoPS))
- 3 entries of *Trichomonas* (*Trichomonas vaginalis*, and its family and genus)
- 1 entry of *Blastocystis* (*Blastocystis hominis*), although it officially does not exist (Noel et al. 2005, PMID 15634993)
- 5 other 'undefined' entries (unknown, unknown Gram negatives, unknown Gram positives, unknown yeast and unknown fungus)
- 6 families under the Enterobacterales order, according to Adeolu et al. (2016, PMID 27620848), that are not (yet) in the Catalogue of Life
- 12,600 species from the DSMZ (Deutsche Sammlung von Mikroorganismen und Zellkulturen) since the DSMZ contain the latest taxonomic information based on recent publications

Direct download:

This data set is available as 'flat file' for use even without R - you can find the file here:


The file in R format (which data structure) can be found here:

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 (~70,000 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. Check which version of the Catalogue of Life 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), http://www.catalogueoflife.org (check included annual version with catalogue_of_life_version()).

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

See Also

as.mo(), mo_property(), microorganisms.codes

microorganisms.codes  Translation table for common microorganism codes

Description

A data set containing commonly used codes for microorganisms, from laboratory systems and WHONET. Define your own with set_mo_source(). They will all be searched when using as.mo() and consequently all the mo_* functions.
microorganisms.old

Usage

microorganisms.codes

Format

A data.frame with 5,585 observations and 2 variables:

- code
  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 (~70,000 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. Check which version of the Catalogue of Life 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 24,253 observations and 5 variables:

- `col_id`  
  Catalogue of Life ID that was originally given
- `col_id_new`  
  New Catalogue of Life ID that responds to an entry in the `microorganisms` data set
- `fullname`  
  Old full taxonomic name of the microorganism
- `ref`  
  Author(s) and year of concerning scientific publication
- `prevalence`  
  Prevalence of the microorganism, see `as.mo()`

Catalogue of Life

This package contains the complete taxonomic tree of almost all microorganisms (~70,000 species) from the authoritative and comprehensive Catalogue of Life ([http://www.catalogueoflife.org](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. Check which version of the Catalogue of Life was included in this package with `catalogue_of_life_version()`.

Read more on our website!

On our website [https://msberends.gitlab.io/AMR](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


See Also

`as.mo()` `mo_property()` `microorganisms`

---

mo_property  

**Property of a microorganism**

Description

Use these functions to return a specific property of a microorganism. All input values will be evaluated internally with `as.mo()`, which makes it possible to use microbial abbreviations, codes and names as input. Please see *Examples*. 
Usage

mo_name(x, language = get_locale(), ...)
mo_fullname(x, language = get_locale(), ...)
mo_shortname(x, language = get_locale(), ...)
mo_subspecies(x, language = get_locale(), ...)
mo_species(x, language = get_locale(), ...)
mo_genus(x, language = get_locale(), ...)
mo_family(x, language = get_locale(), ...)
mo_order(x, language = get_locale(), ...)
mo_class(x, language = get_locale(), ...)
mo_phylum(x, language = get_locale(), ...)
mo_kingdom(x, language = get_locale(), ...)
mo_type(x, language = get_locale(), ...)
mo_gramstain(x, language = get_locale(), ...)
mo_snomed(x, ...)
mo_ref(x, ...)
mo_authors(x, ...)
mo_year(x, ...)
mo_rank(x, ...)
mo_taxonomy(x, language = get_locale(), ...)
mo_synonyms(x, ...)
mo_info(x, language = get_locale(), ...)
mo_url(x, open = FALSE, ...)
mo_property(x, property = "fullname", language = get_locale(), ...)
Arguments

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

Details

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

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

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

All output will be translated where possible.

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

Value

- An integer in case of `mo_year()`  
- A list in case of `mo_taxonomy()` and `mo_info()`  
- A named character in case of `mo_url()`  
- A double in case of `mo_snomed()`  
- A character in all other cases

Catalogue of Life

This package contains the complete taxonomic tree of almost all microorganisms (~70,000 species) from the authoritative and comprehensive Catalogue of Life ([http://www.catalogueoflife.org](http://www.catalogueoflife.org)). The Catalogue of Life is the most comprehensive and authoritative global index of species currently available.

[Click here](http://www.catalogueoflife.org) for more information about the included taxa. Check which version of the Catalogue of Life was included in this package with `catalogue_of_life_version()`.
Source


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

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

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

# other properties ---------------------------------------------
mo_gramstain("E. coli") # "Gram-negative"
mo_snomed("E. coli") # 112283007, 116395006, ... (SNOMED codes)
mo_type("E. coli") # "Bacteria" (equal to kingdom, but may be translated)
mo_rank("E. coli") # "species"
mo_url("E. coli") # get the direct url to the online database entry
mo_synonyms("E. coli") # get previously accepted taxonomic names

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

# abbreviations known in the field ---------------------------------------------
mo_genus("MRSA") # "Staphylococcus"
mo_species("MRSA") # "aureus"
mo_shortname("VISA") # "S. aureus"
mo_gramstain("VISA") # "Gram-positive"
mo_genus("EHEC") # "Escherichia"
mo_species("EHEC") # "coli"

# known subspecies -----------------------------------------------------------
mo_name("doylei") # "Campylobacter jejuni doylei"
mo_genus("doylei") # "Campylobacter"
mo_species("doylei") # "jejuni"
mo_subspecies("doylei") # "doylei"
mo_fullname("K. pneu rh") # "Klebsiella pneumoniae rhinoscleromatis"
mo_shortname("K. pneu rh") # "K. pneumoniae"

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

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

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

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

mo_fullname("S. pyogenes",
  Lancefield = TRUE,
  language = "de") # "Streptococcus Gruppe A"
mo_fullname("S. pyogenes",
  Lancefield = TRUE,
  language = "nl") # "Streptococcus groep A"
# get a list with the complete taxonomy (from kingdom to subspecies)
motaxonomy("E. coli")
# get a list with the taxonomy, the authors, Gram-stain and URL to the online database
mo_info("E. coli")

---

**Use predefined reference data set**

**Description**

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

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

**Usage**

```r
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:

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Organisation XYZ</td>
<td>mo</td>
</tr>
<tr>
<td>2</td>
<td>lab_mo_ecoli</td>
<td>B_ESCHR_COLI</td>
</tr>
<tr>
<td>3</td>
<td>lab_mo_kpneumoniae</td>
<td>B_KLBSL_PNMM</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

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_COLI
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_COLI B_ESCHR_COLI B_ESCHR_COLI
```

If we edit the Excel file to, let's say, by adding row 4 like this:

```
<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Organisation XYZ</td>
<td>mo</td>
</tr>
<tr>
<td>2</td>
<td>lab_mo_ecoli</td>
<td>B_ESCHR_COLI</td>
</tr>
<tr>
<td>3</td>
<td>lab_mo_kpneumoniae</td>
<td>B_KLBSL_PNMM</td>
</tr>
<tr>
<td>4</td>
<td>lab_Staph_aureus</td>
<td>B_STPHY_AURS</td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

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

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

To remove the reference data file completely, just use "" or NULL as input for [set_mo_source()]:
```
set_mo_source(NULL)
# Removed mo_source file '~/mo_source.rds'.
```
Stable lifecycle

The lifecycle of this function is stable. In a stable function, we are largely happy with the underlying code, and major changes are unlikely. This means that the underlying code will generally evolve by adding new arguments; we will avoid removing arguments or changing the meaning of existing arguments.

If the underlying code needs breaking changes, they will occur gradually. To begin with, the function or argument will be deprecated; it will continue to work but will emit a message informing you of the change. Next, typically after at least one newly released version on CRAN, the message will be transformed to an error.

Read more on our website!

On our website [https://msberends.gitlab.io/AMR](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.

---

### pca

**Principal Component Analysis (for AMR)**

#### Description

Performs a principal component analysis (PCA) based on a data set with automatic determination for afterwards plotting the groups and labels, and automatic filtering on only suitable (i.e. non-empty and numeric) variables.

#### Usage

```r
pca(
  x,
  ..., 
  retx = TRUE,
  center = TRUE,
  scale. = TRUE,
  tol = NULL,
  rank. = NULL
)
```

#### Arguments

- `x` a data.frame containing numeric columns
- `...` columns of `x` to be selected for PCA
- `retx` a logical value indicating whether the rotated variables should be returned.
- `center` a logical value indicating whether the variables should be shifted to be zero centered. Alternately, a vector of length equal the number of columns of `x` can be supplied. The value is passed to `scale`. 
scale.

scale. a logical value indicating whether the variables should be scaled to have unit variance before the analysis takes place. The default is FALSE for consistency with S, but in general scaling is advisable. Alternatively, a vector of length equal the number of columns of x can be supplied. The value is passed to scale.

tol

tol a value indicating the magnitude below which components should be omitted. (Components are omitted if their standard deviations are less than or equal to tol times the standard deviation of the first component.) With the default null setting, no components are omitted (unless rank is specified less than min(dim(x))). Other settings for tol could be tol = 0 or tol = sqrt(.Machine$double.eps), which would omit essentially constant components.

rank.

rank. optionally, a number specifying the maximal rank, i.e., maximal number of principal components to be used. Can be set as alternative or in addition to tol, useful notably when the desired rank is considerably smaller than the dimensions of the matrix.

Details

The pca() function takes a data.frame as input and performs the actual PCA with the R function prcomp().

The result of the pca() function is a prcomp object, with an additional attribute non_numeric_cols which is a vector with the column names of all columns that do not contain numeric values. These are probably the groups and labels, and will be used by ggplot_pca().

Value

An object of classes pca and prcomp

Maturing lifecycle

The lifecycle of this function is maturing. The underlying code of a maturing function has been roughed out, but finer details might still change. We will strive to maintain backward compatibility, but the function needs wider usage and more extensive testing in order to optimise the underlying code.

Examples

# `example_isolates` is a dataset available in the AMR package.
# See ?example_isolates.

# calculate the resistance per group first
library(dplyr)
resistance_data <- example_isolates %>%
group_by(order = mo_order(mo), # group on anything, like order
genus = mo_genus(mo)) %>% # and genus as we do here
summarise_if(is.rsi, resistance) # then get resistance of all drugs

# now conduct PCA for certain antimicrobial agents
pca_result <- resistance_data %>%
pca(AMC, CXM, CTX, CAZ, GEN, TOB, TMP, SXT)

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

`resistance()` should be used to calculate resistance, `susceptibility()` should be used to calculate susceptibility.

**Usage**

resistance(..., minimum = 30, as_percent = FALSE, only_all_tested = FALSE)

susceptibility(..., minimum = 30, as_percent = FALSE, only_all_tested = FALSE)

proportion_R(..., minimum = 30, as_percent = FALSE, only_all_tested = FALSE)

proportion_IR(..., minimum = 30, as_percent = FALSE, only_all_tested = FALSE)

proportion_I(..., minimum = 30, as_percent = FALSE, only_all_tested = FALSE)

proportion_SI(..., minimum = 30, as_percent = FALSE, only_all_tested = FALSE)

proportion_S(..., minimum = 30, as_percent = FALSE, only_all_tested = FALSE)

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

rsi_df(
    data,
    translate_ab = "name",
    language = get_locale(),
    minimum = 30,
proportion

as_percent = FALSE,
combine_SI = TRUE,
combine_IR = FALSE
)

Arguments

... one or more vectors (or columns) with antibiotic interpretations. They will be transformed internally with as.rsi() if needed. Use multiple columns to calculate (the lack of) co-resistance: the probability where one of two drugs have a resistant or susceptible result. See Examples.

minimum the minimum allowed number of available (tested) isolates. Any isolate count lower than minimum will return NA with a warning. The default number of 30 isolates is advised by the Clinical and Laboratory Standards Institute (CLSI) as best practice, see Source.

as_percent a logical to indicate whether the output must be returned as a hundred fold with % sign (a character). A value of 0.123456 will then be returned as “12.3%”.

only_all_tested (for combination therapies, i.e. using more than one variable for ...): a logical to indicate that isolates must be tested for all antibiotics, see section Combination therapy below

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

translate_ab a column name of the antibiotics data set to translate the antibiotic abbreviations to, using ab.property()

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

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

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

Details

The function resistance() is equal to the function proportion_R(). The function susceptibility() is equal to the function proportion_SI().

Remember that you should filter your table to let it contain only first isolates! This is needed to exclude duplicates and to reduce selection bias. Use first_isolate() to determine them in your data set.

These functions are not meant to count isolates, but to calculate the proportion of resistance/susceptibility. Use the count() function to count isolates. The function susceptibility() is essentially equal to count_susceptible() / count_all(). Low counts can influence the outcome.
- the proportion functions may camouflage this, since they only return the proportion (albeit being dependent on the minimum parameter).

The function `proportion_df()` takes any variable from data that has an `rsi` class (created with `as.rsi()`) and calculates the proportions R, I and S. The function `rsi_df()` works exactly like `proportion_df()`, but adds the number of isolates.

**Value**

A double or, when `as_percent = TRUE`, a character.

**Combination therapy**

When using more than one variable for ... (= combination therapy)), use `only_all_tested` to only count isolates that are tested for all antibiotics/variables that you test them for. See this example for two antibiotics, Drug A and Drug B, about how `susceptibility()` works to calculate the %SI:

<table>
<thead>
<tr>
<th>Drug A</th>
<th>Drug B</th>
<th>only_all_tested = FALSE</th>
<th>only_all_tested = TRUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>S or I</td>
<td>S or I</td>
<td>X X X X X</td>
<td>X X X X X</td>
</tr>
<tr>
<td>R</td>
<td>S or I</td>
<td>X X X X X</td>
<td>X X X X X</td>
</tr>
<tr>
<td>&lt;NA&gt;</td>
<td>S or I</td>
<td>X X X X X</td>
<td>X X X X X</td>
</tr>
<tr>
<td>S or I</td>
<td>R</td>
<td>X X X X X</td>
<td>X X X X X</td>
</tr>
<tr>
<td>R</td>
<td>R</td>
<td>X X X X X</td>
<td>X X X X X</td>
</tr>
<tr>
<td>&lt;NA&gt;</td>
<td>R</td>
<td>X X X X X</td>
<td>X X X X X</td>
</tr>
<tr>
<td>S or I</td>
<td>&lt;NA&gt;</td>
<td>X X X X X</td>
<td>X X X X X</td>
</tr>
<tr>
<td>R</td>
<td>&lt;NA&gt;</td>
<td>X X X X X</td>
<td>X X X X X</td>
</tr>
<tr>
<td>&lt;NA&gt;</td>
<td>&lt;NA&gt;</td>
<td>X X X X X</td>
<td>X X X X X</td>
</tr>
</tbody>
</table>

Please note that, in combination therapies, for `only_all_tested = TRUE` applies that:

\[
\text{count_S() + count_I() + count_R() = count_all()}
\]
\[
\text{proportion_S() + proportion_I() + proportion_R() = 1}
\]

and that, in combination therapies, for `only_all_tested = FALSE` applies that:

\[
\text{count_S() + count_I() + count_R() >= count_all()}
\]
\[
\text{proportion_S() + proportion_I() + proportion_R() >= 1}
\]

Using `only_all_tested` has no impact when only using one antibiotic as input.

**Stable lifecycle**

The lifecycle of this function is **stable**. In a stable function, we are largely happy with the underlying code, and major changes are unlikely. This means that the underlying code will generally evolve by
adding new arguments; we will avoid removing arguments or changing the meaning of existing arguments.

If the underlying code needs breaking changes, they will occur gradually. To begin with, the function or argument will be deprecated; it will continue to work but will emit a message informing you of the change. Next, typically after at least one newly released version on CRAN, the message will be transformed to an error.

**Interpretation of R and S/I**

In 2019, the European Committee on Antimicrobial Susceptibility Testing (EUCAST) has decided to change the definitions of susceptibility testing categories R and S/I as shown below (http://www.eucast.org/newsandr/).

- **R = Resistant**
  
  A microorganism is categorised as *Resistant* when there is a high likelihood of therapeutic failure even when there is increased exposure. Exposure is a function of how the mode of administration, dose, dosing interval, infusion time, as well as distribution and excretion of the antimicrobial agent will influence the infecting organism at the site of infection.

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

- **I = Increased exposure, but still susceptible**
  
  A microorganism is categorised as *Susceptible, Increased exposure* when there is a high likelihood of therapeutic success because exposure to the agent is increased by adjusting the dosing regimen or by its concentration at the site of infection.

This AMR package honours this new insight. Use `susceptibility()` (equal to `proportion_SI()`) to determine antimicrobial susceptibility and `count_susceptible()` (equal to `count_SI()`) to count susceptible isolates.

**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**


**See Also**

AMR::count() to count resistant and susceptible isolates.
Examples

# example_isolates is a data set available in the AMR package.
?example_isolates

resistance(example_isolates$AMX) # determines %R
susceptibility(example_isolates$AMX) # determines %S+I

# be more specific
proportion_S(example_isolates$AMX)
proportion_SI(example_isolates$AMX)
proportion_I(example_isolates$AMX)
proportion_IR(example_isolates$AMX)
proportion_R(example_isolates$AMX)

library(dplyr)
example_isolates %>%
group_by(hospital_id) %>%
summarise(r = resistance(CIP),
          n = n_rsi(CIP)) # n_rsi works like n_distinct in dplyr, see ?n_rsi

example_isolates %>%
group_by(hospital_id) %>%
summarise(R = resistance(CIP, as_percent = TRUE),
          SI = susceptibility(CIP, as_percent = TRUE),
          n1 = count_all(CIP), # the actual total; sum of all three
          n2 = n_rsi(CIP), # same - analogous to n_distinct
          total = n()) # NOT the number of tested isolates!

# Calculate co-resistance between amoxicillin/clav acid and gentamicin,
# so we can see that combination therapy does a lot more than mono therapy:
example_isolates %>% susceptibility(AMC) # %SI = 76.3%
example_isolates %>% count_all(AMC) # n = 1879

example_isolates %>% susceptibility(GEN) # %SI = 75.4%
example_isolates %>% count_all(GEN) # n = 1855

example_isolates %>% susceptibility(AMC, GEN) # %SI = 94.1%
example_isolates %>% count_all(AMC, GEN) # n = 1939

# See Details on how `only_all_tested` works. Example:
example_isolates %>%
summarise(numerator = count_susceptible(AMC, GEN),
          denominator = count_all(AMC, GEN),
          proportion = susceptibility(AMC, GEN))

example_isolates %>%
summarise(numerator = count_susceptible(AMC, GEN, only_all_tested = TRUE),
          denominator = count_all(AMC, GEN, only_all_tested = TRUE),
          proportion = susceptibility(AMC, GEN, only_all_tested = TRUE))

example_isolates %>%
group_by(hospital_id) %>%
  summarise(cipro_p = susceptibility(CIP, as_percent = TRUE),
             cipro_n = count_all(CIP),
             genta_p = susceptibility(GEN, as_percent = TRUE),
             genta_n = count_all(GEN),
             combination_p = susceptibility(CIP, GEN, as_percent = TRUE),
             combination_n = count_all(CIP, GEN))

# Get proportions S/I/R immediately of all rsi columns
example_isolates %>%
  select(AMX, CIP) %>%
  proportion_df(translate = FALSE)

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

## Not run:
# calculate current empiric combination therapy of Helicobacter gastritis:
my_table %>%
  filter(first_isolate == TRUE,
         genus == "Helicobacter") %>%
  summarise(p = susceptibility(AMX, MTR), # amoxicillin with metronidazole
             n = count_all(AMX, MTR))

## End(Not run)

---

**p_symbol**

**Symbol of a p-value**

**Description**

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

**Usage**

```r
p_symbol(p, emptychar = " ")
```

**Arguments**

- `p`: p value
- `emptychar`: text to show when p > 0.1
Questioning lifecycle

The lifecycle of this function is questioning. We are no longer convinced that this function is the optimal approach (but we do not know yet what a better approach would be), or whether this function should be in our AMR package at all.

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.

---

**read.4D**

*Read data from 4D database*

**Description**

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

**Usage**

```r
read.4D(
  file,
  info = interactive(),
  header = TRUE,
  row.names = NULL,
  sep = "\t",
  quote = "\"",
  dec = ",",
  na.strings = c("NA", "", "."),
  skip = 2,
  check.names = TRUE,
  strip.white = TRUE,
  fill = TRUE,
  blank.lines.skip = TRUE,
  stringsAsFactors = FALSE,
  fileEncoding = "UTF-8",
  encoding = "UTF-8"
)
```

Arguments

file
the name of the file which the data are to be read from. Each row of the table appears as one line of the file. If it does not contain an absolute path, the file name is relative to the current working directory, getwd(). Tilde-expansion is performed where supported. This can be a compressed file (see file).
Alternatively, file can be a readable text-mode connection (which will be opened for reading if necessary, and if so closed (and hence destroyed) at the end of the function call). (If stdin() is used, the prompts for lines may be somewhat confusing. Terminate input with a blank line or an EOF signal, Ctrl-D on Unix and Ctrl-Z on Windows. Any pushback on stdin() will be cleared before return.) file can also be a complete URL. (For the supported URL schemes, see the 'URLs' section of the help for url.)

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

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

row.names
a vector of row names. This can be a vector giving the actual row names, or a single number giving the column of the table which contains the row names, or character string giving the name of the table column containing the row names. If there is a header and the first row contains one fewer field than the number of columns, the first column in the input is used for the row names. Otherwise if row.names is missing, the rows are numbered.
Using row.names = NULL forces row numbering. Missing or NULL row.names generate row names that are considered to be 'automatic' (and not preserved by as.matrix).

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

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

dec
the character used in the file for decimal points.

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

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

check.names
logical. If TRUE then the names of the variables in the data frame are checked to ensure that they are syntactically valid variable names. If necessary they are adjusted (by make.names) so that they are, and also to ensure that there are no duplicates.
**strip.white** logical. Used only when `sep` has been specified, and allows the stripping of leading and trailing white space from unquoted character fields (numeric fields are always stripped). See `scan` for further details (including the exact meaning of ‘white space’), remembering that the columns may include the row names.

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

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

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

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

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

**Details**

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

**Dormant lifecycle**

The lifecycle of this function is **dormant**. A dormant function is currently not under active development and has not reached a stable phase. We might return to it in the future. As with experimental functions, you are best off waiting until a function is more mature before you use it in production code.

**Read more on our website!**

On our website [https://msberends.gitlab.io/AMR](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.
resistance_predict

Usage

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

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

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

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

Arguments

- **x** a data.frame containing isolates.
- **col_ab** column name of x containing antimicrobial interpretations ("R", "I" and "S")
- **col_date** column name of the date, will be used to calculate years if this column doesn’t consist of years already, defaults to the first column of with a date class
year_min  lowest year to use in the prediction model, defaults to the lowest year in `col_date`
year_max  highest year to use in the prediction model, defaults to 10 years after today
year_every  unit of sequence between lowest year found in the data and `year_max`
minimum  minimal amount of available isolates per year to include. Years containing less observations will be estimated by the model.
model  the statistical model of choice. This could be a generalised linear regression model with binomial distribution (i.e. using ‘`glm(..., family = binomial)`’, 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 all valid options.
I_as_S  a logical to indicate whether values I should be treated as S (will otherwise be treated as R). The default, TRUE, follows the redefinition by EUCAST about the interpretation of I (increased exposure) in 2019, see section Interpretation of S, I and R below.
preserve_measurements  a logical to indicate whether predictions of years that are actually available in the data should be overwritten by the original data. The standard errors of those years will be NA.
info  a logical to indicate whether textual analysis should be printed with the name and `summary()` of the statistical model.
...  parameters passed on to functions
main  title of the plot
ribbon  a logical to indicate whether a ribbon should be shown (default) or error bars

Details

Valid options for the statistical model (parameter `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

A `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, please see Examples.

**Maturing lifecycle**

The lifecycle of this function is **maturing**. The underlying code of a maturing function has been roughed out, but finer details might still change. We will strive to maintain backward compatibility, but the function needs wider usage and more extensive testing in order to optimise the underlying code.

**Interpretation of R and S/I**

In 2019, the European Committee on Antimicrobial Susceptibility Testing (EUCAST) has decided to change the definitions of susceptibility testing categories R and S/I as shown below (http://www.eucast.org/newsIandR/).

- **R = Resistant**
  A microorganism is categorised as *Resistant* when there is a high likelihood of therapeutic failure even when there is increased exposure. Exposure is a function of how the mode of administration, dose, dosing interval, infusion time, as well as distribution and excretion of the antimicrobial agent will influence the infecting organism at the site of infection.

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

- **I = Increased exposure, but still susceptible**
  A microorganism is categorised as *Susceptible, Increased exposure* when there is a high likelihood of therapeutic success because exposure to the agent is increased by adjusting the dosing regimen or by its concentration at the site of infection.

This AMR package honours this new insight. Use `susceptibility()` (equal to `proportion_SI()`) to determine antimicrobial susceptibility and `count_susceptible()` (equal to `count_SI()`) to count susceptible isolates.

**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 `proportion()` functions to calculate resistance

Models: `lm()` `glm()`
Examples

```r
x <- resistance_predict(example_isolates,
    col_ab = "AMX",
    year_min = 2010,
    model = "binomial")

plot(x)
ggplot_rsi_predict(x)

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

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

# create nice plots with ggplot2 yourself
if (!require(ggplot2)) {
  data <- example_isolates %>%
      filter(mo == as.mo("E. coli")) %>%
      resistance_predict(col_ab = "AMX",
          col_date = "date",
          model = "binomial",
          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 = "% R",
        x = "Year") +
  theme_minimal(base_size = 13)
}
```
rsi_translation  Data set for R/SI interpretation

Description

Data set to interpret MIC and disk diffusion to R/SI values. Included guidelines are CLSI (2011-2019) and EUCAST (2011-2020). Use `as.rsi()` to transform MICs or disks measurements to R/SI values.

Usage

`rsi_translation`

Format

A `data.frame` with 18,964 observations and 10 variables:

- `guideline` Name of the guideline
- `method` Either "MIC" or "DISK"
- `site` Body site, e.g. "Oral" or "Respiratory"
- `mo` Microbial ID, see `as.mo()`
- `ab` Antibiotic ID, see `as.ab()`
- `ref_tbl` Info about where the guideline rule can be found
- `disk_dose` Dose of the used disk diffusion method
- `breakpoint_S` Lowest MIC value or highest number of millimetres that leads to "S"
- `breakpoint_R` Highest MIC value or lowest number of millimetres that leads to "R"
- `uti` A logical value (TRUE/FALSE) to indicate whether the rule applies to a urinary tract infection (UTI)

Details

The repository of this AMR package contains a file comprising this exact data set: https://gitlab.com/msberends/AMR/blob/master/data-raw/rsi_translation.txt. This file allows for machine reading EUCAST and CLSI guidelines, which is almost impossible with the Excel and PDF files distributed by EUCAST and CLSI. This file is updated automatically.
**Skewness**

**Skewness of the sample**

**Description**

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

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

**Usage**

```r
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.

**Questioning lifecycle**

The lifecycle of this function is **questioning**. We are no longer convinced that this function is the optimal approach (but we do not know yet what a better approach would be), or whether this function should be in our AMR package at all.

**Read more on our website!**

On our website [https://msberends.gitlab.io/AMR](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()

translate

Translate strings from AMR package

Description

For language-dependent output of AMR functions, like mo_name(), mo_type() and ab_name().

Usage

get_locale()

details

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

Currently supported languages can be found if running: unique(AMR:::translations_file$lang).

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

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

The system language will be used at default, if that language is supported. The system language can be overwritten with Sys.setenv(AMR_locale = yourlanguage).

Stable lifecycle

The lifecycle of this function is stable. In a stable function, we are largely happy with the underlying code, and major changes are unlikely. This means that the underlying code will generally evolve by adding new arguments; we will avoid removing arguments or changing the meaning of existing arguments.

If the underlying code needs breaking changes, they will occur gradually. To begin with, the function or argument will be deprecated; it will continue to work but will emit a message informing you of the change. Next, typically after at least one newly released version on CRAN, the message will be transformed to an error.

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

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

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

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

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

# Spanish
mo_name("CoNS", language = "es")
#> "Staphylococcus coagulasa negativo (SCN)"

# Italian
mo_name("CoNS", language = "it")
#> "Staphylococcus negativo coagulasì (CoNS)"

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

---

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


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.

NOTE: The WHOCC copyright does not allow use for commercial purposes, unlike any other info from this package. See https://www.whocc.no/copyright_disclaimer/.

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

```r
as.ab("meropenem")
ab_name("J01DH02")
ab_tradenames("flucloxacillin")
```

---

**WHONET**

*Data set with 500 isolates - WHONET example*

---

**Description**

This example data set has the exact same structure as an export file from WHONET. Such files can be used with this package, as this example data set shows. The data itself was based on our example_isolates 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
  28 different antibiotics. You can lookup the abbreviations in the antibiotics data set, or use e.g. ab_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, 17
*Topic Lancefield
   as.mo, 17
*Topic becker
   as.mo, 17
*Topic datasets
   antibiotics, 10
   example_isolates, 43
   example_isolates_unclean, 44
   microorganisms, 77
   microorganisms.codes, 79
   microorganisms.old, 80
   rsi_translation, 103
   WHONET, 107
*Topic guess
   as.mo, 17
*Topic lancefield
   as.mo, 17
*Topic mo
   as.mo, 17
%like% (like), 71
%like_case% (like), 71
3MRGN (mdro), 73
4MRGN (mdro), 73

ab, 13
ab (as.ab), 13
ab_atc (ab_property), 3
ab_atc_group1 (ab_property), 3
ab_atc_group2 (ab_property), 3
ab_cid (ab_property), 3
ab_cid(), 4
ab_ddd (ab_property), 3
ab_ddd(), 4
ab_group (ab_property), 3
ab_info (ab_property), 3
ab_info(), 4
ab_loinc (ab_property), 3
ab_loinc(), 11
ab_name (ab_property), 3
ab_name(), 44, 60, 105
ab_name(AMP), 108
ab_property, 3
ab_property(), 10, 11, 13, 36, 59, 91
ab_synonyms (ab_property), 3
ab_synonyms(), 4
ab_tradenames (ab_property), 3
ab_tradenames(), 4
age, 5
age(), 7, 8
age_groups, 7
age_groups(), 6, 108
AMR, 9
AMR::count(), 93
anti_join_microorganisms (join), 64
antibiotics, 3–5, 10, 10, 13, 14, 31, 36, 44,
   46, 59, 62, 63, 91, 108
antibiotics$group, 45
antivirals, 11
antivirals (antibiotics), 10
as.ab, 13
as.ab(), 3, 4, 10, 24, 63, 103
as.character(), 72
as.disk, 15
as.matrix, 97
as.mic, 16
as.mic(), 25
as.mo, 17
as.mo(), 19, 20, 23, 24, 30, 33, 40, 44, 48, 64,
   66, 74, 77–81, 83, 86, 96, 103, 107
as.POSIXlt(), 6
as.rsi, 23
as.rsi(), 15–17, 24, 36, 40, 44, 59, 60, 91,
   92, 96, 103, 108
ATC (ab_property), 3
atc_online_ddd (atc_online_property), 27
atc_online_groups
   (atc_online_property), 27
INDEX

atc_online_property, 27
availability, 29
base::grep(), 73
biplot(), 55
BRMO (mdro), 73
brmo (mdro), 73
bug_drug_combinations, 30
bug_drug_combinations(), 31
catalogue_of_life, 32
catalogue_of_life_version, 34
catalogue_of_life_version(), 21, 32, 35, 79–81, 83, 84
class, 4, 20, 64, 72, 83, 92
chisq.test(), 51–54
Click here, 21, 32, 35, 79–81, 83
close, 97
count, 35
count_all (count), 35
count_all(), 36
count_df (count), 35
count_df(), 36, 60
count_I (count), 35
count_IR (count), 35
count_R (count), 35
count_R(), 36
count_resistant (count), 35
count_resistant(), 35, 36
count_S (count), 35
count_SI (count), 35
count_SI(), 25, 36, 37, 77, 93, 101
count_susceptible (count), 35
count_susceptible(), 25, 35–37, 77, 93, 101
data.frame, 10, 11, 18–21, 24, 29, 31, 36, 41, 43, 44, 48, 59, 63, 70, 78, 80, 81, 86, 88, 89, 91, 99, 100, 103, 104, 107
disk, 15, 23
disk (as.disk), 15
double, 4, 6, 83, 92
dplyr::join(), 64

Encoding, 98
EUCAST (eucast_rules), 39
eucast_exceptional_phenotypes (mdro), 73
eucast_rules, 39
eucast_rules(), 24, 41, 76
example_isolates, 43, 107
example_isolates_unclean, 44
facets_rsi (ggplot_rsi), 57
facets_rsi(), 60
factor, 7, 16, 75
file, 97, 98
filter_1st_cephalosporins (filter_ab_class), 45
filter_2nd_cephalosporins (filter_ab_class), 45
filter_3rd_cephalosporins (filter_ab_class), 45
filter_4th_cephalosporins (filter_ab_class), 45
filter_5th_cephalosporins (filter_ab_class), 45
filter_ab_class, 45
filter_aminoglycosides (filter_ab_class), 45
filter_carbapenems (filter_ab_class), 45
filter_cephalosporins (filter_ab_class), 45
filter_first_isolate (first_isolate), 47
filter_first_isolate(), 49
filter_first_weighted_isolate (first_isolate), 47
filter_first_weighted_isolate(), 49
filter_fluoroquinolones (filter_ab_class), 45
filter_glycopeptides (filter_ab_class), 45
filter_macrolides (filter_ab_class), 45
filter_tetracyclines (filter_ab_class), 45
first_isolate, 47
first_isolate(), 48, 65, 67, 68, 91
fisher.test(), 52, 53
format(), 30, 31
format.bug_drug_combinations (bug_drug_combinations), 30
full_join_microorganisms (join), 64
g.test, 51
g.test(), 51
ggplot_rsi (ggplot_rsi), 57
ggplot_rsi(), 60
get_locale (translate), 105
INDEX

get_locale(), 4, 31, 36, 59, 83, 91
get_mo_source (mo_source), 86
get_mo_source (), 19, 86
getwd, 97
ggplot_pca, 57
ggplot2::facet_wrap(), 60
ggplot2::geom_text(), 60
ggplot2::ggplot(), 59
ggplot2::scale_brewer(), 60
ggplot2::scale_continuous(), 60
ggplot2::theme(), 60
glm(), 101
glm(), 101
inner_join (join), 64
inner_join_microorganisms (join), 64
grep(), 71
is.ab (as.ab), 13
is.disk (as.disk), 15
is.mic (as.mic), 16
is.mo (as.mo), 17
is.rsi (as.rsi), 23
is.rsi.eligible (), 24
join, 64
key_antibiotics, 65
key_antibiotics (), 48–50, 67, 68
key_antibiotics_equal
 (key_antibiotics), 65
key_antibiotics_equal(), 67, 68
kurtosis, 69
kurtosis(), 105
labels_rsi_count (ggplot_rsi), 57
labels_rsi_count (), 59, 60
left_join_microorganisms (join), 64
lifecycle, 4, 6, 7, 13, 15, 17, 21, 25, 28, 29,
31, 37, 41, 46, 50, 53, 57, 60, 63, 65,
68, 70, 70, 71, 72, 75, 88, 89, 92, 96,
98, 101, 104, 105
like, 71
list, 4, 29, 34, 83
lm(), 101
logical, 7, 24, 49, 68, 71, 72
make.names, 97
matrix, 70, 104
MDR (mdro), 73
mdr_cmi2012 (mdro), 73
mdr_cmi2012(), 75
mdr_tb (mdro), 73
mdr_tb(), 75
mdro, 73
mdro (), 41, 74–76
mic, 16, 23
mic (as.mic), 16
microorganisms, 12, 19–21, 33–35, 44, 64,
77, 80, 81, 83, 84
microorganisms.codes, 79, 79
microorganisms.old, 20, 80
microorganisms$mo, 86
mo, 18–20, 24, 30, 40, 48, 64, 66, 74
mo (as.mo), 17
mo_* , 19, 79
mo_authors (mo_property), 81
mo_authors(), 83
mo_class (mo_property), 81
mo_failures (as.mo), 17
mo_failures(), 20
mo_family (mo_property), 81
mo_fullname (mo_property), 81
mo_fullname (), 105
mo_genus (mo_property), 81
mo_genus (), 21, 86
mo_gramstain (mo_property), 81
mo_gramstain(), 21, 83, 86
mo_info (mo_property), 81
mo_info(), 83
mo_kingdom (mo_property), 81
mo_name (mo_property), 81
mo_name (), 105
mo_order (mo_property), 81
mo_phylum (mo_property), 81
mo_property, 81
mo_property (), 21, 78, 79, 81, 105
mo_rank (mo_property), 81
mo_ref (mo_property), 81
mo_ref (), 83
mo_renamed (as.mo), 17
mo_renamed(), 20
mo_shortname (mo_property), 81
mo_shortname(), 31
mo_snomed (mo_property), 81
mo_snomed(), 78, 83
mo_source, 86
mo_species (mo_property), 81
mo_species(), 81, 83
mo_species(), 81
mo_subspecies (mo_property), 81
mo_subspecies(), 81
mo_synonyms (mo_property), 81
mo_synonyms(), 81
mo_taxonomy (mo_property), 81
mo_taxonomy(), 81
mo_taxonomy(), 83
mo_type (mo_property), 81
mo_type(), 105, 108
mo_uncertainties (as.mo), 17
mo_uncertainties(), 20
mo_url (mo_property), 81
mo_url(), 83
mo_year (mo_property), 81
mo_year(), 83
mrgn (mdro), 73
mrgn(). 75
n_distinct(), 36
n_rsi (count), 35
n_rsi(), 36
NA, 97

p_symbol, 95
pca, 88, 89
pca(), 55, 56, 89
PDR (mdro), 73
plot.resistance_predict
(resistance_predict), 98
portion (proportion), 90
prcomp, 89
prcomp(), 55, 89
princomp(), 55
proportion(), 101
proportion_*, 38
proportion_df (proportion), 90
proportion_df(), 92
proportion_I (proportion), 90
proportion_IR (proportion), 90
proportion_R (proportion), 91
proportion_R(), 91
proportion_S (proportion), 90
proportion_SI (proportion), 90
proportion_SI(), 25, 37, 77, 91, 93, 101
read.4D, 96
readRDS(), 86
resistance (proportion), 90
resistance(), 29, 36, 90, 91
resistance_predict, 98, 100
right_join_microorganisms (join), 64
rsi, 23, 24, 36, 44, 59, 60, 91, 92
rsi (as.rsi), 23
rsi_df (proportion), 90
rsi_df(), 36, 60, 92
rsi_predict (resistance_predict), 98
rsi_translation, 103
scale, 89
scale_colour_brewer(), 56
scale_colour_viridis_d(), 56
scale_rsi_colours (ggplot_rsi), 57
scale_rsi_colours(), 60
scale_y_percent (ggplot_rsi), 57
scale_y_percent(), 60
scan, 97, 98
semi_join_microorganisms (join), 64
set_mo_source (mo_source), 86
set_mo_source(), 19, 79, 86
skewness, 104
skewness(), 70
stdin, 97
summarise(), 70
summary(), 100
susceptibility (proportion), 90
susceptibility(), 25, 29, 36, 37, 77, 90–93, 101
theme_rsi (ggplot_rsi), 57
theme_rsi(), 60
translate, 4, 31, 83, 105
url, 97
utils::browseURL(), 83
vector, 20
WHOCC, 106
WHONET, 107
XDR (mdro), 73