# Package 'AMR'

October 16, 2026

**Version** 3.1.0

**Date** 2026-10-16

**Title** Antimicrobial Resistance Data Analysis

**Description** Functions to simplify and standardise antimicrobial resistance (AMR) data analysis and to work with microbial and antimicrobial properties by using evidence-based methods, as described in <doi:10.18637/jss.v104.i03>.

**Depends** R (>= 3.0.0)

**Suggests** cleaner, cli, crayon, curl, data.table, dplyr, future, future.apply, ggplot2, knitr, openxlsx, pillar, progress, readxl, recipes, rlang, rmarkdown, rstudioapi, rvest, skimr, testthat, tibble, tidymodels, tidyselect, tinytest, vctrs, xml2, usethis

**VignetteBuilder** knitr,rmarkdown

**URL** https://amr-for-r.org, https://github.com/msberends/AMR

**BugReports** https://github.com/msberends/AMR/issues

**License** GPL-2 | file LICENSE

**Encoding** UTF-8

**LazyData** true

**Config/roxygen2/version** 8.1.0

**NeedsCompilation** no

**Author** Matthijs S. Berends [aut, cre] (ORCID: <https://orcid.org/0000-0001-7620-1800>), Dennis Souverein [aut, ctb] (ORCID: <https://orcid.org/0000-0003-0455-0336>), Erwin E. A. Hassing [aut, ctb], Aislinn Cook [ctb] (ORCID: <https://orcid.org/0000-0002-9189-7815>), Andrew P. Norgan [ctb] (ORCID: <https://orcid.org/0000-0002-2955-2066>), Anita Williams [ctb] (ORCID: <https://orcid.org/0000-0002-5295-8451>), Annick Lenglet [ctb] (ORCID: <https://orcid.org/0000-0003-2013-8405>), Anthony Underwood [ctb] (ORCID: <https://orcid.org/0000-0002-8547-4277>),

Anton Mymrikov [ctb],
Bart C. Meijer [ctb],
Christian F. Luz [ctb] (ORCID: <https://orcid.org/0000-0001-5809-5995>),
Dmytro Mykhailenko [ctb],
Eric H. L. C. M. Hazenberg [ctb],
Gwen Knight [ctb] (ORCID: <https://orcid.org/0000-0002-7263-9896>),
Jane Hawkey [ctb] (ORCID: <https://orcid.org/0000-0001-9661-5293>),
Jason Stull [ctb] (ORCID: <https://orcid.org/0000-0002-9028-8153>),
Javier Sanchez [ctb] (ORCID: <https://orcid.org/0000-0003-2605-8094>),
Jonas Salm [ctb],
Judith M. Fonville [ctb],
Kathryn Holt [ctb] (ORCID: <https://orcid.org/0000-0003-3949-2471>),
Larisse Bolton [ctb] (ORCID: <https://orcid.org/0000-0001-7879-2173>),
Matthew Saab [ctb] (ORCID: <https://orcid.org/0009-0008-6626-7919>),
Natacha Couto [ctb] (ORCID: <https://orcid.org/0000-0002-9152-5464>),
Peter Dutey-Magni [ctb] (ORCID:
<https://orcid.org/0000-0002-8942-9836>),
Rogier P. Schade [ctb] (ORCID: <https://orcid.org/0000-0002-9487-4467>),
Sofia Ny [ctb] (ORCID: <https://orcid.org/0000-0002-2017-1363>),
Alex W. Friedrich [ths] (ORCID:
<https://orcid.org/0000-0003-4881-038X>),
Bhanu N. M. Sinha [ths] (ORCID:
<https://orcid.org/0000-0003-1634-0010>),
Casper J. Albers [ths] (ORCID: <https://orcid.org/0000-0002-9213-6743>),
Corinna Glasner [ths] (ORCID: <https://orcid.org/0000-0003-1241-1328>)

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

**Repository** CRAN

**Date/Publication** 2026-10-16 12:10:02 UTC

# Contents

ab_from_text . . . . . 4
ab_property . . . . . 6
add_custom_antimicrobials . . . . . 10
add_custom_microorganisms . . . . . 13
age . . . . . 15
age_groups . . . . . 16
amr_course . . . . . 18
antibiogram . . . . . 19
antimicrobials . . . . . 31
antimicrobial_selectors . . . . . 35
as.ab . . . . . 49
as.av . . . . . 52
as.disk . . . . . 54
as.mic . . . . . 55
as.mo . . . . . 59
as.sir . . . . . 67

atc_online_property . . . 80
availability . . . 82
av_from_text . . . 83
av_property . . . 84
bug_drug_combinations . . . 87
clinical_breakpoints . . . 90
count . . . 93
custom_interpretive_rules . . . 98
custom_mdro_guideline . . . 106
dosage . . . 113
esbl_isolates . . . 114
example_isolates . . . 115
example_isolates_unclean . . . 116
first_isolate . . . 117
g.test . . . 122
get_episode . . . 125
ggplot_pca . . . 129
ggplot_sir . . . 132
guess_ab_col . . . 138
interpretive_rules . . . 139
intrinsic_resistant . . . 144
italicise_taxonomy . . . 145
join . . . 146
key_antimicrobials . . . 147
kurtosis . . . 150
like . . . 151
mdro . . . 153
mean_amr_distance . . . 158
microorganisms . . . 160
microorganisms.codes . . . 164
microorganisms.groups . . . 166
mo_matching_score . . . 167
mo_property . . . 169
mo_source . . . 181
pca . . . 184
plot . . . 186
proportion . . . 193
random . . . 201
resistance_predict . . . 203
sir_combine . . . 207
skewness . . . 212
top_n_microorganisms . . . 213
translate . . . 214
WHOCC . . . 216
WHONET . . . 217

**Index** **220**

ab_from_text *Retrieve Antimicrobial Drug Names and Doses from Clinical Text*

## Description

Use this function on e.g. clinical texts from health care records. It returns a list with all antimicrobial drugs, doses and forms of administration found in the texts.

## Usage

```
ab_from_text(
  text,
  type = c("drug", "dose", "administration"),
  collapse = NULL,
  translate_ab = FALSE,
  thorough_search = NULL,
  info = interactive(),
  ...
)
```

## Arguments

| | |
|---|---|
| text | Text to analyse. |
| type | Type of property to search for, either "drug", "dose" or "administration", see *Examples*. |
| collapse | A character to pass on to paste(, collapse = ...) to only return one character per element of text, see *Examples*. |
| translate_ab | If type = "drug": a column name of the antimicrobials data set to translate the antibiotic abbreviations to, using ab_property(). The default is FALSE. Using TRUE is equal to using "name". |
| thorough_search | A logical to indicate whether the input must be extensively searched for misspelling and other faulty input values. Setting this to TRUE will take considerably more time than when using FALSE. At default, it will turn TRUE when all input elements contain a maximum of three words. |
| info | A logical to indicate whether a progress bar should be printed - the default is TRUE only in interactive mode. |
| ... | Arguments passed on to as.ab(). |

## Details

This function is also internally used by as.ab(), although it then only searches for the first drug name and will throw a note if more drug names could have been returned. Note: the as.ab() function may use very long regular expression to match brand names of antimicrobial drugs. This may fail on some systems.

**Argument** `type`**:**

At default, the function will search for antimicrobial drug names. All text elements will be searched for official names, ATC codes and brand names. As it uses `as.ab()` internally, it will correct for misspelling.

With `type = "dose"` (or similar, like "dosing", "doses"), all text elements will be searched for numeric values that are higher than 100 and do not resemble years. The output will be numeric. It supports any unit (g, mg, IE, etc.) and multiple values in one clinical text, see *Examples*.

With `type = "administration"` (or abbreviations, like "admin", "adm"), all text elements will be searched for a form of drug administration. It supports the following forms (including common abbreviations): buccal, implant, inhalation, instillation, intravenous, nasal, oral, parenteral, rectal, sublingual, transdermal and vaginal. Abbreviations for oral (such as 'po', 'per os') will become "oral", all values for intravenous (such as 'iv', 'intraven') will become "iv". It supports multiple values in one clinical text, see *Examples*.

**Argument** `collapse`**:**

Without using `collapse`, this function will return a list. This can be convenient to use e.g. inside a `mutate()`):
`df %>% mutate(abx = ab_from_text(clinical_text))`

The returned AB codes can be transformed to official names, groups, etc. with all `ab_*` functions such as `ab_name()` and `ab_group()`, or by using the `translate_ab` argument.

With using `collapse`, this function will return a character:
`df %>% mutate(abx = ab_from_text(clinical_text, collapse = "|"))`

## Value

A list, or a character if `collapse` is not `NULL`

## Examples

```
# mind the bad spelling of amoxicillin in this line,
# straight from a true health care record:
ab_from_text("28/03/2020 regular amoxicilliin 500mg po tid")

ab_from_text("500 mg amoxi po and 400mg cipro iv")
ab_from_text("500 mg amoxi po and 400mg cipro iv", type = "dose")
ab_from_text("500 mg amoxi po and 400mg cipro iv", type = "admin")

ab_from_text("500 mg amoxi po and 400mg cipro iv", collapse = ", ")

# if you want to know which antibiotic groups were administered, do e.g.:
abx <- ab_from_text("500 mg amoxi po and 400mg cipro iv")
ab_group(abx[[1]])

if (require("dplyr")) {
  tibble(clinical_text = c(
    "given 400mg cipro and 500 mg amox",
    "started on doxy iv today"
  )) %>%
    mutate(
      abx_codes = ab_from_text(clinical_text),
```

```
      abx_doses = ab_from_text(clinical_text, type = "doses"),
      abx_admin = ab_from_text(clinical_text, type = "admin"),
      abx_coll = ab_from_text(clinical_text, collapse = "|"),
      abx_coll_names = ab_from_text(clinical_text,
        collapse = "|",
        translate_ab = "name"
      ),
      abx_coll_doses = ab_from_text(clinical_text,
        type = "doses",
        collapse = "|"
      ),
      abx_coll_admin = ab_from_text(clinical_text,
        type = "admin",
        collapse = "|"
      )
    )
}
```

---

ab_property *Get Properties of an Antibiotic*

---

## Description

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

## Usage

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

ab_cid(x, ...)

ab_synonyms(x, ...)

ab_tradenames(x, ...)

ab_group(x, language = get_AMR_locale(), all_groups = FALSE, ...)

ab_atc(x, only_first = FALSE, ...)

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

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

ab_loinc(x, ...)

ab_ddd(x, administration = "oral", ...)
```

```
ab_ddd_units(x, administration = "oral", ...)

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

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

ab_property(x, property = "name", language = get_AMR_locale(), ...)

set_ab_names(
  data,
  ...,
  property = "name",
  language = get_AMR_locale(),
  snake_case = NULL
)
```

## Arguments

| | |
|---|---|
| x | Any (vector of) text that can be coerced to a valid antibiotic drug code with as.ab(). |
| language | Language of the returned text - the default is the current system language (see get_AMR_locale()) and can also be set with the package option AMR_locale. Use language = NULL or language = "" to prevent translation. |
| tolower | A 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". |
| ... | In case of set_ab_names() and data is a data.frame: columns to select (supports tidy selection such as column1:column4), otherwise other arguments passed on to as.ab(). |
| all_groups | A logical to indicate whether all antimicrobial groups must be return as a vector for each input value. For example, an antibiotic in the "aminopenicillins" group, is also in the "penicillins" and "beta-lactams" groups. Setting all_groups = TRUE would return all three for such an antibiotic, while all_groups = FALSE (default) only returns the most specific group name. |
| only_first | A logical to indicate whether only the first ATC code must be returned, with giving preference to J0-codes (i.e., the antimicrobial drug group). |
| administration | Way of administration, either "oral" or "iv". |
| open | Browse the URL using utils::browseURL(). |
| property | One of the column names of one of the antimicrobials data set: "ab", "cid", "name", "group", "atc", "atc_group1", "atc_group2", "abbreviations", "synonyms", "oral_ddd", "oral_units", "iv_ddd", "iv_units", or "loinc". |
| data | A data.frame of which the columns need to be renamed, or a character vector of column names. |
| snake_case | A logical to indicate whether the names should be in so-called snake case: in lower case and all spaces/slashes replaced with an underscore (_). |

## Details

All output will be translated where possible.

The function `ab_url()` will return the direct URL to the official WHO website. A warning will be returned if the required ATC code is not available.

The function `set_ab_names()` is a special column renaming function for data.frames. It renames columns names that resemble antimicrobial drugs. It always makes sure that the new column names are unique. If `property = "atc"` is set, preference is given to ATC codes from the J-group.

## Value

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

## Source

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

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

## Download Our Reference Data

All reference data sets in the AMR package - including information on microorganisms, antimicrobials, and clinical breakpoints - are freely available for download in multiple formats: R, MS Excel, Apache Feather, Apache Parquet, SPSS, and Stata.

For maximum compatibility, we also provide machine-readable, tab-separated plain text files suitable for use in any software, including laboratory information systems.

Visit our website for direct download links, or explore the actual files in our GitHub repository.

## See Also

antimicrobials

## Examples

```
# all properties:
ab_name("AMX")
ab_atc("AMX")
ab_cid("AMX")
ab_synonyms("AMX")
ab_tradenames("AMX")
ab_group("AMX")
ab_group("AMX", all_groups = TRUE) # most specific to most general
ab_atc_group1("AMX")
```

```
ab_atc_group2("AMX")
ab_url("AMX")

# smart lowercase transformation
ab_name(x = c("AMC", "PLB"))
ab_name(x = c("AMC", "PLB"), tolower = TRUE)

# defined daily doses (DDD)
ab_ddd("AMX", "oral")
ab_ddd_units("AMX", "oral")
ab_ddd("AMX", "iv")
ab_ddd_units("AMX", "iv")

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")
ab_group("J01CA01")
ab_loinc("ampicillin")
ab_name("21066-6")
ab_name(6249)
ab_name("J01CA01")

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

# use set_ab_names() for renaming columns
colnames(example_isolates)
colnames(set_ab_names(example_isolates))
colnames(set_ab_names(example_isolates, NIT:VAN))

if (require("dplyr")) {
  example_isolates %>%
    set_ab_names()

  # this does the same:
  example_isolates %>%
    rename_with(set_ab_names)

  # set_ab_names() works with any AB property:
  example_isolates %>%
    set_ab_names(property = "atc")

  example_isolates %>%
    set_ab_names(where(is.sir)) %>%
    colnames()

  example_isolates %>%
    set_ab_names(NIT:VAN) %>%
    colnames()
```

```
}
```

---

`add_custom_antimicrobials` *Add Custom Antimicrobials*

---

## Description

With `add_custom_antimicrobials()` you can add your own custom antimicrobial drug names and codes.

## Usage

```
add_custom_antimicrobials(x)

add_custom_antimicrobial_synonyms(ab, synonyms = NULL)

clear_custom_antimicrobials()
```

## Arguments

| | |
|---|---|
| `x` | A data.frame resembling the antimicrobials data set, at least containing columns "ab" and "name". To add synonyms to existing antimicrobials, columns "ab" and "synonyms" suffice (see *Details*). |
| `ab` | For `add_custom_antimicrobial_synonyms()`: a character vector of existing antimicrobial codes (see `as.ab()`), or a data.frame with columns "ab" and "synonyms" (one synonym per row). |
| `synonyms` | A character vector of synonyms. If `ab` has length 1, all `synonyms` are added to that antimicrobial; otherwise `ab` and `synonyms` must have the same length. |

## Details

**Important:** Due to how R works, the `add_custom_antimicrobials()` function has to be run in every R session - added antimicrobials are not stored between sessions and are thus lost when R is exited.

There are two ways to circumvent this and automate the process of adding antimicrobials:

**Method 1:** Using the package option `AMR_custom_ab`, which is the preferred method. To use this method:

1. Create a data set in the structure of the antimicrobials data set (containing at the very least columns "ab" and "name", or only columns "ab" and "synonyms" for synonyms of existing antimicrobials) and save it with `saveRDS()` to a location of choice, e.g. `"~/my_custom_ab.rds"`, or any remote location.

2. Set the file location to the package option AMR_custom_ab: `options(AMR_custom_ab = "~/my_custom_ab.rds")`. This can even be a remote file location, such as an https URL. Since options are not saved between R sessions, it is best to save this option to the `.Rprofile` file so that it will be loaded on start-up of R. To do this, open the `.Rprofile` file using e.g. `utils::file.edit("~/.Rprofile")`, add this text and save the file:

   ```
   # Add custom antimicrobial codes:
   options(AMR_custom_ab = "~/my_custom_ab.rds")
   ```

   Upon package load, this file will be loaded and run through the `add_custom_antimicrobials()` function.

   This file can also hold synonyms for antimicrobials that already exist, such as local trade names: rows with an existing code in column "ab" and only column "synonyms" filled (one synonym per row, codes may repeat) are passed on to `add_custom_antimicrobial_synonyms()`. Rows with an existing code that have any other column filled still give an error. Synonyms must be valid UTF-8 text; convert other encodings first, e.g. with `iconv(x, from = "CP949", to = "UTF-8")`.

**Method 2:** Loading the antimicrobial additions directly from your `.Rprofile` file. Note that the definitions will be stored in a user-specific R file, which is a suboptimal workflow. To use this method:

1. Edit the `.Rprofile` file using e.g. `utils::file.edit("~/.Rprofile")`.
2. Add a text like below and save the file:

```
# Add custom antibiotic drug codes:
AMR::add_custom_antimicrobials(
  data.frame(ab = "TESTAB",
             name = "Test Antibiotic",
             group = "Test Group")
)
```

Use `add_custom_antimicrobial_synonyms()` to add extra names, such as local trade names, to antimicrobials that already exist (including antimicrobials added with `add_custom_antimicrobials()`). These synonyms are recognised by `as.ab()` and all `ab_*()` functions. Synonyms may contain non-ASCII characters (e.g. trade names in other scripts): they are matched against the input as given, before `as.ab()` transliterates the input to ASCII. Matching ignores case, all white space (including no-break and ideographic spaces) and zero-width characters. If the input does not match as given, a trailing part in parentheses or brackets (such as `(piperacillin, tazobactam)`) and a trailing strength written as a number with a unit symbol (such as `4.5g`, `4/0.5 g` or `500 mg/mL`) are removed one at a time, and matching is tried again after each removal, so that the longest registered synonym wins. Full-width characters are read as their ASCII equivalents. No words are removed, in any language: to recognise a name with a dosage form (such as `Inj`), add that variant as a synonym as well. Only user-added synonyms are matched this way, there is no other fuzzy matching for them, and a strength on its own is never matched to a user-added synonym. A synonym cannot be added if it already identifies another antimicrobial as its code, name, synonym, abbreviation, ATC code, CID or LOINC code, also when written with other punctuation (such as `J01-CR05`).

Use `clear_custom_antimicrobials()` to clear the previously added antimicrobials and synonyms.

## See Also

add_custom_microorganisms() to add custom microorganisms.

## Examples

```
# returns a wildly guessed result:
as.ab("testab")

# now add a custom entry - it will be considered by as.ab() and
# all ab_*() functions
add_custom_antimicrobials(
  data.frame(
    ab = "TESTAB",
    name = "Test Antibiotic",
    # you can add any property present in the
    # 'antimicrobials' data set, such as 'group':
    group = "Test Group"
  )
)

# "testab" is now a new antibiotic:
as.ab("testab")
ab_name("testab")
ab_group("testab")

ab_info("testab")


# Add Co-fluampicil, which is one of the many J01CR50 codes, see
# https://atcddd.fhi.no/ddd/list_of_ddds_combined_products/
add_custom_antimicrobials(
  data.frame(
    ab = "COFLU",
    name = "Co-fluampicil",
    atc = "J01CR50",
    group = "Beta-lactams/penicillins"
  )
)
ab_atc("Co-fluampicil")
ab_name("J01CR50")

# even antimicrobial selectors work
# see ?amr_selector
x <- data.frame(
  random_column = "some value",
  coflu = as.sir("S"),
  ampicillin = as.sir("R")
)
x
x[, betalactams()]

# add local trade names as synonyms to existing antimicrobials,
```

```
# here a Korean trade name of piperacillin/tazobactam
add_custom_antimicrobial_synonyms("TZP", "\uD0C0\uC870\uC2E0\uC8FC")
ab_name("\uD0C0\uC870\uC2E0\uC8FC")
# a trailing strength, as in many hospital exports, is ignored
as.ab("\uD0C0\uC870\uC2E0\uC8FC 4.5g")
```

---

`add_custom_microorganisms` *Add Custom Microorganisms*

---

## Description

With `add_custom_microorganisms()` you can add your own custom microorganisms, such the non-taxonomic outcome of laboratory analysis.

## Usage

```
add_custom_microorganisms(x)

clear_custom_microorganisms()
```

## Arguments

| | |
|---|---|
| x | A data.frame resembling the microorganisms data set, at least containing column "genus" (case-insensitive). |

## Details

This function will fill in missing taxonomy for you, if specific taxonomic columns are missing, see *Examples*.

**Important:** Due to how R works, the `add_custom_microorganisms()` function has to be run in every R session - added microorganisms are not stored between sessions and are thus lost when R is exited.

There are two ways to circumvent this and automate the process of adding microorganisms:

**Method 1:** Using the package option `AMR_custom_mo`, which is the preferred method. To use this method:

1. Create a data set in the structure of the microorganisms data set (containing at the very least column "genus") and save it with `saveRDS()` to a location of choice, e.g. `"~/my_custom_mo.rds"`, or any remote location.
2. Set the file location to the package option `AMR_custom_mo`: `options(AMR_custom_mo = "~/my_custom_mo.rds")`. This can even be a remote file location, such as an https URL. Since options are not saved between R sessions, it is best to save this option to the `.Rprofile` file so that it will be loaded on start-up of R. To do this, open the `.Rprofile` file using e.g. `utils::file.edit("~/.Rprofile")`, add this text and save the file:

```
# Add custom microorganism codes:
options(AMR_custom_mo = "~/my_custom_mo.rds")
```

Upon package load, this file will be loaded and run through the add_custom_microorganisms() function.

**Method 2:** Loading the microorganism directly from your .Rprofile file. Note that the definitions will be stored in a user-specific R file, which is a suboptimal workflow. To use this method:

1. Edit the .Rprofile file using e.g. utils::file.edit("~/.Rprofile").
2. Add a text like below and save the file:

```
# Add custom antibiotic drug codes:
AMR::add_custom_microorganisms(
  data.frame(genus = "Enterobacter",
             species = "asburiae/cloacae")
)
```

Use clear_custom_microorganisms() to clear the previously added microorganisms.

## See Also

add_custom_antimicrobials() to add custom antimicrobials.

## Examples

```
# a combination of species is not formal taxonomy, so
# this will result in "Enterobacter cloacae cloacae",
# since it resembles the input best:
mo_name("Enterobacter asburiae/cloacae")

# now add a custom entry - it will be considered by as.mo() and
# all mo_*() functions
add_custom_microorganisms(
  data.frame(
    genus = "Enterobacter",
    species = "asburiae/cloacae"
  )
)

# E. asburiae/cloacae is now a new microorganism:
mo_name("Enterobacter asburiae/cloacae")

# its code:
as.mo("Enterobacter asburiae/cloacae")

# all internal algorithms will work as well:
mo_name("Ent asburia cloacae")

# and even the taxonomy was added based on the genus!
mo_family("E. asburiae/cloacae")
mo_gramstain("Enterobacter asburiae/cloacae")
```

```
mo_info("Enterobacter asburiae/cloacae")


# the function tries to be forgiving:
add_custom_microorganisms(
  data.frame(
    GENUS = "BACTEROIDES / PARABACTEROIDES SLASHLINE",
    SPECIES = "SPECIES"
  )
)
mo_name("BACTEROIDES / PARABACTEROIDES")
mo_rank("BACTEROIDES / PARABACTEROIDES")

# taxonomy still works, even though a slashline genus was given as input:
mo_family("Bacteroides/Parabacteroides")


# for groups and complexes, set them as species or subspecies:
add_custom_microorganisms(
  data.frame(
    genus = "Citrobacter",
    species = c("freundii", "braakii complex"),
    subspecies = c("complex", "")
  )
)
mo_name(c("C. freundii complex", "C. braakii complex"))
mo_species(c("C. freundii complex", "C. braakii complex"))
mo_gramstain(c("C. freundii complex", "C. braakii complex"))
```

---

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), character (vectors) will be coerced with `as.POSIXlt()`. |
| reference | Reference date(s) (default is today), character (vectors) will be coerced with `as.POSIXlt()`. |
| 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. |
| ... | Arguments passed on to as.POSIXlt(), such as origin. |

## Details

Ages below 0 will be returned as NA with a warning. Ages above 120 will only give a warning.

This function vectorises over both x and reference, meaning that either can have a length of 1 while the other argument has a larger length.

## Value

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

## See Also

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

## Examples

```
# 10 random pre-Y2K birth dates
df <- data.frame(birth_date = as.Date("2000-01-01") - runif(10) * 25000)

# add ages
df$age <- age(df$birth_date)

# add exact ages
df$age_exact <- age(df$birth_date, exact = TRUE)

# add age at millenium switch
df$age_at_y2k <- age(df$birth_date, "2000-01-01")

df
```

---

# age_groups *Split Ages into Age Groups*

---

## Description

Split ages into age groups defined by the split argument. This allows for easier demographic (antimicrobial resistance) analysis. The function returns an ordered factor.

## Usage

```
age_groups(x, split_at = c(0, 12, 25, 55, 75), names = NULL, na.rm = FALSE)
```

## Arguments

| | |
|---|---|
| x | Age, e.g. calculated with age(). |
| split_at | Values to split x at - the default is age groups 0-11, 12-24, 25-54, 55-74 and 75+. See *Details*. |
| names | Optional names to be given to the various age groups. |
| na.rm | A logical to indicate whether missing values should be removed. |

## Details

To split ages, the input for the split_at argument can be:

- A numeric vector. A value of e.g. c(10, 20) will split x on 0-9, 10-19 and 20+. A value of only 50 will split x 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, ..., 95-99, 100+.
  - "tens", equivalent of: 1:10 * 10. This will split on 0-9, 10-19, ..., 90-99, 100+.

## Value

Ordered factor

## See Also

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

## Examples

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

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

# split into 0-19, 20-49 and 50+
age_groups(ages, c(20, 50))
age_groups(ages, c(20, 50), names = c("Under 20 years", "20 to 50 years", "Over 50 years"))

# 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, c(1, 2, 4, 6, 13, 18))
age_groups(ages, "children")


# resistance of ciprofloxacin per age group
if (require("dplyr") && require("ggplot2")) {
  example_isolates %>%
    filter_first_isolate() %>%
    filter(mo == as.mo("Escherichia coli")) %>%
    group_by(age_group = age_groups(age)) %>%
    select(age_group, CIP) %>%
    ggplot_sir(
      x = "age_group",
      minimum = 0,
      x.title = "Age Group",
      title = "Ciprofloxacin resistance per age group"
    )
}
```

---

`amr_course` *Download and Unpack an AMR Course Repository*

---

## Description

Downloads and unpacks a GitHub repository containing course materials, using `usethis::use_course()`. This is a convenience wrapper intended for use in educational settings, such as workshops or tutorials associated with the AMR package.

## Usage

```
amr_course(github_repo, branch = "main", ...)
```

## Arguments

| | |
|---|---|
| `github_repo` | A character string specifying the GitHub repository with username and repo name, e.g. `"https://github.com/username/repo"`. |
| `branch` | A character string specifying the branch to download. Defaults to `"main"`. |
| `...` | Additional arguments passed on to `usethis::use_course()`. |

## Details

This function constructs a ZIP archive URL from the provided `github_repo` and `branch`, then delegates to `usethis::use_course()` to handle the download and extraction.

The function is designed for interactive use in course or workshop settings and is not intended for use in non-interactive or automated pipelines.

## Value

Called for its side effect. `usethis::use_course()` will prompt the user to choose a destination and open the extracted project. Returns invisibly whatever `usethis::use_course()` returns.

## See Also

`usethis::use_course()`

## Examples

```
## Not run:

# Let this run by users, e.g., webinar participants
amr_course("https://github.com/my_user_name/our_AMR_course")

## End(Not run)
```

---

`antibiogram` *Generate Antibiograms (WISCA, Traditional, Combination, or Syndromic)*

---

## Description

Generate antibiograms from antimicrobial susceptibility data, with support for traditional, combination, syndromic, and WISCA (Weighted-Incidence Syndromic Combination Antibiogram) methods.

**For empirical therapy guidance, WISCA is the recommended approach.** When initiating empirical treatment, the causative pathogen is unknown, and the clinically relevant question is: *"what is the probability that this regimen will cover whatever pathogen turns out to cause the infection?"* WISCA answers that question directly by weighting susceptibility by pathogen incidence within a syndrome and providing credible intervals via Bayesian Monte Carlo simulation. Traditional antibiograms remain appropriate for tracking resistance per species for surveillance purposes. See the section *Explaining WISCA* on this page and the WISCA vignette for details.

All antibiogram types adhere to previously described approaches (see *Source*), and the WISCA method implements the Bayesian decision model by Bielicki *et al.* (2016, doi:10.1093/jac/dkv397). Output formats include plots and tables, ideal for integration with R Markdown and Quarto reports.

## Usage

```
wisca(
  x,
  antimicrobials = where(is.sir),
  ab_transform = "name",
  syndromic_group = NULL,
  only_all_tested = FALSE,
  digits = 1,
  formatting_type = getOption("AMR_antibiogram_formatting_type", 14),
```

```
  col_mo = NULL,
  language = get_AMR_locale(),
  combine_SI,
  guideline = getOption("AMR_guideline", "EUCAST"),
  sep = " + ",
  sort_columns = TRUE,
  simulations = 1000,
  conf_interval = 0.95,
  interval_side = "two-tailed",
  info = interactive(),
  parallel = FALSE,
  ...
)

antibiogram(
  x,
  antimicrobials = where(is.sir),
  mo_transform = "shortname",
  ab_transform = "name",
  syndromic_group = NULL,
  add_total_n = FALSE,
  only_all_tested = FALSE,
  digits = ifelse(wisca, 1, 0),
 formatting_type = getOption("AMR_antibiogram_formatting_type", ifelse(wisca, 14, 18)),
  col_mo = NULL,
  language = get_AMR_locale(),
  minimum = 30,
  combine_SI,
  guideline = getOption("AMR_guideline", "EUCAST"),
  sep = " + ",
  sort_columns = TRUE,
  wisca = FALSE,
  simulations = 1000,
  conf_interval = 0.95,
  interval_side = "two-tailed",
  info = interactive(),
  parallel = FALSE,
  ...
)

retrieve_wisca_parameters(wisca_model, ...)

## S3 method for class 'antibiogram'
plot(x, ...)

## S3 method for class 'antibiogram'
autoplot(
  object,
```

```
  geom = c("pointrange", "point", "col", "bar", "errorbar"),
  ci = TRUE,
  sort = TRUE,
  flip = NULL,
  caption = NULL,
  ...
)

wisca_plot(
  wisca_model,
  wisca_plot_type = c("susceptibility_incidence", "posterior_coverage"),
  ...
)

## S3 method for class 'antibiogram'
knit_print(
  x,
  italicise = TRUE,
  na = getOption("knitr.kable.NA", default = ""),
  ...
)
```

## Arguments

| | |
|---|---|
| x | A data.frame containing at least a column with microorganisms and columns with antimicrobial results (class 'sir', see `as.sir()`). |
| antimicrobials | A vector specifying the antimicrobials containing SIR values to include in the antibiogram (see *Examples*). Will be evaluated using `guess_ab_col()`. This can be:<br>• Any antimicrobial name or code that could match (see `guess_ab_col()`) to any column in `x`<br>• Any antimicrobial selector, such as `aminoglycosides()` or `carbapenems()`<br>• A combination of the above, using `c()`, e.g.:<br>– `c(aminoglycosides(), "AMP", "AMC")`<br>– `c(aminoglycosides(), carbapenems())`<br>• Column indices using numbers<br>• Combination therapy, indicated by using `"+"`, with or without antimicrobial selectors, e.g.:<br>– `"cipro + genta"`<br>– `"TZP+TOB"`<br>– `c("TZP", "TZP+GEN", "TZP+TOB")`<br>– `carbapenems() + "GEN"`<br>– `carbapenems() + c("", "GEN")`<br>– `carbapenems() + c("", aminoglycosides())` |
| ab_transform | A character to transform antimicrobial input - must be one of the column names of the antimicrobials data set (defaults to `"name"`): `"ab"`, `"cid"`, `"name"`, `"group"`, |

"atc", "atc_group1", "atc_group2", "abbreviations", "synonyms", "oral_ddd", "oral_units", "iv_ddd", "iv_units", or "loinc". Can also be NULL to not transform the input.

syndromic_group
: A column name of x, or values calculated to split rows of x, e.g. by using ifelse() or case_when(). See *Examples*.

only_all_tested
: (for combination antibiograms): a logical to indicate that isolates must be tested for all antimicrobials, see *Details*.

digits
: Number of digits to use for rounding the antimicrobial coverage, defaults to 1 for WISCA and 0 otherwise.

formatting_type
: Numeric value (1-22 for WISCA, 1-12 for non-WISCA) indicating how the 'cells' of the antibiogram table should be formatted. See *Details* > *Formatting Type* for a list of options.

col_mo
: Column name of the names or codes of the microorganisms (see as.mo()) - the default is the first column of class mo. Values will be coerced using as.mo().

language
: Language to translate text, which defaults to the system language (see get_AMR_locale()).

combine_SI
: *(deprecated in favour of* guideline*)* A logical. If set, it takes precedence over guideline: TRUE considers S, SDD, and I susceptible, and FALSE only S.

guideline
: A guideline name, "EUCAST" or "CLSI" (or any version thereof, e.g. "CLSI 2025"), to determine whether the 'I' category is considered susceptible (EUCAST) or resistant (CLSI). 'SDD' is always considered susceptible. The default is the package option AMR_guideline, or "EUCAST" if it is not set. See section *Interpretation of SIR* below.

sep
: A separating character for antimicrobial columns in combination antibiograms.

sort_columns
: A logical to indicate whether the antimicrobial columns must be sorted on name.

simulations
: (for WISCA) a numerical value to set the number of Monte Carlo simulations.

conf_interval
: A numerical value to set confidence interval (default is 0.95).

interval_side
: The side of the confidence interval, either "two-tailed" (default), "left" or "right".

info
: A logical to indicate info should be printed - the default is TRUE only in interactive mode.

parallel
: A logical to indicate if parallel computing must be used, defaults to FALSE. Requires the future.apply package. For WISCA, Monte Carlo simulations are distributed across workers; for grouped antibiograms, each group is processed by a separate worker. **A non-sequential future::plan() must already be active before setting** parallel = TRUE – for example, future::plan(future::multisession). An error is thrown if parallel = TRUE is used without a plan set by the user.

...
: Currently unused.

mo_transform
: A character to transform microorganism input - must be "name", "shortname" (default), "gramstain", or one of the column names of the microorganisms data set: "mo", "fullname", "status", "domain", "kingdom", "phylum", "class", "order", "family", "genus", "species", "subspecies", "rank", "ref", "oxygen_tolerance",

| | |
|---|---|
| | "morphology", "source", "lpsn", "lpsn_parent", "lpsn_renamed_to", "mycobank", "mycobank_parent", "mycobank_renamed_to", "gbif", "gbif_parent", "gbif_renamed_to", "prevalence", or "snomed". Can also be NULL to not transform the input or NA to consider all microorganisms 'unknown'. |
| add_total_n | *(deprecated in favour of* formatting_type*)* A logical to indicate whether n_tested available numbers per pathogen should be added to the table (default is TRUE). This will add the lowest and highest number of available isolates per antimicrobial (e.g., if for *E. coli* 200 isolates are available for ciprofloxacin and 150 for amoxicillin, the returned number will be "150-200"). This option is unavailable when wisca = TRUE; in that case, use retrieve_wisca_parameters() to get the parameters used for WISCA. |
| 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*. |
| wisca | A logical to indicate whether a Weighted-Incidence Syndromic Combination Antibiogram (WISCA) must be generated (default is FALSE). This will use a Bayesian decision model to estimate regimen coverage probabilities using Monte Carlo simulations. Per doi:10.1093/jac/dkv397, susceptibility priors are $\beta(0.5, 0.5)$ (Jeffreys) and intrinsically resistant pairs (based on intrinsic_resistant) use $\beta(1, 9999)$.<br>Set simulations, conf_interval, and interval_side to adjust. |
| wisca_model | The outcome of wisca() or antibiogram(..., wisca = TRUE). |
| object | An antibiogram() object. |
| geom | The plotting style for the point estimate. One of "pointrange" (default), "point", "col"/"bar", or "errorbar". "pointrange" is recommended for coverage data: bars imply a meaningful baseline at zero, which coverage estimates rarely have. |
| ci | Logical, whether to draw the credible/confidence interval. Defaults to TRUE. Ignored (forced TRUE) when geom = "pointrange" or "errorbar", since the interval is intrinsic to those geoms. |
| sort | Logical, whether to order regimens by coverage. Defaults to TRUE. When faceted (per pathogen) or grouped (syndromic), ordering is applied within each panel/group. |
| flip | Logical, whether to draw regimens on the y-axis (horizontal). Defaults to NULL, which flips automatically when any regimen label exceeds 20 characters (long combination names read poorly on the x-axis). Set TRUE/FALSE to override. |
| caption | Text to show as caption, will explain non-inferiority for WISCA models. |
| wisca_plot_type | Either "susceptibility_incidence" (default) or "posterior_coverage". |
| italicise | A logical to indicate whether the microorganism names in the knitr table should be made italic, using italicise_taxonomy(). |
| na | Character to use for showing NA values. |

## Details

These functions return a table with values between 0 and 100 for *susceptibility*, not resistance.

**Remember that you should filter your data to let it contain only first isolates!** This is needed to exclude duplicates and to reduce selection bias. Use `first_isolate()` to determine them with one of the four available algorithms: isolate-based, patient-based, episode-based, or phenotype-based.

For estimating antimicrobial coverage, especially when creating a WISCA, the outcome might become more reliable by only including the top *n* species encountered in the data. You can filter on this top *n* using `top_n_microorganisms()`. For example, use `top_n_microorganisms(your_data, n = 10)` as a pre-processing step to only include the top 10 species in the data.

The numeric values of an antibiogram are stored in a long format as the attribute `long_numeric`. You can retrieve them using `attributes(x)$long_numeric`, where `x` is the outcome of `antibiogram()` or `wisca()`. This is ideal for e.g. advanced plotting.

**Formatting Type:**

The formatting of the 'cells' of the table can be set with the argument `formatting_type`. In these examples, `5` indicates the antimicrobial coverage (`4-6` the confidence level), `15` the number of susceptible isolates, and `300` the number of tested (i.e., available) isolates:

1. 5
2. 15
3. 300
4. 15/300
5. 5 (300)
6. 5% (300)
7. 5 (N=300)
8. 5% (N=300)
9. 5 (15/300)
10. 5% (15/300)
11. 5 (N=15/300)
12. 5% (N=15/300)
13. 5 (4-6)
14. 5% (4-6%) - **default for WISCA**
15. 5 (4-6,300)
16. 5% (4-6%,300)
17. 5 (4-6,N=300)
18. 5% (4-6%,N=300) - **default for non-WISCA**
19. 5 (4-6,15/300)
20. 5% (4-6%,15/300)
21. 5 (4-6,N=15/300)
22. 5% (4-6%,N=15/300)

The default can be set globally with the package option `AMR_antibiogram_formatting_type`, e.g. `options(AMR_antibiogram_formatting_type = 5)`. Do note that for WISCA, the total numbers of tested and susceptible isolates are less useful to report, since these are included in the Bayesian model and apparent from the susceptibility and its confidence level.

Set `digits` (defaults to `0`) to alter the rounding of the susceptibility percentages.

**When to Use WISCA vs. Traditional Antibiograms:**

There are various antibiogram types, as summarised by Klinker *et al.* (2021, doi:10.1177/20499361211011373), and they are all supported by `antibiogram()`: traditional, combination, syndromic, and WISCA.

**If your goal is to guide empirical therapy, use WISCA.** Traditional antibiograms fragment susceptibility information by species, but at the point of prescribing, the clinician does not know which species is causing the infection. WISCA shifts the unit of analysis from the isolate to the patient: it estimates the probability that a regimen will cover the infection, given the local distribution of causative pathogens. It evaluates combination regimens, weights by pathogen incidence, and provides credible intervals that honestly communicate uncertainty. Hebert *et al.* (2012) demonstrated this concretely for the first time: ciprofloxacin showed 84% susceptibility against *E. coli* in the traditional antibiogram, but WISCA coverage was only 62% for UTI and 37% for abdominal infections, because other species (including intrinsically resistant enterococci) contribute substantially to these syndromes. Note that WISCA is pathogen-agnostic: the outcome is not stratified by species, but by syndrome.

**Traditional, combination, and syndromic antibiograms remain appropriate for AMR surveillance**, i.e., tracking resistance trends per species over time. They are the right tool when the question is *"how resistant is species X to drug Y in our setting?"* rather than *"what regimen best covers this syndrome?"*.

All four types are demonstrated in the *Examples* section below.

**Grouped tibbles:**

For any type of antibiogram, grouped tibbles can also be used to calculate susceptibilities over various groups.

Code example:

```
library(dplyr)
your_data %>%
  group_by(has_sepsis, is_neonate, sex) %>%
  wisca(antimicrobials = c("TZP", "TZP+TOB", "TZP+GEN"))
```

**Inclusion in Combination Antibiograms:**

For combination antibiograms, the results of the antimicrobials are combined using the same rules as `sir_combine()`, see the section *Combination Therapy* below. Whether isolates must be tested for all antimicrobials can be set with the `only_all_tested` argument (default is `FALSE`). Whether the 'I' category is considered susceptible follows the `guideline` argument, which also determines whether a missing result can be ignored. To use a combination in other analyses as well, create a new variable with `sir_combine()`.

**Plotting:**

All types of antibiograms as listed above can be plotted (using `ggplot2::autoplot()` or base R's `plot()` and `barplot()`). As mentioned above, the numeric values of an antibiogram are stored in a long format as the attribute `long_numeric`. You can retrieve them using `attributes(x)$long_numeric`, where `x` is the outcome of `antibiogram()` or `wisca()`.

The outcome of `antibiogram()` can also be used directly in R Markdown / Quarto (i.e., `knitr`) for reports. In this case, `knitr::kable()` will be applied automatically and microorganism names will even be printed in italics at default (see argument `italicise`).

You can also use functions from specific 'table reporting' packages to transform the output of `antibiogram()` to your needs, e.g. with `flextable::as_flextable()` or `gt::gt()`.

### Explaining WISCA

WISCA (Weighted-Incidence Syndromic Combination Antibiogram) estimates the probability that an empirical antimicrobial regimen will provide adequate coverage for a given infection syndrome, before the causative pathogen has been identified.

It does so by combining two quantities: the relative incidence of each pathogen within the syndrome (modelled as a Dirichlet distribution) and the susceptibility of each pathogen to the regimen (modelled as Beta distributions). These are combined via Monte Carlo simulation to produce a coverage estimate with a credible interval.

**Prior distributions:** Pathogen incidence uses a non-informative $Dirichlet(1, 1, \ldots, 1)$ prior. Susceptibility proportions use the Jeffreys prior, $\beta(0.5, 0.5)$, except for pathogen-drug combinations with known intrinsic resistance, which use a strongly informative $\beta(1, 9999)$ prior that forces near-zero susceptibility regardless of observed data. Intrinsic resistance is determined using the intrinsic_resistant data set, which is based on 'EUCAST Expected Resistant Phenotypes' v1.2 (2023).

**Interpreting the output:** Overlapping credible intervals between regimens indicate no significant difference in coverage; if a narrower-spectrum regimen overlaps with a broader one, the narrower-spectrum option may be preferred on stewardship grounds. Non-overlapping intervals indicate a clinically meaningful difference. For small sample sizes, consider pooling data from multiple sites to improve precision, provided pathogen distributions are sufficiently similar (Bielicki *et al.*, 2016).

For the full mathematical derivation and worked examples, see the WISCA vignette.

### Combination Therapy

Results are ordered from susceptible to resistant: S (and WT), SDD, I, R (and NWT, NS). With `method = "most_susceptible"` (in `sir_combine()`), the most susceptible result of the antimicrobials is returned, and with `method = "most_resistant"`, the most resistant result. The value `"NI"` (non-interpretable) is considered a missing result.

A missing result can only be ignored if it cannot change the conclusion: a susceptible result for `"most_susceptible"`, and a non-susceptible result for `"most_resistant"`. Under EUCAST, the 'I' category is considered susceptible (*susceptible, increased exposure*), and under CLSI, it is not. With `only_all_tested = TRUE`, any missing result leads to `NA`.

This is an example for two antimicrobials, Drug A and Drug B, under EUCAST (the order of the antimicrobials does not matter):

| | | Result of method = "most_susceptible" | | Result of method = "most_resistant" | |
|---|---|---|---|---|---|
| | | only_all_tested = | | only_all_tested = | |
| Drug A | Drug B | FALSE | TRUE | FALSE | TRUE |
| S | S | S | S | S | S |
| S | I | S | S | I | I |
| S | R | S | S | R | R |
| I | R | I | I | R | R |
| R | R | R | R | R | R |
| S | <NA> | S | <NA> | <NA> | <NA> |

```
   I       <NA>          I         <NA>          <NA>          <NA>
   R       <NA>        <NA>        <NA>            R           <NA>
 <NA>      <NA>        <NA>        <NA>          <NA>          <NA>
-------------------------------------------------------------------------
```

Under CLSI, the combination of I with a missing result is <NA> for "most_susceptible" and I for "most_resistant".

The count_*() and proportion_*() functions apply these rules when more than one variable is used. With only_all_tested = FALSE (default), the functions for susceptibility, such as susceptibility() and count_susceptible(), count an isolate as susceptible if any of the antimicrobials is susceptible ("most_susceptible"), and the functions for resistance, such as resistance() and count_resistant(), count an isolate as resistant if any of the antimicrobials is resistant ("most_resistant"). An isolate is considered tested if it has such a result, or if it was tested for all antimicrobials. Therefore, for combinations with only_all_tested = FALSE:

```
   count_S()    +   count_I()    +   count_R()    >= count_all()
proportion_S() + proportion_I() + proportion_R() >= 1
```

With only_all_tested = TRUE, all these functions count the result of the combination therapy ("most_susceptible") of isolates tested for all antimicrobials, so that:

```
   count_S()    +   count_I()    +   count_R()    = count_all()
proportion_S() + proportion_I() + proportion_R() = 1
```

In antibiogram() and wisca(), the result of the combination therapy ("most_susceptible") is used, the 'I' category is considered susceptible or not following the guideline argument ('SDD' is always considered susceptible), and the result is then reduced to S or R.

### Interpretation of SIR

In 2019, the European Committee on Antimicrobial Susceptibility Testing (EUCAST) has decided to change the definitions of susceptibility testing categories S, I, and R (https://www.eucast.org/bacteria/clinical-breakpoints-and-interpretation/definition-of-s-i-and-r/).

In aggregated analyses, such as proportions, counts, antibiograms, and plots, the 'I' category must be considered either susceptible or resistant. In epidemiological AMR data analysis, this follows the guideline: EUCAST considers 'I' susceptible, while CLSI considers 'I' resistant. The dose-dependent susceptibility of CLSI ('SDD') is always considered susceptible, as it equals what EUCAST defines as 'I'. All functions with a guideline argument for this purpose use these categories:

| guideline | Susceptible | Resistant |
|---|---|---|
| "EUCAST" | S, SDD, I, WT | R, NWT, NS |
| "CLSI" | S, SDD, WT | I, R, NWT, NS |

The values 'WT' (wild type) and 'NWT' (non-wild type) result from epidemiological cut-off values (ECOFFs) instead of clinical breakpoints. Since they are no clinical categories, 'WT' is always considered susceptible and 'NWT' always resistant, regardless of the guideline. The value 'NS'

(non-susceptible) is always considered resistant, and 'NI' (non-interpretable) is always considered a missing result.

The `guideline` argument can be:

- **Left unset**, to use the package option `AMR_guideline`, or `"EUCAST"` if this option is not set. In the latter case, a message is shown once per session (but not in R Markdown or Quarto documents); set the option to silence it.
- `"EUCAST"`, or any EUCAST version such as `"EUCAST 2025"`, to consider 'I' susceptible, e.g., to return `"SI"` vs. `"R"` in `proportion_df()`.
- `"CLSI"`, or any CLSI version such as `"CLSI 2025"`, to consider 'I' resistant, e.g., to return `"S"` vs. `"IR"` in `proportion_df()`.
- `NULL` **or** `FALSE`, to keep S, I, and R separate. This is only possible in functions that return results per category: `count_df()`, `proportion_df()`, `sir_df()`, `ggplot_sir()`, `geom_sir()`, `labels_sir_count()`, and `bug_drug_combinations()`. Functions that return a single proportion or count, such as `susceptibility()`, `count_resistant()`, `antibiogram()`, `wisca()`, and `format()` on the result of `bug_drug_combinations()`, require 'I' to be considered either susceptible or resistant, and return an error.

Use `susceptibility()` and `resistance()` (or `count_susceptible()` and `count_resistant()`) to follow the guideline, or e.g. `proportion_SI()` and `proportion_R()` to be explicit regardless of the guideline.

The `combine_SI` argument of the functions that return results per category, `antibiogram()`, and `wisca()` is deprecated in favour of `guideline`. If set, it is followed literally and takes precedence over `guideline`.

## Author(s)

Implementation: Dr. Larisse Bolton and Dr. Matthijs Berends

## References


- Hebert C *et al.* (2012). **Demonstration of the weighted-incidence syndromic combination antibiogram: an empiric prescribing decision aid.** *Infection Control & Hospital Epidemiology* 33(4):381-388; doi:10.1086/664768
- Bielicki JA *et al.* (2016). **Selecting appropriate empirical antibiotic regimens for paediatric bloodstream infections: application of a Bayesian decision model to local and pooled antimicrobial resistance surveillance data.** *Journal of Antimicrobial Chemotherapy* 71(3):794-802; doi:10.1093/jac/dkv397
- Cook A *et al.* (2022). **Improving empiric antibiotic prescribing in pediatric bloodstream infections: a potential application of weighted-incidence syndromic combination antibiograms (WISCA).** *Expert Review of Anti-infective Therapy* 20(3):445-456; doi:10.1080/14787210.2021.1967145
- Klinker KP *et al.* (2021). **Antimicrobial stewardship and antibiograms: importance of moving beyond traditional antibiograms.** *Therapeutic Advances in Infectious Disease*, May 5;8:20499361211011373; doi:10.1177/20499361211011373

- Barbieri E *et al.* (2021). **Development of a Weighted-Incidence Syndromic Combination Antibiogram (WISCA) to guide the choice of the empiric antibiotic treatment for urinary tract infection in paediatric patients: a Bayesian approach.** *Antimicrobial Resistance & Infection Control* May 1;10(1):74; doi:10.1186/s13756021009392
- **M39 Analysis and Presentation of Cumulative Antimicrobial Susceptibility Test Data, 5th Edition**, 2022, *Clinical and Laboratory Standards Institute (CLSI)*. https://clsi.org/standards/products/microbiology/documents/m39/.


## Examples

```
# example_isolates is a data set available in the AMR package.
# run ?example_isolates for more info.
example_isolates


# WISCA antibiogram (recommended for empirical therapy) -----------------

# basic WISCA: empirical coverage per regimen, weighted by pathogen
# incidence, with 95% credible intervals
wisca(example_isolates,
  antimicrobials = c("AMC", "AMC+CIP", "AMC+GEN")
)

# equivalent using antibiogram():
antibiogram(example_isolates,
  antimicrobials = c("AMC", "AMC+CIP", "AMC+GEN"),
  wisca = TRUE
)

# stratified by syndrome or clinical group (5 most common species, for a readable plot)
out <- wisca(top_n_microorganisms(example_isolates, n = 5),
  antimicrobials = c("TZP", "TZP+TOB", "TZP+GEN"),
  syndromic_group = "ward"
)
out
if (requireNamespace("ggplot2")) {
  wisca_plot(out)
}

# stratified using grouped tibbles (e.g. by age and gender)
if (requireNamespace("dplyr")) {
  library(dplyr)
  example_isolates %>%
    top_n_microorganisms(n = 10) %>%
    group_by(
      age_group = age_groups(age, c(25, 50, 75)),
      gender
    ) %>%
    wisca(antimicrobials = c("TZP", "TZP+TOB", "TZP+GEN"))
}
```

```
# Traditional antibiogram (for AMR surveillance) ------------------------

antibiogram(example_isolates,
  antimicrobials = c(aminoglycosides(), carbapenems())
)

antibiogram(example_isolates,
  antimicrobials = aminoglycosides(),
  ab_transform = "atc",
  mo_transform = "gramstain"
)


# Combination antibiogram (for AMR surveillance) ------------------------

antibiogram(example_isolates,
  antimicrobials = c("TZP", "TZP+TOB", "TZP+GEN"),
  mo_transform = "gramstain"
)

# you can use any antimicrobial selector with `+` too:
antibiogram(example_isolates,
  antimicrobials = ureidopenicillins() + c("", "GEN", "tobra"),
  mo_transform = "gramstain"
)

# names of antimicrobials do not need to resemble columns exactly:
antibiogram(example_isolates,
  antimicrobials = c("Cipro", "cipro + genta"),
  mo_transform = "gramstain",
  ab_transform = "name",
  sep = " & "
)


# Syndromic antibiogram (for AMR surveillance) --------------------------

antibiogram(example_isolates,
  antimicrobials = c(aminoglycosides(), carbapenems()),
  syndromic_group = "ward"
)

# with a custom language, though this will be determined automatically
# (i.e., this table will be in Spanish on Spanish systems)
ex1 <- example_isolates[which(mo_genus() == "Escherichia"), ]
antibiogram(ex1,
  antimicrobials = aminoglycosides(),
  ab_transform = "name",
  syndromic_group = ifelse(ex1$ward == "ICU",
    "UCI", "No UCI"
  ),
  language = "es"
)
```

```
# Print the output for R Markdown / Quarto -----------------------------

ureido <- wisca(example_isolates,
  antimicrobials = ureidopenicillins(),
  syndromic_group = "ward"
)

# in an Rmd file, you would just need to return `ureido` in a chunk,
# but to be explicit here:
if (requireNamespace("knitr")) {
  cat(knitr::knit_print(ureido))
}


# Generate plots with ggplot2 or base R --------------------------------

ab1 <- antibiogram(example_isolates,
  antimicrobials = c("AMC", "CIP", "TZP", "TZP+TOB"),
  mo_transform = "gramstain"
)
ab2 <- wisca(example_isolates,
  antimicrobials = c("AMC", "CIP", "TZP", "TZP+TOB"),
  syndromic_group = "ward"
)

if (requireNamespace("ggplot2")) {
  ggplot2::autoplot(ab1)
}
if (requireNamespace("ggplot2")) {
  ggplot2::autoplot(ab2)
}

plot(ab1)
plot(ab2)
```

---

`antimicrobials` *Data Sets with 626 Antimicrobial Drugs*

---

## Description

Two data sets containing all antimicrobials and antivirals. Use `as.ab()` or one of the `ab_*` functions to retrieve values from the antimicrobials data set. Three identifiers are included in this data set: an antimicrobial 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. Note that some drugs have multiple ATC codes.

**The `antibiotics` data set has been renamed to `antimicrobials`. The old name will be removed in a future version.**

## Usage

```
antimicrobials

antibiotics

antivirals
```

## Format

**For the antimicrobials data set: a tibble with 506 observations and 14 variables::**

- `ab`
  antimicrobial ID as used in this package (such as `AMC`), using the official EARS-Net (European Antimicrobial Resistance Surveillance Network) codes where available. **This is a unique identifier.**
- `cid`
  Compound ID as found in PubChem. **This is a unique identifier.**
- `name`
  Official name as used by WHONET/EARS-Net or the WHO. **This is a unique identifier.**
- `group`
  One or more short and concise group names, based on WHONET and WHOCC definitions
- `atc`
  ATC codes (Anatomical Therapeutic Chemical) as defined by the WHOCC, like `J01CR02` (last updated 4th of May, 2025):
- `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 antimicrobial susceptibility testing (AST)
- `synonyms`
  Synonyms (often trade names) of a drug, as found in PubChem based on their compound ID

ATC properties (last updated 4th of May, 2025):

- `oral_ddd`
  Defined Daily Dose (DDD), oral treatment, currently available for 180 drugs
- `oral_units`
  Units of `oral_ddd`
- `iv_ddd`
  Defined Daily Dose (DDD), parenteral (intravenous) treatment, currently available for 153 drugs
- `iv_units`
  Units of `iv_ddd`

LOINC:

- `loinc`
  All codes associated with the name of the antimicrobial drug from Logical Observation Identifiers Names and Codes (LOINC), Version 2.76 (18 September, 2023). Use `ab_loinc()` to retrieve them quickly, see `ab_property()`.

**For the antivirals data set: a tibble with 120 observations and 11 variables::**

- `av`
  Antiviral ID as used in this package (such as `ACI`), using the official EARS-Net (European Antimicrobial Resistance Surveillance Network) codes where available. **This is a unique identifier.** Combinations are codes that contain a `+` to indicate this, such as `ATA+COBI` for atazanavir/cobicistat.
- `name`
  Official name as used by WHONET/EARS-Net or the WHO. **This is a unique identifier.**
- `atc`
  ATC codes (Anatomical Therapeutic Chemical) as defined by the WHOCC, see *Details*
- `cid`
  Compound ID as found in PubChem. **This is a unique identifier.**
- `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`
- `loinc`
  All codes associated with the name of the antiviral drug from Logical Observation Identifiers Names and Codes (LOINC), Version 2.76 (18 September, 2023). Use `av_loinc()` to retrieve them quickly, see `av_property()`.

An object of class `deprecated_amr_dataset` (inherits from `tbl_df`, `tbl`, `data.frame`) with 506 rows and 14 columns.

An object of class `tbl_df` (inherits from `tbl`, `data.frame`) with 120 rows and 11 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`. Do note that ATC codes are not unique. For example, J01CR02 is officially the ATC code for "amoxicillin and beta-lactamase inhibitor". Consequently, these two items from the antimicrobials data set both return `"J01CR02"`:

```
ab_atc("amoxicillin/clavulanic acid")
ab_atc("amoxicillin/sulbactam")
```

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

### Download Our Reference Data

All reference data sets in the AMR package - including information on microorganisms, antimicrobials, and clinical breakpoints - are freely available for download in multiple formats: R, MS Excel, Apache Feather, Apache Parquet, SPSS, and Stata.

For maximum compatibility, we also provide machine-readable, tab-separated plain text files suitable for use in any software, including laboratory information systems.

Visit our website for direct download links, or explore the actual files in our GitHub repository.

### WHOCC

This package contains **all ~550 antibiotic, antimycotic and antiviral drugs** and their Anatomical Therapeutic Chemical (ATC) codes, ATC groups and Defined Daily Dose (DDD) from the World Health Organization Collaborating Centre for Drug Statistics Methodology (WHOCC, https://atcddd.fhi.no) and the Pharmaceuticals Community Register of the European Commission (https://ec.europa.eu/health/documents/community-register/html/index_en.htm).

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

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

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

### References


- WHO Collaborating Centre for Drug Statistics Methodology, Guidelines for ATC classification and DDD assignment, Oslo Accessed from https://atcddd.fhi.no/atc_ddd_index/ on 4th of May, 2025.
- Logical Observation Identifiers Names and Codes (LOINC), Version 2.76 (18 September, 2023). Accessed from https://loinc.org on 19th of October, 2023.
- European Commission Public Health PHARMACEUTICALS - COMMUNITY REGISTER: https://ec.europa.eu/health/documents/community-register/html/index_en.htm


### See Also

microorganisms, intrinsic_resistant

### Examples

```
antimicrobials
antivirals
```

`antimicrobial_selectors` *Antimicrobial Selectors*

## Description

These functions allow for filtering rows and selecting columns based on antimicrobial test results that are of a specific antimicrobial class or group, without the need to define the columns or antimicrobial abbreviations. They can be used in base R, tidyverse, tidymodels, and `data.table`.

Simply puy, if you have a column name that resembles an antimicrobial drug, it will be picked up by any of these functions that matches its pharmaceutical class, code or name: column names "cefazolin", "kefzol", "CZO" and "J01DB04" would all be included in the following selection:

```
library(dplyr)
my_data_with_all_these_columns %>%
  select(cephalosporins())
```

## Usage

```
aminoglycosides(
  only_sir_columns = FALSE,
  only_treatable = TRUE,
  return_all = TRUE,
  ...
)

aminopenicillins(only_sir_columns = FALSE, return_all = TRUE, ...)

antifungals(only_sir_columns = FALSE, return_all = TRUE, ...)

antimycobacterials(only_sir_columns = FALSE, return_all = TRUE, ...)

betalactams(
  only_sir_columns = FALSE,
  only_treatable = TRUE,
  return_all = TRUE,
  ...
)

betalactams_with_inhibitor(only_sir_columns = FALSE, return_all = TRUE, ...)

carbapenems(
  only_sir_columns = FALSE,
  only_treatable = TRUE,
  return_all = TRUE,
  ...
```

```
)

cephalosporins(
  only_sir_columns = FALSE,
  only_treatable = TRUE,
  return_all = TRUE,
  ...
)

cephalosporins_1st(only_sir_columns = FALSE, return_all = TRUE, ...)

cephalosporins_2nd(only_sir_columns = FALSE, return_all = TRUE, ...)

cephalosporins_3rd(
  only_sir_columns = FALSE,
  only_treatable = TRUE,
  return_all = TRUE,
  ...
)

cephalosporins_4th(only_sir_columns = FALSE, return_all = TRUE, ...)

cephalosporins_5th(only_sir_columns = FALSE, return_all = TRUE, ...)

fluoroquinolones(
  only_sir_columns = FALSE,
  only_treatable = TRUE,
  return_all = TRUE,
  ...
)

glycopeptides(only_sir_columns = FALSE, return_all = TRUE, ...)

ionophores(only_sir_columns = FALSE, return_all = TRUE, ...)

isoxazolylpenicillins(
  only_sir_columns = FALSE,
  only_treatable = TRUE,
  return_all = TRUE,
  ...
)

lincosamides(
  only_sir_columns = FALSE,
  only_treatable = TRUE,
  return_all = TRUE,
  ...
)
```

```
lipoglycopeptides(only_sir_columns = FALSE, return_all = TRUE, ...)

macrolides(only_sir_columns = FALSE, return_all = TRUE, ...)

monobactams(only_sir_columns = FALSE, return_all = TRUE, ...)

nitrofurans(only_sir_columns = FALSE, return_all = TRUE, ...)

oxazolidinones(only_sir_columns = FALSE, return_all = TRUE, ...)

penicillins(only_sir_columns = FALSE, return_all = TRUE, ...)

peptides(only_sir_columns = FALSE, return_all = TRUE, ...)

phenicols(only_sir_columns = FALSE, return_all = TRUE, ...)

phosphonics(only_sir_columns = FALSE, return_all = TRUE, ...)

polymyxins(
  only_sir_columns = FALSE,
  only_treatable = TRUE,
  return_all = TRUE,
  ...
)

quinolones(
  only_sir_columns = FALSE,
  only_treatable = TRUE,
  return_all = TRUE,
  ...
)

rifamycins(only_sir_columns = FALSE, return_all = TRUE, ...)

spiropyrimidinetriones(only_sir_columns = FALSE, return_all = TRUE, ...)

streptogramins(only_sir_columns = FALSE, return_all = TRUE, ...)

sulfonamides(only_sir_columns = FALSE, return_all = TRUE, ...)

tetracyclines(
  only_sir_columns = FALSE,
  only_treatable = TRUE,
  return_all = TRUE,
  ...
)
```

```
trimethoprims(only_sir_columns = FALSE, return_all = TRUE, ...)

ureidopenicillins(only_sir_columns = FALSE, return_all = TRUE, ...)

amr_class(
  amr_class,
  only_sir_columns = FALSE,
  only_treatable = TRUE,
  return_all = TRUE,
  ...
)

amr_selector(
  filter,
  only_sir_columns = FALSE,
  only_treatable = TRUE,
  return_all = TRUE,
  ...
)

administrable_per_os(only_sir_columns = FALSE, return_all = TRUE, ...)

administrable_iv(only_sir_columns = FALSE, return_all = TRUE, ...)

not_intrinsic_resistant(
  only_sir_columns = FALSE,
  col_mo = NULL,
  version_expected_phenotypes = 1.2,
  ...
)
```

## Arguments

| | |
|---|---|
| only_sir_columns | A logical to indicate whether only antimicrobial columns must be included that were transformed to class sir on beforehand. Defaults to FALSE. |
| only_treatable | A logical to indicate whether antimicrobial drugs should be excluded that are only for laboratory tests (default is TRUE), such as gentamicin-high (GEH) and imipenem/EDTA (IPE). |
| return_all | A logical to indicate whether all matched columns must be returned (default is TRUE). With FALSE, only the first of each unique antimicrobial will be returned, e.g. if both columns "genta" and "gentamicin" exist in the data, only the first hit for gentamicin will be returned. |
| ... | Ignored, only in place to allow future extensions. |
| amr_class | An antimicrobial class or a part of it, such as "carba" and "carbapenems". The columns group, atc_group1 and atc_group2 of the antimicrobials data set will be searched (case-insensitive) for this value. |

`filter` An expression to be evaluated in the antimicrobials data set, such as `name %like% "trim"`.

`col_mo` Column name of the names or codes of the microorganisms (see `as.mo()`) - the default is the first column of class `mo`. Values will be coerced using `as.mo()`.

`version_expected_phenotypes` The version number to use for the EUCAST Expected Phenotypes. Can be `"1.2"`.

## Details

These functions can be used in data set calls for selecting columns and filtering rows. They work with base R, the Tidyverse, and `data.table`. They are heavily inspired by the Tidyverse selection helpers such as `everything()`, but are not limited to `dplyr` verbs. Nonetheless, they are very convenient to use with `dplyr` functions such as `select()`, `filter()` and `summarise()`, see *Examples*.

All selectors can also be used in `tidymodels` packages such as `recipe` and `parsnip`. See for more info our tutorial on using antimicrobial selectors for predictive modelling.

All columns in the data in which these functions are called will be searched for known antimicrobial names, abbreviations, brand names, and codes (ATC, EARS-Net, WHO, etc.) according to the antimicrobials data set. This means that a selector such as `aminoglycosides()` will pick up column names like 'gen', 'genta', 'J01GB03', 'tobra', 'Tobracin', etc.

The `amr_class()` function can be used to filter/select on a manually defined antimicrobial class. It searches for results in the antimicrobials data set within the columns `group`, `atc_group1` and `atc_group2`.

The `administrable_per_os()` and `administrable_iv()` functions also rely on the antimicrobials data set - antimicrobials will be matched where a DDD (defined daily dose) for resp. oral and IV treatment is available in the antimicrobials data set.

The `amr_selector()` function can be used to internally filter the antimicrobials data set on any results, see *Examples*. It allows for filtering on a (part of) a certain name, and/or a group name or even a minimum of DDDs for oral treatment. This function yields the highest flexibility, but is also the least user-friendly, since it requires a hard-coded filter to set.

The `not_intrinsic_resistant()` function can be used to only select antimicrobials that pose no intrinsic resistance for the microorganisms in the data set. For example, if a data set contains only microorganism codes or names of *E. coli* and *K. pneumoniae* and contains a column "vancomycin", this column will be removed (or rather, unselected) using this function. It currently applies 'EUCAST Expected Resistant Phenotypes' v1.2 (2023) to determine intrinsic resistance, using the `eucast_rules()` function internally. Because of this determination, this function is quite slow in terms of performance.

## Value

When used inside selecting or filtering, this returns a character vector of column names, with additional class `"amr_selector"`. When used individually, this returns an 'ab' vector with all possible antimicrobials that the function would be able to select or filter.

**Full list of supported (antimicrobial) classes**

- `aminoglycosides()` can select:
  amikacin (AMK), amikacin/fosfomycin (AKF), apramycin (APR), arbekacin (ARB), astromicin (AST), bekanamycin (BEK), dibekacin (DKB), framycetin (FRM), gentamicin (GEN), gentamicin-high (GEH), habekacin (HAB), hygromycin (HYG), isepamicin (ISE), kanamycin (KAN), kanamycin-high (KAH), kanamycin/cephalexin (KAC), kasugamycin (KAS), micronomicin (MCR), neomycin (NEO), netilmicin (NET), pentisomicin (PIM), plazomicin (PLZ), propikacin (PKA), ribostamycin (RST), sisomicin (SIS), streptoduocin (STR), streptomycin (STR1), streptomycin-high (STH), tobramycin (TOB), and tobramycin-high (TOH)
- `aminopenicillins()` can select:
  amoxicillin (AMX), amoxicillin/clavulanic acid (AMC), and ampicillin (AMP)
- `antifungals()` can select:
  amorolfine (AMO), amphotericin B (AMB), amphotericin B-high (AMH), anidulafungin (ANI), butoconazole (BUT), caspofungin (CAS), ciclopirox (CIX), clotrimazole (CTR), econazole (ECO), fluconazole (FLU), flucytosine (FCT), fosfluconazole (FFL), griseofulvin (GRI), hachimycin (HCH), ibrexafungerp (IBX), isavuconazole (ISV), isoconazole (ISO), itraconazole (ITR), ketoconazole (KET), manogepix (MGX), micafungin (MIF), miconazole (MCZ), nystatin (NYS), oteseconazole (OTE), pimaricin (PMR), posaconazole (POS), rezafungin (RZF), ribociclib (RBC), sulconazole (SUC), terbinafine (TRB), terconazole (TRC), and voriconazole (VOR)
- `antimycobacterials()` can select:
  4-aminosalicylic acid (AMA), calcium aminosalicylate (CLA), capreomycin (CAP), clofazimine (CLF), delamanid (DLM), enviomycin (ENV), ethambutol (ETH), ethambutol/isoniazid (ETI), ethionamide (ETI1), isoniazid (INH), isoniazid/sulfamethoxazole/trimethoprim/pyridoxine (IST), morinamide (MRN), p-aminosalicylic acid (PAS), pretomanid (PMD), protionamide (PTH), pyrazinamide (PZA), rifabutin (RIB), rifampicin (RIF), rifampicin/ethambutol/isoniazid (REI), rifampicin/isoniazid (RFI), rifampicin/pyrazinamide/ethambutol/isoniazid (RPEI), rifampicin/pyrazinamide/isoniazid (RPI), rifamycin (RFM), rifapentine (RFP), sodium aminosalicylate (SDA), streptomycin/isoniazid (STI), terizidone (TRZ), thioacetazone (TAT), thioacetazone/isoniazid (THI1), tiocarlide (TCR), and viomycin (VIO)
- `betalactams()` can select:
  amoxicillin (AMX), amoxicillin/clavulanic acid (AMC), amoxicillin/sulbactam (AXS), ampicillin (AMP), ampicillin/sulbactam (SAM), apalcillin (APL), aspoxicillin (APX), azidocillin (AZD), azlocillin (AZL), aztreonam (ATM), aztreonam/avibactam (AZA), aztreonam/nacubactam (ANC), bacampicillin (BAM), benzathine benzylpenicillin (BNB), benzathine phenoxymethylpenicillin (BNP), benzylpenicillin (PEN), benzylpenicillin screening test (PEN-S), biapenem (BIA), carbenicillin (CRB), carindacillin (CRN), carumonam (CAR), cefacetrile (CAC), cefaclor (CEC), cefadroxil (CFR), cefalexin (LEX), cefaloridine (RID), cefalotin (CEP), cefamandole (MAN), cefapirin (HAP), cefatrizine (CTZ), cefazedone (CZD), cefazolin (CZO), cefcapene (CCP), cefcapene pivoxil (CCX), cefdinir (CDR), cefditoren (DIT), cefditoren pivoxil (DIX), cefepime (FEP), cefepime/amikacin (CFA), cefepime/clavulanic acid (CPC), cefepime/enmetazobactam (FPE), cefepime/nacubactam (FNC), cefepime/taniborbactam (FTA), cefepime/tazobactam (FPT), cefepime/zidebactam (FPZ), cefetamet (CAT), cefetamet pivoxil (CPI), cefetecol (CCL), cefetrizole (CZL), cefiderocol (FDC), cefixime (CFM), cefmenoxime (CMX), cefmetazole (CMZ), cefodizime (DIZ), cefonicid (CID), cefoperazone (CFP), cefoperazone/sulbactam (CSL), ceforanide (CND), cefoselis (CSE), cefotaxime (CTX), cefotaxime screening test (CTX-S), cefotaxime/clavulanic acid (CTC), cefotaxime/sulbactam (CTS), cefotetan (CTT), cefotiam

(CTF), cefotiam hexetil (CHE), cefovecin (FOV), cefoxitin (FOX), cefoxitin screening test (FOX-S), cefozopran (ZOP), cefpimizole (CFZ), cefpiramide (CPM), cefpirome (CPO), cefpodoxime (CPD), cefpodoxime proxetil (CPX), cefpodoxime/clavulanic acid (CDC), cefprozil (CPR), cefquinome (CEQ), cefroxadine (CRD), cefsulodin (CFS), cefsumide (CSU), ceftaroline (CPT), ceftaroline/avibactam (CPA), ceftazidime (CAZ), ceftazidime/avibactam (CZA), ceftazidime/clavulanic acid (CCV), cefteram (CEM), cefteram pivoxil (CPL), ceftezole (CTL), ceftibuten (CTB), ceftibuten/avibactam (CTA), ceftiofur (TIO), ceftizoxime (CZX), ceftizoxime alapivoxil (CZP), ceftobiprole (BPR), ceftobiprole medocaril (CFM1), ceftolozane/tazobactam (CZT), ceftriaxone (CRO), ceftriaxone/beta-lactamase inhibitor (CEB), cefuroxime (CXM), cefuroxime axetil (CXA), cephradine (CED), ciclacillin (CIC), clometocillin (CLM), cloxacillin (CLO), dicloxacillin (DIC), doripenem (DOR), epicillin (EPC), ertapenem (ETP), flucloxacillin (FLC), hetacillin (HET), imipenem (IPM), imipenem/EDTA (IPE), imipenem/relebactam (IMR), latamoxef (LTM), lenampicillin (LEN), loracarbef (LOR), mecillinam (MEC), meropenem (MEM), meropenem/nacubactam (MNC), meropenem/vaborbactam (MEV), metampicillin (MTM), meticillin (MET), mezlocillin (MEZ), mezlocillin/sulbactam (MSU), nafcillin (NAF), oxacillin (OXA), oxacillin screening test (OXA-S), panipenem (PAN), penamecillin (PNM), penicillin/novobiocin (PNO), penicillin/sulbactam (PSU), pheneticillin (PHE), phenoxymethylpenicillin (PHN), piperacillin (PIP), piperacillin/sulbactam (PIS), piperacillin/tazobactam (TZP), piridicillin (PRC), pivampicillin (PVM), pivmecillinam (PME), procaine benzylpenicillin (PRB), propicillin (PRP), razupenem (RZM), ritipenem (RIT), ritipenem acoxil (RIA), sarmoxicillin (SRX), sulbenicillin (SBC), sultamicillin (SLT6), talampicillin (TAL), taniborbactam (TAN), tebipenem (TBP), temocillin (TEM), ticarcillin (TIC), ticarcillin/clavulanic acid (TCC), and tigemonam (TMN)

- `betalactams_with_inhibitor()` can select:
  amoxicillin/clavulanic acid (AMC), amoxicillin/sulbactam (AXS), ampicillin/sulbactam (SAM), aztreonam/avibactam (AZA), aztreonam/nacubactam (ANC), cefepime/amikacin (CFA), cefepime/clavulanic acid (CPC), cefepime/enmetazobactam (FPE), cefepime/nacubactam (FNC), cefepime/taniborbactam (FTA), cefepime/tazobactam (FPT), cefepime/zidebactam (FPZ), cefoperazone/sulbactam (CSL), cefotaxime/clavulanic acid (CTC), cefotaxime/sulbactam (CTS), cefpodoxime/clavulanic acid (CDC), ceftaroline/avibactam (CPA), ceftazidime/avibactam (CZA), ceftazidime/clavulanic acid (CCV), ceftibuten/avibactam (CTA), ceftolozane/tazobactam (CZT), ceftriaxone/beta-lactamase inhibitor (CEB), imipenem/relebactam (IMR), meropenem/nacubactam (MNC), meropenem/vaborbactam (MEV), mezlocillin/sulbactam (MSU), penicillin/novobiocin (PNO), penicillin/sulbactam (PSU), piperacillin/sulbactam (PIS), piperacillin/tazobactam (TZP), and ticarcillin/clavulanic acid (TCC)
- `carbapenems()` can select:
  biapenem (BIA), doripenem (DOR), ertapenem (ETP), imipenem (IPM), imipenem/EDTA (IPE), imipenem/relebactam (IMR), meropenem (MEM), meropenem/nacubactam (MNC), meropenem/vaborbactam (MEV), panipenem (PAN), razupenem (RZM), ritipenem (RIT), ritipenem acoxil (RIA), taniborbactam (TAN), and tebipenem (TBP)
- `cephalosporins()` can select:
  cefacetrile (CAC), cefaclor (CEC), cefadroxil (CFR), cefalexin (LEX), cefaloridine (RID), cefalotin (CEP), cefamandole (MAN), cefapirin (HAP), cefatrizine (CTZ), cefazedone (CZD), cefazolin (CZO), cefcapene (CCP), cefcapene pivoxil (CCX), cefdinir (CDR), cefditoren (DIT), cefditoren pivoxil (DIX), cefepime (FEP), cefepime/amikacin (CFA), cefepime/clavulanic acid (CPC), cefepime/enmetazobactam (FPE), cefepime/nacubactam (FNC), cefepime/taniborbactam (FTA), cefepime/tazobactam (FPT), cefepime/zidebactam (FPZ), cefetamet (CAT), cefetamet pivoxil (CPI), cefetecol (CCL), cefetrizole (CZL), cefiderocol (FDC), cefixime (CFM), cefmenoxime (CMX), cefmetazole (CMZ), cefodizime (DIZ), cefonicid (CID), cefoperazone

(CFP), cefoperazone/sulbactam (CSL), ceforanide (CND), cefoselis (CSE), cefotaxime (CTX), cefotaxime screening test (CTX-S), cefotaxime/clavulanic acid (CTC), cefotaxime/sulbactam (CTS), cefotetan (CTT), cefotiam (CTF), cefotiam hexetil (CHE), cefovecin (FOV), cefoxitin (FOX), cefoxitin screening test (FOX-S), cefozopran (ZOP), cefpimizole (CFZ), cefpiramide (CPM), cefpirome (CPO), cefpodoxime (CPD), cefpodoxime proxetil (CPX), cefpodoxime/clavulanic acid (CDC), cefprozil (CPR), cefquinome (CEQ), cefroxadine (CRD), cefsulodin (CFS), cefsumide (CSU), ceftaroline (CPT), ceftaroline/avibactam (CPA), ceftazidime (CAZ), ceftazidime/avibactam (CZA), ceftazidime/clavulanic acid (CCV), cefteram (CEM), cefteram pivoxil (CPL), ceftezole (CTL), ceftibuten (CTB), ceftibuten/avibactam (CTA), ceftiofur (TIO), ceftizoxime (CZX), ceftizoxime alapivoxil (CZP), ceftobiprole (BPR), ceftobiprole medocaril (CFM1), ceftolozane/tazobactam (CZT), ceftriaxone (CRO), ceftriaxone/beta-lactamase inhibitor (CEB), cefuroxime (CXM), cefuroxime axetil (CXA), cephradine (CED), latamoxef (LTM), and loracarbef (LOR)

- `cephalosporins_1st()` can select:
  cefacetrile (CAC), cefadroxil (CFR), cefalexin (LEX), cefaloridine (RID), cefalotin (CEP), cefapirin (HAP), cefatrizine (CTZ), cefazedone (CZD), cefazolin (CZO), cefroxadine (CRD), ceftezole (CTL), and cephradine (CED)
- `cephalosporins_2nd()` can select:
  cefaclor (CEC), cefamandole (MAN), cefmetazole (CMZ), cefonicid (CID), ceforanide (CND), cefotetan (CTT), cefotiam (CTF), cefoxitin (FOX), cefoxitin screening test (FOX-S), cefprozil (CPR), cefuroxime (CXM), cefuroxime axetil (CXA), and loracarbef (LOR)
- `cephalosporins_3rd()` can select:
  cefcapene (CCP), cefcapene pivoxil (CCX), cefdinir (CDR), cefditoren (DIT), cefditoren pivoxil (DIX), cefetamet (CAT), cefetamet pivoxil (CPI), cefixime (CFM), cefmenoxime (CMX), cefodizime (DIZ), cefoperazone (CFP), cefoperazone/sulbactam (CSL), cefotaxime (CTX), cefotaxime screening test (CTX-S), cefotaxime/clavulanic acid (CTC), cefotaxime/sulbactam (CTS), cefotiam hexetil (CHE), cefovecin (FOV), cefpimizole (CFZ), cefpiramide (CPM), cefpodoxime (CPD), cefpodoxime proxetil (CPX), cefpodoxime/clavulanic acid (CDC), cefsulodin (CFS), ceftazidime (CAZ), ceftazidime/avibactam (CZA), ceftazidime/clavulanic acid (CCV), cefteram (CEM), cefteram pivoxil (CPL), ceftibuten (CTB), ceftibuten/avibactam (CTA), ceftiofur (TIO), ceftizoxime (CZX), ceftizoxime alapivoxil (CZP), ceftriaxone (CRO), ceftriaxone/beta-lactamase inhibitor (CEB), and latamoxef (LTM)
- `cephalosporins_4th()` can select:
  cefepime (FEP), cefepime/amikacin (CFA), cefepime/clavulanic acid (CPC), cefepime/enmetazobactam (FPE), cefepime/nacubactam (FNC), cefepime/taniborbactam (FTA), cefepime/tazobactam (FPT), cefepime/zidebactam (FPZ), cefetecol (CCL), cefoselis (CSE), cefozopran (ZOP), cefpirome (CPO), and cefquinome (CEQ)
- `cephalosporins_5th()` can select:
  ceftaroline (CPT), ceftaroline/avibactam (CPA), ceftobiprole (BPR), ceftobiprole medocaril (CFM1), and ceftolozane/tazobactam (CZT)
- `fluoroquinolones()` can select:
  besifloxacin (BES), ciprofloxacin (CIP), ciprofloxacin/metronidazole (CIM), ciprofloxacin/ornidazole (CIO), ciprofloxacin/tinidazole (CIT), clinafloxacin (CLX), danofloxacin (DAN), delafloxacin (DFX), difloxacin (DIF), enoxacin (ENX), enrofloxacin (ENR), finafloxacin (FIN), fleroxacin (FLE), garenoxacin (GRN), gatifloxacin (GAT), gemifloxacin (GEM), grepafloxacin (GRX), lascufloxacin (LSC), levofloxacin (LVX), levofloxacin/ornidazole (LEO), levonadifloxacin (LND), lomefloxacin (LOM), marbofloxacin (MAR), metioxate (MXT), miloxacin (MIL), moxifloxacin

(MFX), nadifloxacin (NAD), nemonoxacin (NEM), nifuroquine (NIF), nitroxoline (NTR), norfloxacin (NOR), norfloxacin screening test (NOR-S), norfloxacin/metronidazole (NME), norfloxacin/tinidazole (NTI), ofloxacin (OFX), ofloxacin/ornidazole (OOR), orbifloxacin (ORB), pazufloxacin (PAZ), pefloxacin (PEF), pefloxacin screening test (PEF-S), pradofloxacin (PRA), premafloxacin (PRX), prulifloxacin (PRU), rufloxacin (RFL), sarafloxacin (SAR), sitafloxacin (SIT), sparfloxacin (SPX), temafloxacin (TMX), tilbroquinol (TBQ), tioxacin (TXC), tosufloxacin (TFX), and trovafloxacin (TVA)

- `glycopeptides()` can select:
  avoparcin (AVO), bleomycin (BLM), dalbavancin (DAL), norvancomycin (NVA), oritavancin (ORI), ramoplanin (RAM), teicoplanin (TEC), teicoplanin-macromethod (TCM), telavancin (TLV), vancomycin (VAN), vancomycin-macromethod (VAM), and zorbamycin (ZOR)
- `ionophores()` can select:
  lasalocid (LAS), monensin sodium (MON), narasin (NAR), nystatin (NYS), and salinomycin (SAL)
- `isoxazolylpenicillins()` can select:
  cloxacillin (CLO), dicloxacillin (DIC), flucloxacillin (FLC), meticillin (MET), oxacillin (OXA), and oxacillin screening test (OXA-S)
- `lincosamides()` can select:
  clindamycin (CLI), clindamycin inducible screening test (CLI-S), lincomycin (LIN), and pirlimycin (PRL)
- `lipoglycopeptides()` can select:
  dalbavancin (DAL), oritavancin (ORI), and telavancin (TLV)
- `macrolides()` can select:
  acetylmidecamycin (ACM), acetylspiramycin (ASP), azithromycin (AZM), clarithromycin (CLR), clindamycin inducible screening test (CLI-S), dirithromycin (DIR), erythromycin (ERY), flurithromycin (FLR1), gamithromycin (GAM), josamycin (JOS), kitasamycin (KIT), meleumycin (MEL), midecamycin (MID), miocamycin (MCM), nafithromycin (ZWK), oleandomycin (OLE), pirlimycin (PRL), primycin (PRM), rokitamycin (ROK), roxithromycin (RXT), solithromycin (SOL), spiramycin (SPI), telithromycin (TLT), tildipirosin (TIP), tilmicosin (TIL), troleandomycin (TRL), tulathromycin (TUL), tylosin (TYL), and tylvalosin (TYL1)
- `monobactams()` can select:
  aztreonam (ATM), aztreonam/avibactam (AZA), aztreonam/nacubactam (ANC), carumonam (CAR), and tigemonam (TMN)
- `nitrofurans()` can select:
  furazidin (FUR), furazolidone (FRZ), nifurtoinol (NFR), nitrofurantoin (NIT), and nitrofurazone (NIZ)
- `oxazolidinones()` can select:
  cadazolid (CDZ), cycloserine (CYC), linezolid (LNZ), tedizolid (TZD), and thiacetazone (THA)
- `penicillins()` can select:
  amoxicillin (AMX), amoxicillin/clavulanic acid (AMC), amoxicillin/sulbactam (AXS), ampicillin (AMP), ampicillin/sulbactam (SAM), apalcillin (APL), aspoxicillin (APX), azidocillin (AZD), azlocillin (AZL), bacampicillin (BAM), benzathine benzylpenicillin (BNB), benzathine phenoxymethylpenicillin (BNP), benzylpenicillin (PEN), benzylpenicillin screening test (PEN-S), carbenicillin (CRB), carindacillin (CRN), ciclacillin (CIC), clometocillin (CLM), cloxacillin (CLO), dicloxacillin (DIC), epicillin (EPC), flucloxacillin (FLC), hetacillin (HET),

lenampicillin (LEN), mecillinam (MEC), metampicillin (MTM), meticillin (MET), mezlocillin (MEZ), mezlocillin/sulbactam (MSU), nafcillin (NAF), oxacillin (OXA), oxacillin screening test (OXA-S), penamecillin (PNM), penicillin/novobiocin (PNO), penicillin/sulbactam (PSU), pheneticillin (PHE), phenoxymethylpenicillin (PHN), piperacillin (PIP), piperacillin/sulbactam (PIS), piperacillin/tazobactam (TZP), piridicillin (PRC), pivampicillin (PVM), pivmecillinam (PME), procaine benzylpenicillin (PRB), propicillin (PRP), sarmoxicillin (SRX), sulbenicillin (SBC), sultamicillin (SLT6), talampicillin (TAL), temocillin (TEM), ticarcillin (TIC), and ticarcillin/clavulanic acid (TCC)

- `peptides()` can select:
  avoparcin (AVO), bacitracin (BAC), bleomycin (BLM), dalbavancin (DAL), daptomycin (DAP), norvancomycin (NVA), oritavancin (ORI), ramoplanin (RAM), teicoplanin (TEC), teicoplanin-macromethod (TCM), telavancin (TLV), thiostrepton (THS), vancomycin (VAN), vancomycin-macromethod (VAM), and zorbamycin (ZOR)
- `phenicols()` can select:
  chloramphenicol (CHL), florfenicol (FLR), and thiamphenicol (THI)
- `phosphonics()` can select:
  amikacin/fosfomycin (AKF) and fosfomycin (FOS)
- `polymyxins()` can select:
  colistin (COL), polymyxin B (PLB), and polymyxin B/polysorbate 80 (POP)
- `quinolones()` can select:
  besifloxacin (BES), cinoxacin (CIN), ciprofloxacin (CIP), ciprofloxacin/metronidazole (CIM), ciprofloxacin/ornidazole (CIO), ciprofloxacin/tinidazole (CIT), clinafloxacin (CLX), danofloxacin (DAN), delafloxacin (DFX), difloxacin (DIF), enoxacin (ENX), enrofloxacin (ENR), finafloxacin (FIN), fleroxacin (FLE), flumequine (FLM), garenoxacin (GRN), gatifloxacin (GAT), gemifloxacin (GEM), grepafloxacin (GRX), lascufloxacin (LSC), levofloxacin (LVX), levofloxacin/ornidazole (LEO), levonadifloxacin (LND), lomefloxacin (LOM), marbofloxacin (MAR), metioxate (MXT), miloxacin (MIL), moxifloxacin (MFX), nadifloxacin (NAD), nalidixic acid (NAL), nalidixic acid screening test (NAL-S), nemonoxacin (NEM), nifuroquine (NIF), nitroxoline (NTR), norfloxacin (NOR), norfloxacin screening test (NOR-S), norfloxacin/metronidazole (NME), norfloxacin/tinidazole (NTI), ofloxacin (OFX), ofloxacin/ornidazole (OOR), orbifloxacin (ORB), oxolinic acid (OXO), ozenoxacin (OZN), pazufloxacin (PAZ), pefloxacin (PEF), pefloxacin screening test (PEF-S), pipemidic acid (PPA), piromidic acid (PIR), pradofloxacin (PRA), premafloxacin (PRX), prulifloxacin (PRU), rosoxacin (ROS), rufloxacin (RFL), sarafloxacin (SAR), sitafloxacin (SIT), sparfloxacin (SPX), temafloxacin (TMX), tilbroquinol (TBQ), tioxacin (TXC), tosufloxacin (TFX), and trovafloxacin (TVA)
- `rifamycins()` can select:
  rifabutin (RIB), rifampicin (RIF), rifampicin/ethambutol/isoniazid (REI), rifampicin/isoniazid (RFI), rifampicin/pyrazinamide/ethambutol/isoniazid (RPEI), rifampicin/pyrazinamide/isoniazid (RPI), rifamycin (RFM), and rifapentine (RFP)
- `spiropyrimidinetriones()` can select:
  zoliflodacin (ZFD)
- `streptogramins()` can select:
  ostreogrycin (OST), pristinamycin (PRI), quinupristin/dalfopristin (QDA), and virginiamycine (VIR)
- `sulfonamides()` can select:
  isoniazid/sulfamethoxazole/trimethoprim/pyridoxine (IST), ormetroprim/sulfamethoxazole (ORS),

sulfachlorpyridazine (SUP), sulfadiazine (SDI), sulfadiazine/tetroxoprim (SLT), sulfadiazine/trimethoprim (SLT1), sulfadimethoxine (SUD), sulfadimidine (SDM), sulfadimidine/trimethoprim (SLT2), sulfafurazole (SLF), sulfaisodimidine (SLF1), sulfalene (SLF2), sulfamazone (SZO), sulfamerazine (SLF3), sulfamerazine/trimethoprim (SLT3), sulfamethazine (SUM), sulfamethizole (SLF4), sulfamethoxazole (SMX), sulfamethoxypyridazine (SLF5), sulfametomidine (SLF6), sulfametoxydiazine (SLF7), sulfametrole/trimethoprim (SLT4), sulfamoxole (SLF8), sulfamoxole/trimethoprim (SLT5), sulfanilamide (SLF9), sulfaperin (SLF10), sulfaphenazole (SLF11), sulfapyridine (SLF12), sulfasuccinamide (SNA), sulfathiazole (SUT), sulfathiourea (SLF13), sulfisoxazole (SOX), sulfonamide (SSS), and trimethoprim/sulfamethoxazole (SXT)

- `tetracyclines()` can select:
  cetocycline (CTO), chlortetracycline (CTE), clomocycline (CLM1), demeclocycline (DEM), doxycycline (DOX), eravacycline (ERV), lymecycline (LYM), metacycline (MTC), minocycline (MNO), omadacycline (OMC), oxytetracycline (OXY), penimepicycline (PNM1), rolitetracycline (RLT), sarecycline (SRC), tetracycline (TCY), tetracycline screening test (TCY-S), tetracycline/oleandomycin (TOL), and tigecycline (TGC)
- `trimethoprims()` can select:
  brodimoprim (BDP), iclaprim (ICL), isoniazid/sulfamethoxazole/trimethoprim/pyridoxine (IST), ormetroprim/sulfamethoxazole (ORS), sulfadiazine (SDI), sulfadiazine/tetroxoprim (SLT), sulfadiazine/trimethoprim (SLT1), sulfadimethoxine (SUD), sulfadimidine (SDM), sulfadimidine/trimethoprim (SLT2), sulfafurazole (SLF), sulfaisodimidine (SLF1), sulfalene (SLF2), sulfamazone (SZO), sulfamerazine (SLF3), sulfamerazine/trimethoprim (SLT3), sulfamethizole (SLF4), sulfamethoxazole (SMX), sulfamethoxypyridazine (SLF5), sulfametomidine (SLF6), sulfametoxydiazine (SLF7), sulfametrole/trimethoprim (SLT4), sulfamoxole (SLF8), sulfamoxole/trimethoprim (SLT5), sulfanilamide (SLF9), sulfaperin (SLF10), sulfaphenazole (SLF11), sulfapyridine (SLF12), sulfathiazole (SUT), sulfathiourea (SLF13), trimethoprim (TMP), and trimethoprim/sulfamethoxazole (SXT)
- `ureidopenicillins()` can select:
  azlocillin (AZL), mezlocillin (MEZ), piperacillin (PIP), and piperacillin/tazobactam (TZP)

## Download Our Reference Data

All reference data sets in the AMR package - including information on microorganisms, antimicrobials, and clinical breakpoints - are freely available for download in multiple formats: R, MS Excel, Apache Feather, Apache Parquet, SPSS, and Stata.

For maximum compatibility, we also provide machine-readable, tab-separated plain text files suitable for use in any software, including laboratory information systems.

Visit our website for direct download links, or explore the actual files in our GitHub repository.

## Examples

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


# you can use the selectors separately to retrieve all possible antimicrobials:
carbapenems()
```

```
# Though they are primarily intended to use for selections and filters.
# Examples sections below are split into 'dplyr', 'base R', and 'data.table':

## Not run:
# dplyr ------------------------------------------------------------------

library(dplyr, warn.conflicts = FALSE)

example_isolates %>% select(carbapenems())

# select columns 'mo', 'AMK', 'GEN', 'KAN' and 'TOB'
example_isolates %>% select(mo, aminoglycosides())

# you can combine selectors like you are used with tidyverse
# e.g., for betalactams, but not the ones with an enzyme inhibitor:
example_isolates %>% select(betalactams(), -betalactams_with_inhibitor())

# select only antimicrobials with DDDs for oral treatment
example_isolates %>% select(administrable_per_os())

# get AMR for all aminoglycosides e.g., per ward:
example_isolates %>%
  group_by(ward) %>%
  summarise(across(
    aminoglycosides(),
    resistance
  ))

# You can combine selectors with '&' to be more specific:
example_isolates %>%
  select(penicillins() & administrable_per_os())

# get AMR for only drugs that matter - no intrinsic resistance:
example_isolates %>%
  filter(mo_genus() %in% c("Escherichia", "Klebsiella")) %>%
  group_by(ward) %>%
  summarise_at(
    not_intrinsic_resistant(),
    resistance
  )

# get susceptibility for antimicrobials whose name contains "trim":
example_isolates %>%
  filter(first_isolate()) %>%
  group_by(ward) %>%
  summarise(across(amr_selector(name %like% "trim"), susceptibility))

# this will select columns 'IPM' (imipenem) and 'MEM' (meropenem):
example_isolates %>%
  select(carbapenems())

# this will select columns 'mo', 'AMK', 'GEN', 'KAN' and 'TOB':
```

```
example_isolates %>%
  select(mo, aminoglycosides())

# any() and all() work in dplyr's filter() too:
example_isolates %>%
  filter(
    any(aminoglycosides() == "R"),
    all(cephalosporins_2nd() == "R")
  )

# also works with c():
example_isolates %>%
  filter(any(c(carbapenems(), aminoglycosides()) == "R"))

# not setting any/all will automatically apply all():
example_isolates %>%
  filter(aminoglycosides() == "R")

# this will select columns 'mo' and all antimycobacterial drugs ('RIF'):
example_isolates %>%
  select(mo, amr_class("mycobact"))

# get bug/drug combinations for only glycopeptides in Gram-positives:
example_isolates %>%
  filter(mo_is_gram_positive()) %>%
  select(mo, glycopeptides()) %>%
  bug_drug_combinations() %>%
  format()

data.frame(
  some_column = "some_value",
  J01CA01 = "S"
) %>% # ATC code of ampicillin
  select(penicillins()) # only the 'J01CA01' column will be selected

# with recent versions of dplyr, this is all equal:
x <- example_isolates[carbapenems() == "R", ]
y <- example_isolates %>% filter(carbapenems() == "R")
z <- example_isolates %>% filter(if_all(carbapenems(), ~ .x == "R"))
identical(x, y) && identical(y, z)

## End(Not run)


# base R ------------------------------------------------------------------

# select columns 'IPM' (imipenem) and 'MEM' (meropenem)
example_isolates[, carbapenems()]

# select columns 'mo', 'AMK', 'GEN', 'KAN' and 'TOB'
example_isolates[, c("mo", aminoglycosides())]

# select only antimicrobials with DDDs for oral treatment
```

```
example_isolates[, administrable_per_os()]

# filter using any() or all()
example_isolates[any(carbapenems() == "R"), ]
subset(example_isolates, any(carbapenems() == "R"))

# filter on any or all results in the carbapenem columns (i.e., IPM, MEM):
example_isolates[any(carbapenems()), ]
example_isolates[all(carbapenems()), ]

# filter with multiple antimicrobial selectors using c()
example_isolates[all(c(carbapenems(), aminoglycosides()) == "R"), ]

# filter + select in one go: get penicillins in carbapenem-resistant strains
example_isolates[any(carbapenems() == "R"), penicillins()]

# You can combine selectors with '&' to be more specific. For example,
# penicillins() would select benzylpenicillin ('peni G') and
# administrable_per_os() would select erythromycin. Yet, when combined these
# drugs are both omitted since benzylpenicillin is not administrable per os
# and erythromycin is not a penicillin:
example_isolates[, penicillins() & administrable_per_os()]

# amr_selector() applies a filter in the `antimicrobials` data set and is thus
# very flexible. For instance, to select antimicrobials with an oral DDD
# of at least 1 gram:
example_isolates[, amr_selector(oral_ddd > 1 & oral_units == "g")]


# data.table ----------------------------------------------------------------

# data.table is supported as well, just use it in the same way as with
# base R, but add `with = FALSE` if using a single AMR selector.

if (require("data.table")) {
  dt <- as.data.table(example_isolates)

  # this does not work, it returns column *names*
  dt[, carbapenems()]
}
if (require("data.table")) {
  # so `with = FALSE` is required
  dt[, carbapenems(), with = FALSE]
}

# for multiple selections or AMR selectors, `with = FALSE` is not needed:
if (require("data.table")) {
  dt[, c("mo", aminoglycosides())]
}
if (require("data.table")) {
  dt[, c(carbapenems(), aminoglycosides())]
}
```

```
# row filters are also supported:
if (require("data.table")) {
  dt[any(carbapenems() == "S"), ]
}
if (require("data.table")) {
  dt[any(carbapenems() == "S"), penicillins(), with = FALSE]
}
```

---

as.ab *Transform Input to an Antibiotic ID*

---

## Description

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

## Usage

```
as.ab(
  x,
  flag_multiple_results = TRUE,
  language = get_AMR_locale(),
  info = interactive(),
  ...
)

is.ab(x)

ab_reset_session()

NA_ab_
```

## Arguments

| | |
|---|---|
| x | A character vector to determine to antibiotic ID. |
| flag_multiple_results | A logical to indicate whether a note should be printed to the console that probably more than one antibiotic drug code or name can be retrieved from a single input value. |
| language | Language to coerce input values from any of the 28 supported languages - default to the system language if supported (see get_AMR_locale()). |
| info | A logical to indicate whether a progress bar should be printed - the default is TRUE only in interactive mode. |
| ... | Arguments passed on to internal functions. |

## Details

All entries in the antimicrobials 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. Not that some drugs contain multiple ATC codes.

All these properties will be searched for the user input. The as.ab() can correct for different forms of misspelling:

- Wrong spelling of drug names (such as "tobramicin" or "gentamycin"), which corrects for most audible similarities such as f/ph, x/ks, c/z/s, t/th, etc.
- Too few or too many vowels or consonants
- Switching two characters (such as "mreopenem", often the case in clinical data, when doctors typed too fast)
- Digitalised paper records, leaving artefacts like 0/o/O (zero and O's), B/8, n/r, etc.

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

Results of non-exact input are remembered during the session, so that the same input is not matched again. Use ab_reset_session() to forget these results. They are also forgotten after add_custom_antimicrobials() or clear_custom_antimicrobials().

Note: the as.ab() and ab_* functions may use very long regular expression to match brand names of antimicrobial drugs. This may fail on some systems.

You can add your own manual codes to be considered by as.ab() and all ab_* functions, see add_custom_antimicrobials().

NA_ab_ is a missing value of the new ab class, analogous to e.g. base R's NA_character_.

## Value

A character vector with additional class ab

## Source

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

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

## WHOCC

This package contains **all ~550 antibiotic, antimycotic and antiviral drugs** and their Anatomical Therapeutic Chemical (ATC) codes, ATC groups and Defined Daily Dose (DDD) from the World Health Organization Collaborating Centre for Drug Statistics Methodology (WHOCC, https://atcddd.fhi.no) and the Pharmaceuticals Community Register of the European Commission (https://ec.europa.eu/health/documents/community-register/html/index_en.htm).

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

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

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

### Download Our Reference Data

All reference data sets in the AMR package - including information on microorganisms, antimicrobials, and clinical breakpoints - are freely available for download in multiple formats: R, MS Excel, Apache Feather, Apache Parquet, SPSS, and Stata.

For maximum compatibility, we also provide machine-readable, tab-separated plain text files suitable for use in any software, including laboratory information systems.

Visit our website for direct download links, or explore the actual files in our GitHub repository.

### See Also

- antimicrobials for the data.frame that is being used to determine ATCs
- `ab_from_text()` for a function to retrieve antimicrobial drugs from clinical text (from health care records)

### Examples

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

# spelling from different languages and dyslexia are no problem
ab_atc("ceftriaxon")
ab_atc("cephtriaxone") # small spelling error
ab_atc("cephthriaxone") # 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")
ab_name("eryt")


if (require("dplyr")) {
  # you can quickly rename 'sir' columns using set_ab_names() with dplyr:
  example_isolates %>%
    set_ab_names(where(is.sir), property = "atc")
}
```

`as.av` *Transform Input to an Antiviral Drug ID*

## Description

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

## Usage

```
as.av(x, flag_multiple_results = TRUE, info = interactive(), ...)

is.av(x)
```

## Arguments

| | |
|---|---|
| `x` | A character vector to determine to antiviral drug ID. |
| `flag_multiple_results` | A logical to indicate whether a note should be printed to the console that probably more than one antiviral drug code or name can be retrieved from a single input value. |
| `info` | A logical to indicate whether a progress bar should be printed - the default is `TRUE` only in interactive mode. |
| `...` | Arguments passed on to internal functions. |

## Details

All entries in the antivirals 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. Not that some drugs contain multiple ATC codes.

All these properties will be searched for the user input. The `as.av()` can correct for different forms of misspelling:

- Wrong spelling of drug names (such as "acyclovir"), which corrects for most audible similarities such as f/ph, x/ks, c/z/s, t/th, etc.
- Too few or too many vowels or consonants
- Switching two characters (such as "aycclovir", often the case in clinical data, when doctors typed too fast)
- Digitalised paper records, leaving artefacts like 0/o/O (zero and O's), B/8, n/r, etc.

Use the `av_*` functions to get properties based on the returned antiviral drug ID, see *Examples*.

Note: the `as.av()` and `av_*` functions may use very long regular expression to match brand names of antimicrobial drugs. This may fail on some systems.

## Value

A character vector with additional class ab

## Source

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

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

## WHOCC

This package contains **all ~550 antibiotic, antimycotic and antiviral drugs** and their Anatomical Therapeutic Chemical (ATC) codes, ATC groups and Defined Daily Dose (DDD) from the World Health Organization Collaborating Centre for Drug Statistics Methodology (WHOCC, https://atcddd.fhi.no) and the Pharmaceuticals Community Register of the European Commission (https://ec.europa.eu/health/documents/community-register/html/index_en.htm).

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

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

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

## Download Our Reference Data

All reference data sets in the AMR package - including information on microorganisms, antimicrobials, and clinical breakpoints - are freely available for download in multiple formats: R, MS Excel, Apache Feather, Apache Parquet, SPSS, and Stata.

For maximum compatibility, we also provide machine-readable, tab-separated plain text files suitable for use in any software, including laboratory information systems.

Visit our website for direct download links, or explore the actual files in our GitHub repository.

## See Also

- antivirals for the data.frame that is being used to determine ATCs
- av_from_text() for a function to retrieve antimicrobial drugs from clinical text (from health care records)

## Examples

```
# these examples all return "ACI", the ID of aciclovir:
as.av("J05AB01")
as.av("J 05 AB 01")
as.av("Aciclovir")
as.av("aciclo")
as.av("   aciclo 123")
```

```
as.av("ACICL")
as.av("ACI")
as.av("Virorax") # trade name
as.av("Zovirax") # trade name

as.av("acyklofir") # severe spelling error, yet works

# use av_* functions to get a specific properties (see ?av_property);
# they use as.av() internally:
av_name("J05AB01")
av_name("acicl")
```

---

`as.disk` *Transform Input to Disk Diffusion Diameters*

---

## Description

This transforms a vector to a new class `disk`, which is a disk diffusion growth zone size (around an antibiotic disk) in millimetres between 0 and 50.

## Usage

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

NA_disk_

is.disk(x)
```

## Arguments

| | |
|---|---|
| `x` | Vector. |
| `na.rm` | A logical indicating whether missing values should be removed. |

## Details

Interpret disk values as SIR values with `as.sir()`. It supports guidelines from EUCAST and CLSI.

Disk diffusion growth zone sizes must be between 0 and 50 millimetres. Decimal values are rounded up. All input values outside the 0-50 range, including values higher than 50, return `NA` with a warning.

`NA_disk_` is a missing value of the new `disk` class.

## Value

An integer with additional class `disk`

## See Also

`as.sir()`

## Examples

```
# transform existing disk zones to the `disk` class (using base R)
df <- data.frame(
  microorganism = "Escherichia coli",
  AMP = 20,
  CIP = 14,
  GEN = 18,
  TOB = 16
)
df[, 2:5] <- lapply(df[, 2:5], as.disk)
str(df)


# transforming is easier with dplyr:
if (require("dplyr")) {
  df %>% mutate(across(AMP:TOB, as.disk))
}


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

# interpret whole data set, pretend to be all from urinary tract infections:
as.sir(df, uti = TRUE)
```

---

# as.mic *Transform Input to Minimum Inhibitory Concentrations (MIC)*

---

## Description

This transforms vectors to a new class `mic`, which treats the input as decimal numbers, while maintaining operators (such as ">=") and only allowing valid MIC values known to the field of (medical) microbiology.

## Usage

```
as.mic(x, na.rm = FALSE, keep_operators = "all", round_to_next_log2 = FALSE)

is.mic(x)

NA_mic_

rescale_mic(
  x,
```

```
  mic_range,
  keep_operators = "edges",
  as.mic = TRUE,
  round_to_next_log2 = FALSE
)

mic_p50(x, na.rm = FALSE, ...)

mic_p90(x, na.rm = FALSE, ...)

## S3 method for class 'mic'
droplevels(x, as.mic = FALSE, ...)
```

## Arguments

| | |
|---|---|
| x | A character or numeric vector. |
| na.rm | A logical indicating whether missing values should be removed. |
| keep_operators | A character specifying how to handle operators (such as > and <=) in the input. Accepts one of three values: "all" (or TRUE) to keep all operators, "none" (or FALSE) to remove all operators, or "edges" to keep operators only at both ends of the range. |
| round_to_next_log2 | A logical to round up all values to the next log2 level, that are not either 0.0001, 0.0002, 0.0005, 0.001, 0.002, 0.004, 0.008, 0.016, 0.032, 0.064, 0.125, 0.25, 0.5, 1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048, or 4096. Values that are already in this list (with or without operators), are left unchanged (including any operators). |
| mic_range | A manual range to rescale the MIC values, e.g., mic_range = c(0.001, 32). Use NA to prevent rescaling on one side, e.g., mic_range = c(NA, 32). |
| as.mic | A logical to indicate whether the mic class should be kept - the default is TRUE for rescale_mic() and FALSE for droplevels(). When setting this to FALSE in rescale_mic(), the output will have factor levels that acknowledge mic_range. |
| ... | Arguments passed on to methods. |

## Details

To interpret MIC values as SIR values, use as.sir() on MIC values. It supports guidelines from EUCAST (2011-2026) and CLSI (2011-2026).

This class for MIC values is a quite a special data type: formally it is an ordered factor with valid MIC values as factor levels (to make sure only valid MIC values are retained), but for any mathematical operation it acts as decimal numbers:

```
x <- random_mic(10)
x
#> Class <mic>
#>  [1] 16     1      8      8      64     >=128  0.0625 32     32     16
```

```
is.factor(x)
#> [1] TRUE

x[1] * 2
#> [1] 32

median(x)
#> [1] 26
```

This makes it possible to maintain operators that often come with MIC values, such ">=" and "<=", even when filtering using numeric values in data analysis, e.g.:

```
x[x > 4]
#> Class <mic>
#> [1] 16    8     8     64    >=128 32    32    16

df <- data.frame(x, hospital = "A")
subset(df, x > 4) # or with dplyr: df %>% filter(x > 4)
#>        x hospital
#> 1     16        A
#> 5     64        A
#> 6  >=128        A
#> 8     32        A
#> 9     32        A
#> 10    16        A
```

All so-called group generic functions are implemented for the MIC class (such as `!`, `!=`, `<`, `>=`, `exp()`, `log2()`). Some mathematical functions are also implemented (such as `quantile()`, `median()`, `fivenum()`). Since `sd()` and `var()` are non-generic functions, these could not be extended. Use `mad()` as an alternative, or use e.g. `sd(as.numeric(x))` where `x` is your vector of MIC values.

Using `as.double()` or `as.numeric()` on MIC values will remove the operators and return a numeric vector. Do **not** use `as.integer()` on MIC values as by the R convention on factors, it will return the index of the factor levels (which is often useless for regular users).

The function `is.mic()` detects if the input contains class `mic`. If the input is a data.frame or list, it iterates over all columns/items and returns a logical vector.

Use `droplevels()` to drop unused levels. At default, it will return a plain factor. Use `droplevels(..., as.mic = TRUE)` to maintain the `mic` class.

With `rescale_mic()`, existing MIC ranges can be limited to a defined range of MIC values. This can be useful to better compare MIC distributions.

For `ggplot2`, use one of the `scale_*_mic()` functions to plot MIC values. They allows custom MIC ranges and to plot intermediate log2 levels for missing MIC values.

`NA_mic_` is a missing value of the new `mic` class, analogous to e.g. base R's `NA_character_`.

Use `mic_p50()` and `mic_p90()` to get the 50th and 90th percentile of MIC values. They return 'normal' numeric values.

## Value

Ordered factor with additional class mic, that in mathematical operations acts as a numeric vector. Bear in mind that the outcome of any mathematical operation on MICs will return a numeric value.

## See Also

as.sir()

## Examples

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

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

# mathematical processing treats MICs as, and returns, numeric values
fivenum(mic_data)
quantile(mic_data)
all(mic_data < 512)

# rescale MICs using rescale_mic()
rescale_mic(mic_data, mic_range = c(4, 16))

# round up to nearest log2 level, e.g. for CLSI breakpoint interpretation:
c(1:8)
as.mic(c(1:8), round_to_next_log2 = TRUE)

# interpret MIC values
as.sir(
  x = as.mic(2),
  mo = as.mo("Streptococcus pneumoniae"),
  ab = "AMX",
  guideline = "EUCAST"
)
as.sir(
  x = as.mic(c(0.01, 2, 4, 8)),
  mo = as.mo("Streptococcus pneumoniae"),
  ab = "AMX",
  guideline = "EUCAST"
)

# plot MIC values, see ?plot
plot(mic_data)
plot(mic_data, mo = "E. coli", ab = "cipro")

if (require("ggplot2")) {
  autoplot(mic_data, mo = "E. coli", ab = "cipro")
}
if (require("ggplot2")) {
  autoplot(mic_data, mo = "E. coli", ab = "cipro", language = "nl") # Dutch
```

```
}
```

---

`as.mo` *Transform Arbitrary Input to Valid Microbial Taxonomy*

---

## Description

Use this function to get a valid microorganism code (`mo`) based on arbitrary user input. Determination is done using intelligent rules and the complete taxonomic tree of the domains Animalia, Archaea, Bacteria, Chromista, Plantae, and Protozoa, and most microbial species from the domain Fungi (see *Source*). The input can be almost anything: a full name (like `"Staphylococcus aureus"`), an abbreviated name (such as `"S. aureus"`), an abbreviation known in the field (such as `"MRSA"`), or just a genus. See *Examples*.

## Usage

```
as.mo(
  x,
  Becker = FALSE,
  Lancefield = FALSE,
  minimum_matching_score = NULL,
  keep_synonyms = getOption("AMR_keep_synonyms", FALSE),
  reference_df = get_mo_source(),
  ignore_pattern = getOption("AMR_ignore_pattern", NULL),
  cleaning_regex = getOption("AMR_cleaning_regex", mo_cleaning_regex()),
  only_fungi = getOption("AMR_only_fungi", FALSE),
  language = get_AMR_locale(),
  info = interactive(),
  ...
)

is.mo(x)

mo_uncertainties()

mo_renamed()

mo_failures()

mo_reset_session()

mo_cleaning_regex()

NA_mo_
```

## 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.* (see *Source*). Please see *Details* for a full list of staphylococcal species that will be converted.<br>This excludes *Staphylococcus aureus* at default, use `Becker = "all"` to also categorise *S. aureus* as "CoPS". |
| Lancefield | A logical to indicate whether a beta-haemolytic *Streptococcus* should be categorised into Lancefield groups instead of their own species, according to Rebecca C. Lancefield (see *Source*). 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. . Please see *Details* for a full list of streptococcal species that will be converted.<br>This excludes enterococci at default (who are in group D), use `Lancefield = "all"` to also categorise all enterococci as group D. |
| minimum_matching_score | A numeric value to set as the lower limit for the MO matching score. When left blank, this will be determined automatically based on the character length of x, its taxonomic domain and human pathogenicity. |
| keep_synonyms | A logical to indicate if outdated, previously valid taxonomic names must be preserved and not be corrected to currently accepted names. Do note that the term "synonym" is in this case jargon from the field of microbial taxonomy - it is not in place to denote that e.g. "Streptococcus Group A" is a synonym of *S. pyogenes*. Though this is practically the case, taxonomically it is not as "Streptococcus Group A" is not even a valid taxonomic name.<br>The default is `FALSE`, which will return a note if outdated taxonomic names were processed. The default can be set with the package option `AMR_keep_synonyms`, i.e. `options(AMR_keep_synonyms = TRUE)` or `options(AMR_keep_synonyms = FALSE)`. |
| reference_df | A data.frame to be used 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). |
| ignore_pattern | A Perl-compatible regular expression (case-insensitive) of which all matches in x must return `NA`. This can be convenient to exclude known non-relevant input and can also be set with the package option `AMR_ignore_pattern`, e.g. `options(AMR_ignore_pattern = "(not reported|contaminated flora)")`. |
| cleaning_regex | A Perl-compatible regular expression (case-insensitive) to clean the input of x. Every matched part in x will be removed. At default, this is the outcome of `mo_cleaning_regex()`, which removes texts between brackets and texts such as "species" and "serovar". The default can be set with the package option `AMR_cleaning_regex`. |
| only_fungi | A logical to indicate if only fungi must be found, making sure that e.g. misspellings always return records from the domain of Fungi. This can be set globally for all microorganism functions with the package option `AMR_only_fungi`, i.e. `options(AMR_only_fungi = TRUE)`. |

| | |
|---|---|
| language | Language to translate text like "no growth", which defaults to the system language (see get_AMR_locale()). |
| info | A logical to indicate that info must be printed, e.g. a progress bar when more than 25 items are to be coerced, or a list with outdated taxonomic names. The default is TRUE only in interactive mode. |
| ... | Other arguments passed on to functions. |

## Details

A microorganism (MO) code from this package (class: mo) is human-readable and typically looks like these examples:

```
Code               Full name
---------------    --------------------------------------
B_KLBSL            Klebsiella
B_KLBSL_PNMN       Klebsiella pneumoniae
B_KLBSL_PNMN_RHNS  Klebsiella pneumoniae rhinoscleromatis
|   |    |    |
|   |    |    |
|   |    |    \---> subspecies, a 3-5 letter acronym
|   |    \----> species, a 3-6 letter acronym
|   \----> genus, a 4-8 letter acronym
\----> domain: A (Archaea), AN (Animalia), B (Bacteria),
               C (Chromista), F (Fungi), PL (Plantae),
               P (Protozoa)
```

Values that cannot be coerced will be considered 'unknown' and will return the MO code UNKNOWN with a warning.

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

The as.mo() function uses a novel and scientifically validated (doi:10.18637/jss.v104.i03) matching score algorithm (see *Matching Score for Microorganisms* below) to match input against the available microbial taxonomy in this package. This implicates 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:

Results of non-exact taxonomic input are based on their matching score. The lowest allowed score can be set with the minimum_matching_score argument. At default this will be determined based on the character length of the input, the taxonomic domain, and the human pathogenicity of the taxonomic outcome. If values are matched with uncertainty, a message will be shown to suggest the user to inspect the results with mo_uncertainties(), which returns a data.frame with all specifications.

To increase the quality of matching, the cleaning_regex argument is used to clean the input. This must be a regular expression that matches parts of the input that should be removed before the input is matched against the available microbial taxonomy. It will be matched Perl-compatible and case-insensitive. The default value of cleaning_regex is the outcome of the helper function mo_cleaning_regex().

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

- Use `mo_uncertainties()` to get a data.frame that prints in a pretty format with all taxonomic names that were guessed. The output contains the matching score for all matches (see *Matching Score for Microorganisms* below).
- Use `mo_failures()` to get a character 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 outdated, previously accepted taxonomic names.

Results of non-exact input are remembered during the session, so that the same input is not matched again. This only applies to the same settings of `minimum_matching_score`, `only_fungi` and `cleaning_regex`. Use `mo_reset_session()` to forget these results. They are also forgotten when the taxonomy changes, e.g. after `add_custom_microorganisms()`.

**For Mycologists:**

The matching score algorithm gives precedence to bacteria over fungi. If you are only analysing fungi, be sure to use `only_fungi = TRUE`, or better yet, add this to your code and run it once every session:

```
options(AMR_only_fungi = TRUE)
```

This will make sure that no bacteria or other 'non-fungi' will be returned by `as.mo()`, or any of the `mo_*` functions.

**Coagulase-negative and Coagulase-positive Staphylococci:**

With `Becker = TRUE`, the following staphylococci will be converted to their corresponding coagulase group:

- Coagulase-negative: *S. americanisciuri*, *S. argensis*, *S. arlettae*, *S. auricularis*, *S. borealis*, *S. brunensis*, *S. caeli*, *S. caledonicus*, *S. canis*, *S. capitis*, *S. capitis capitis*, *S. capitis urealyticus*, *S. capitis ureolyticus*, *S. caprae*, *S. carnosus*, *S. carnosus carnosus*, *S. carnosus utilis*, *S. casei*, *S. caseorum*, *S. chromogenes*, *S. cohnii*, *S. cohnii cohnii*, *S. cohnii urealyticum*, *S. cohnii urealyticus*, *S. cohnii ureilyticus*, *S. condimenti*, *S. croceilyticus*, *S. debuckii*, *S. devriesei*, *S. dromedarii*, *S. durrellii*, *S. edaphicus*, *S. epidermidis*, *S. equorum*, *S. equorum equorum*, *S. equorum linens*, *S. felis*, *S. fleurettii*, *S. gallinarum*, *S. haemolyticus*, *S. halotolerans*, *S. hominis*, *S. hominis hominis*, *S. hominis novobiosepticus*, *S. hsinchuensis*, *S. jettensis*, *S. kloosii*, *S. lentus*, *S. lloydii*, *S. lugdunensis*, *S. marylandisciuri*, *S. massiliensis*, *S. microti*, *S. muscae*, *S. nepalensis*, *S. parequorum*, *S. pasteuri*, *S. petrasii*, *S. petrasii croceilyticus*, *S. petrasii jettensis*, *S. petrasii petrasii*, *S. petrasii pragensis*, *S. pettenkoferi*, *S. piscifermentans*, *S. pragensis*, *S. pseudoxylosus*, *S. pulvereri*, *S. ratti*, *S. rostri*, *S. saccharolyticus*, *S. saprophyticus*, *S. saprophyticus bovis*, *S. saprophyticus saprophyticus*, *S. schleiferi*, *S. schleiferi schleiferi*, *S. sciuri*, *S. sciuri carnaticus*, *S. sciuri lentus*, *S. sciuri rodentium*, *S. sciuri sciuri*, *S. shinii*, *S. simulans*, *S. stepanovicii*, *S. succinus*, *S. succinus casei*, *S. succinus succinus*, *S. taiwanensis*, *S. urealyticus*, *S. ureilyticus*, *S. vitulinus*, *S. vitulus*, *S. warneri*, *S. xeri*, and *S. xylosus*
- Coagulase-positive: *S. agnetis*, *S. argenteus*, *S. coagulans*, *S. cornubiensis*, *S. delphini*, *S. hyicus*, *S. hyicus hyicus*, *S. intermedius*, *S. lutrae*, *S. pseudintermedius*, *S. schleiferi coagulans*, *S. schweitzeri*, *S. simiae*, *S. singaporensis*, and *S. ursi*

This is based on:

- Becker K *et al.* (2014). **Coagulase-Negative Staphylococci.** *Clin Microbiol Rev.* 27(4): 870-926; doi:10.1128/CMR.0010913

- Becker K *et al.* (2019). **Implications of identifying the recently defined members of the *S. aureus* complex, *S. argenteus* and *S. schweitzeri*: A position paper of members of the ESCMID Study Group for staphylococci and Staphylococcal Diseases (ESGS).** *Clin Microbiol Infect*; doi:10.1016/j.cmi.2019.02.028
- Becker K *et al.* (2020). **Emergence of coagulase-negative staphylococci.** *Expert Rev Anti Infect Ther.* 18(4):349-366; doi:10.1080/14787210.2020.1730813

For newly named staphylococcal species, such as *S. brunensis* (2024) and *S. shinii* (2023), we looked up the scientific reference to make sure the species are considered for the correct coagulase group.

**Lancefield Groups in Streptococci:**

With `Lancefield = TRUE`, the following streptococci will be converted to their corresponding Lancefield group:

- Streptococcus Group A: *S. pyogenes*
- Streptococcus Group B: *S. agalactiae*
- Streptococcus Group C: *S. dysgalactiae*, *S. dysgalactiae dysgalactiae*, *S. dysgalactiae equisimilis*, *S. equi*, *S. equi equi*, *S. equi ruminatorum*, and *S. equi zooepidemicus*
- Streptococcus Group F: *S. anginosus*, *S. anginosus anginosus*, *S. anginosus whileyi*, *S. constellatus*, *S. constellatus constellatus*, *S. constellatus pharyngis*, *S. constellatus viborgensis*, and *S. intermedius*
- Streptococcus Group G: *S. canis*, *S. dysgalactiae*, *S. dysgalactiae dysgalactiae*, and *S. dysgalactiae equisimilis*
- Streptococcus Group H: *S. sanguinis*
- Streptococcus Group K: *S. salivarius*
- Streptococcus Group L: *S. dysgalactiae*, *S. dysgalactiae dysgalactiae*, and *S. dysgalactiae equisimilis*

This is based on:

- Lancefield RC (1933). **A serological differentiation of human and other groups of hemolytic streptococci.** *J Exp Med.* 57(4): 571-95; doi:10.1084/jem.57.4.571

`NA_mo_` is a missing value of the new `mo` class, analogous to e.g. base R's `NA_character_`.

## Value

A character vector with additional class `mo`

## Source


- Berends MS *et al.* (2022). **AMR: An R Package for Working with Antimicrobial Resistance Data**. *Journal of Statistical Software*, 104(3), 1-31; doi:10.18637/jss.v104.i03
- Freese, HM *et al.* (2026). **TYGS and LPSN in 2025: a Global Core Biodata Resource for genome-based classification and nomenclature of prokaryotes within DSMZ Digital Diversity.** Nucleic Acids Research, 54, D884–D891; doi:10.1093/nar/gkaf1110. Accessed from `https://lpsn.dsmz.de` on 5th of October, 2026.
- Vincent, R *et al* (2013). **MycoBank gearing up for new horizons.** IMA Fungus, 4(2), 371-9; doi:10.5598/imafungus.2013.04.02.16. Accessed from `https://www.mycobank.org` on 7th of January, 2026.

- Banki, O. *et al.* (2026). Catalogue of Life (2026-04-18 XR). Catalogue of Life Foundation, Amsterdam, Netherlands. doi:10.48580/dgxjw. Accessed from `https://www.gbif.org` on 30th of April, 2026.
- Reimer, LC *et al.* (2022). ***BacDive* in 2022: the knowledge base for standardized bacterial and archaeal data.** Nucleic Acids Res., 50(D1):D741-D74; doi:10.1093/nar/gkab961. Accessed from `https://bacdive.dsmz.de` on 5th of October, 2026.
- Public Health Information Network Vocabulary Access and Distribution System (PHIN VADS). US Edition of SNOMED CT from 1 September 2020. Value Set Name 'Microorganism', OID 2.16.840.1.114222.4.11.1009 (v12). URL: `https://www.cdc.gov/phin/php/phinvads/`
- Bartlett A *et al.* (2022). **A comprehensive list of bacterial pathogens infecting humans** *Microbiology* 168:001269; doi:10.1099/mic.0.001269


### Matching Score for Microorganisms

With ambiguous user input in `as.mo()` and all the `mo_*` functions, the returned results are chosen based on their matching score using `mo_matching_score()`. This matching score $m$, is calculated as:

$$m_{(x,n)} = \frac{l_n - 0.5 \cdot \min \begin{cases} l_n \\ \text{lev}(x, n) \end{cases}}{l_n \cdot p_n \cdot k_n}$$

where:

- $x$ is the user input;
- $n$ is a taxonomic name (genus, species, and subspecies);
- $l_n$ is the length of $n$;
- $lev$ is the Levenshtein distance function (counting any insertion as 1, and any deletion or substitution as 2) that is needed to change $x$ into $n$;
- $p_n$ is the human pathogenic prevalence group of $n$, as described below;
- $k_n$ is the taxonomic domain ('kingdom' until taxonomic reclassification of 2024) of $n$, set as Bacteria = 1, Fungi = 1.25, Protozoa = 1.5, Chromista = 1.75, Archaea = 2, others = 3.

The grouping into human pathogenic prevalence $p$ is based on recent work from Bartlett *et al.* (2022, doi:10.1099/mic.0.001269) who extensively studied medical-scientific literature to categorise all bacterial species into these groups:

- **Established**, if a taxonomic species has infected at least three persons in three or more references. These records have `prevalence = 1.15` in the microorganisms data set;
- **Putative**, if a taxonomic species has fewer than three known cases. These records have `prevalence = 1.25` in the microorganisms data set.

Furthermore,

- Genera from the World Health Organization's (WHO) Priority Pathogen List have `prevalence = 1.0` in the microorganisms data set;

- Any genus present in the **established** list also has `prevalence = 1.15` in the microorganisms data set;
- Any other genus present in the **putative** list has `prevalence = 1.25` in the microorganisms data set;
- Any other species or subspecies of which the genus is present in the two aforementioned groups, has `prevalence = 1.5` in the microorganisms data set;
- Any *non-bacterial* genus, species or subspecies of which the genus is present in the following list, has `prevalence = 1.25` in the microorganisms data set: *Absidia*, *Acanthamoeba*, *Acremonium*, *Actinomucor*, *Aedes*, *Alternaria*, *Amoeba*, *Ancylostoma*, *Angiostrongylus*, *Anisakis*, *Anopheles*, *Apiotrichum*, *Apophysomyces*, *Arthroderma*, *Ascaris*, *Aspergillus*, *Aureobasidium*, *Babesia*, *Balamuthia*, *Balantidium*, *Balantioides*, *Basidiobolus*, *Beauveria*, *Bipolaris*, *Blastobotrys*, *Blastocystis*, *Blastomyces*, *Brugia*, *Candida*, *Candidozyma*, *Capillaria*, *Chaetomium*, *Chilomastix*, *Chrysonilia*, *Chrysosporium*, *Cladophialophora*, *Cladosporium*, *Clavispora*, *Clonorchis*, *Coccidioides*, *Cokeromyces*, *Conidiobolus*, *Coniochaeta*, *Contracaecum*, *Cordylobia*, *Cryptococcus*, *Cryptosporidium*, *Cunninghamella*, *Curvularia*, *Cutaneotrichosporon*, *Cyberlindnera*, *Cyclospora*, *Cystoisospora*, *Debaryomyces*, *Debaryozyma*, *Demodex*, *Dermatobia*, *Dientamoeba*, *Diphyllobothrium*, *Dirofilaria*, *Diutina*, *Echinococcus*, *Echinostoma*, *Emergomyces*, *Emmonsia*, *Encephalitozoon*, *Entamoeba*, *Enterobius*, *Enterocytozoon*, *Epidermophyton*, *Exidia*, *Exophiala*, *Exserohilum*, *Fasciola*, *Fasciolopsis*, *Fonsecaea*, *Fusarium*, *Geotrichum*, *Giardia*, *Graphium*, *Haloarcula*, *Halobacterium*, *Halococcus*, *Hansenula*, *Hendersonula*, *Heterophyes*, *Histomonas*, *Histoplasma*, *Hortaea*, *Hymenolepis*, *Hypomyces*, *Hysterothylacium*, *Isospora*, *Kazachstania*, *Kloeckera*, *Kluyveromyces*, *Kodamaea*, *Lacazia*, *Leishmania*, *Lichtheimia*, *Loa*, *Lodderomyces*, *Lomentospora*, *Madurella*, *Magnusiomyces*, *Malassezia*, *Malbranchea*, *Mansonella*, *Metagonimus*, *Meyerozyma*, *Microascus*, *Microsporidium*, *Microsporum*, *Millerozyma*, *Mortierella*, *Mucor*, *Mycocentrospora*, *Naegleria*, *Nakaseomyces*, *Nannizzia*, *Necator*, *Nectria*, *Neocosmospora*, *Ochroconis*, *Oesophagostomum*, *Oidiodendron*, *Onchocerca*, *Opisthorchis*, *Paecilomyces*, *Paracoccidioides*, *Paragonimus*, *Pediculus*, *Penicillium*, *Phaeoacremonium*, *Phaeomoniella*, *Phialophora*, *Phlebotomus*, *Phoma*, *Pichia*, *Piedraia*, *Pithomyces*, *Pityrosporum*, *Plasmodium*, *Pneumocystis*, *Pseudallescheria*, *Pseudoscopulariopsis*, *Pseudoterranova*, *Pulex*, *Purpureocillium*, *Quambalaria*, *Rasamsonia*, *Rhinocladiella*, *Rhizomucor*, *Rhizopus*, *Rhodotorula*, *Saccharomyces*, *Saksenaea*, *Saprochaete*, *Sarcocystis*, *Sarcoptes*, *Scedosporium*, *Schistosoma*, *Schizophyllum*, *Schizosaccharomyces*, *Scolecobasidium*, *Scopulariopsis*, *Scytalidium*, *Spirometra*, *Sporobolomyces*, *Sporopachydermia*, *Sporothrix*, *Sporotrichum*, *Stachybotrys*, *Strongyloides*, *Syncephalastrum*, *Syngamus*, *Taenia*, *Talaromyces*, *Toxocara*, *Toxoplasma*, *Trichinella*, *Trichobilharzia*, *Trichoderma*, *Trichomonas*, *Trichophyton*, *Trichosporon*, *Trichostrongylus*, *Trichuris*, *Tritirachium*, *Trombicula*, *Trypanosoma*, *Tunga*, *Ulocladium*, *Ustilago*, *Verticillium*, *Wallemia*, *Wangiella*, *Wickerhamomyces*, *Wuchereria*, *Yarrowia*, or *Zygosaccharomyces*;
- All other records have `prevalence = 2.0` in the microorganisms data set.

When calculating the matching score, all characters in $x$ and $n$ are ignored that are other than A-Z, a-z, 0-9, spaces and parentheses.

All matches are sorted descending on their matching score and for all user input values, the top match will be returned. This will lead to the effect that e.g., `"E. coli"` will return the microbial ID of *Escherichia coli* ($m = 0.688$, a highly prevalent microorganism found in humans) and not *Entamoeba coli* ($m = 0.381$, a less prevalent microorganism in humans), although the latter would alphabetically come first.

### Download Our Reference Data

All reference data sets in the AMR package - including information on microorganisms, antimicrobials, and clinical breakpoints - are freely available for download in multiple formats: R, MS Excel, Apache Feather, Apache Parquet, SPSS, and Stata.

For maximum compatibility, we also provide machine-readable, tab-separated plain text files suitable for use in any software, including laboratory information systems.

Visit our website for direct download links, or explore the actual files in our GitHub repository.

## See Also

microorganisms for the data.frame that is being used to determine ID's.

The `mo_*` functions (such as `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(c(
  "sau", # WHONET code
  "stau",
  "STAU",
  "staaur",
  "S. aureus",
  "S aureus",
  "Sthafilokkockus aureus", # handles incorrect spelling
  "Staphylococcus aureus (MRSA)",
  "MRSA", # Methicillin Resistant S. aureus
  "VISA", # Vancomycin Intermediate S. aureus
  "VRSA", # Vancomycin Resistant S. aureus
  115329001 # SNOMED CT code
))

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

# input will get cleaned up with the input given in the `cleaning_regex` argument,
# which defaults to `mo_cleaning_regex()`:
cat(mo_cleaning_regex(), "\n")

as.mo("Streptococcus group A")

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")
mo_gramstain("ESCO")
mo_is_intrinsic_resistant("ESCCOL", ab = "vanco")
```

---

`as.sir` *Interpret MIC and Disk Diffusion as SIR, or Clean Existing SIR Data*

---

## Description

Clean up existing SIR values, or interpret minimum inhibitory concentration (MIC) values and disk diffusion diameters according to EUCAST or CLSI. `as.sir()` transforms the input to a new class `sir`, which is an ordered factor containing the levels `S`, `SDD`, `I`, `R`, `NI`.

Breakpoints are currently implemented from EUCAST 2011-2026 and CLSI 2011-2026, see *Details*. All breakpoints used for interpretation are available in our clinical_breakpoints data set.

## Usage

```
as.sir(x, ..., enforce_method = "auto")

NA_sir_

is.sir(x)

is_sir_eligible(x, threshold = 0.05)

## Default S3 method:
as.sir(
  x,
  S = "^(S|U|1)+$",
  I = "^(I|2)+$",
  R = "^(R|3)+$",
  NI = "^(N|NI|V|4)+$",
  SDD = "^(SDD|D|H|5)+$",
  WT = "^(WT|6)+$",
  NWT = "^(NWT|7)+$",
  NS = "^(NS|8)+$",
  info = interactive(),
  ...
)

## S3 method for class 'mic'
as.sir(
  x,
```

```
  mo = NULL,
  ab = deparse(substitute(x)),
  guideline = getOption("AMR_guideline", "EUCAST"),
  uti = NULL,
  capped_mic_handling = getOption("AMR_capped_mic_handling", "standard"),
  as_wt_nwt = identical(breakpoint_type, "ECOFF"),
  add_intrinsic_resistance = FALSE,
  reference_data = AMR::clinical_breakpoints,
 substitute_missing_r_breakpoint = getOption("AMR_substitute_missing_r_breakpoint",
    FALSE),
  include_screening = getOption("AMR_include_screening", FALSE),
  include_PKPD = getOption("AMR_include_PKPD", TRUE),
  breakpoint_type = getOption("AMR_breakpoint_type", "human"),
  host = NULL,
  language = get_AMR_locale(),
  verbose = FALSE,
  info = interactive(),
  conserve_capped_values = NULL,
  ...
)

## S3 method for class 'disk'
as.sir(
  x,
  mo = NULL,
  ab = deparse(substitute(x)),
  guideline = getOption("AMR_guideline", "EUCAST"),
  uti = NULL,
  as_wt_nwt = identical(breakpoint_type, "ECOFF"),
  add_intrinsic_resistance = FALSE,
  reference_data = AMR::clinical_breakpoints,
 substitute_missing_r_breakpoint = getOption("AMR_substitute_missing_r_breakpoint",
    FALSE),
  include_screening = getOption("AMR_include_screening", FALSE),
  include_PKPD = getOption("AMR_include_PKPD", TRUE),
  breakpoint_type = getOption("AMR_breakpoint_type", "human"),
  host = NULL,
  language = get_AMR_locale(),
  verbose = FALSE,
  info = interactive(),
  ...
)

## S3 method for class 'data.frame'
as.sir(
  x,
  ...,
  col_mo = NULL,
```

```
  guideline = getOption("AMR_guideline", "EUCAST"),
  uti = NULL,
  capped_mic_handling = getOption("AMR_capped_mic_handling", "standard"),
  as_wt_nwt = identical(breakpoint_type, "ECOFF"),
  add_intrinsic_resistance = FALSE,
  reference_data = AMR::clinical_breakpoints,
 substitute_missing_r_breakpoint = getOption("AMR_substitute_missing_r_breakpoint",
   FALSE),
  include_screening = getOption("AMR_include_screening", FALSE),
  include_PKPD = getOption("AMR_include_PKPD", TRUE),
  breakpoint_type = getOption("AMR_breakpoint_type", "human"),
  host = NULL,
  language = get_AMR_locale(),
  verbose = FALSE,
  info = interactive(),
  parallel = FALSE,
  conserve_capped_values = NULL
)

sir_interpretation_history(clean = FALSE)
```

## Arguments

| | |
|---|---|
| x | Vector of values (for class mic: MIC values in mg/L, for class disk: a disk diffusion radius in millimetres). |
| ... | For using on a data.frame: selection of columns to apply as.sir() to. Supports tidyselect language such as where(is.mic), starts_with(...), or column1:column4, and can thus also be antimicrobial selectors, e.g. as.sir(df, penicillins()). |
| enforce_method | A character string to force interpretation as a specific method, useful when the S3 class of x is lost (e.g., when called from Python via rpy2). Must be one of "auto" (default), "mic", or "disk". |
| | Otherwise: arguments passed on to methods. |
| threshold | Maximum fraction of invalid antimicrobial interpretations of x, see *Examples*. |
| S, I, R, NI, SDD, WT, NWT, NS | A case-independent regular expression to translate input to this result. This regular expression will be run *after* all non-letters and whitespaces are removed from the input. |
| info | A logical to print information about the process, defaults to TRUE only in interactive sessions. |
| mo | A vector (or column name) with characters that can be coerced to valid microorganism codes with as.mo(), can be left empty to determine it automatically. |
| ab | A vector (or column name) with characters that can be coerced to a valid antimicrobial drug code with as.ab(). |
| guideline | A guideline name (or column name) to use for SIR interpretation. Defaults to EUCAST 2026 (the latest implemented EUCAST guideline in the clinical_breakpoints data set), but can be set with the package option AMR_guideline. Currently |

supports EUCAST (2011-2026) and CLSI (2011-2026), see *Details*. Using a column name allows for straightforward interpretation of historical data, which must be analysed in the context of, for example, different years.

`uti` (Urinary Tract Infection) a vector (or column name) with logicals (`TRUE` or `FALSE`) to specify whether a UTI specific interpretation from the guideline should be chosen. For using `as.sir()` on a data.frame, this can also be a column containing logicals or when left blank, the data set will be searched for a column 'specimen', and rows within this column containing 'urin' (such as 'urine', 'urina') will be regarded isolates from a UTI. If `uti` is `NA` (unknown), non-UTI breakpoints are preferred, and UTI breakpoints are only used if no non-UTI breakpoints exist for the drug-bug combination, which is noted in `sir_interpretation_history()`. With `uti = FALSE`, UTI breakpoints are never used. See *Examples*.

`capped_mic_handling`

A character string that controls how MIC values with a cap (i.e., starting with <, <=, >, or >=) are interpreted. Supports the following options:

`"none"`

- <=, <, > and >= are ignored.

`"conservative"`

- <=, <, > and >= return `"NI"` (non-interpretable) if the *true* MIC could be at either side of the breakpoint.
- This is the only mode that preserves uncertainty for ECOFFs.

`"standard"` (default)

- <= and >= return `"NI"` (non-interpretable) if the *true* MIC could be at either side of the breakpoint.
- < always returns `"S"`, regardless of the breakpoint.
- > always returns `"R"`, regardless of the breakpoint.

`"lenient"`

- <= and < always return `"S"`, regardless of the breakpoint.
- >= and > always return `"R"`, regardless of the breakpoint.

In all modes except `"none"`, a capped value is never forced to a category that the guideline does not provide for the drug-bug combination: if a breakpoint is missing, the result is `NA` as for uncapped values. EUCAST uses an off-scale S breakpoint of 0.001 mg/L to indicate that there is no S category (stored as 0.0001 in clinical_breakpoints; any S breakpoint of 0.001 or lower is treated as such). In that case, < and <= values that cannot be `"R"` return `"I"` (or `"SDD"`), and `"standard"` and `"lenient"` return `"I"` (or `"SDD"`) instead of `"S"`.

The default `"standard"` setting ensures cautious handling of uncertain values while preserving interpretability. This option can also be set with the package option `AMR_capped_mic_handling`.

`as_wt_nwt` A logical to return `"WT"`/`"NWT"` instead of `"S"`/`"R"`. Defaults to `TRUE` when using ECOFFs, i.e., when `breakpoint_type` is set to `"ECOFF"`.

`add_intrinsic_resistance`

*(only useful when using a EUCAST guideline)* a logical to indicate whether intrinsic antibiotic resistance must also be considered for applicable bug-drug

combinations, meaning that e.g. ampicillin will always return "R" in *Klebsiella* species. Determination is based on the intrinsic_resistant data set, that itself is based on 'EUCAST Expert Rules' and 'EUCAST Intrinsic Resistance and Unusual Phenotypes' v3.3 (2021).

`reference_data` A data.frame to be used for interpretation, which defaults to the clinical_breakpoints data set. Changing this argument allows for using own interpretation guidelines. This argument must have the same column names as the clinical_breakpoints data set. Column types are coerced automatically where possible: the `mo` column is passed through `as.mo()`, the `ab` column through `as.ab()`, and plain character, numeric, or logical columns are cast to the expected type. When `reference_data` is manually set, the `guideline` argument is optional: if omitted (or if its value does not match any row in the custom data), all rows in `reference_data` are considered. If `guideline` is set to a value that exists in the `guideline` column of the custom data, only matching rows are used — useful when a single custom table contains multiple guidelines. For the R classification, the EUCAST convention is used by default: MIC values `>` `breakpoint_R` and disk diffusion values `<` `breakpoint_R` are classified as R, with values between `breakpoint_S` and `breakpoint_R` classified as I (or SDD). Only when using the standard clinical_breakpoints with a CLSI guideline are the closed-interval rules (`>=` `breakpoint_R` for MIC, `<=` `breakpoint_R` for disk) applied; custom `reference_data` always uses the open-interval (EUCAST) convention regardless of the guideline name.

`substitute_missing_r_breakpoint`
A logical to indicate that a missing clinical breakpoints for R (resistant) must be substituted with R - the default is `FALSE`. Some (especially CLSI) breakpoints only have a breakpoint for S, meaning that the outcome can only be `"S"` or `NA`. Setting this to `TRUE` will convert the `NA`s in these cases to `"R"`. Can also be set with the package option `AMR_substitute_missing_r_breakpoint`.

`include_screening`
A logical to indicate that clinical breakpoints for screening are allowed - the default is `FALSE`. Can also be set with the package option `AMR_include_screening`.

`include_PKPD` A logical to indicate that PK/PD clinical breakpoints must be applied as a last resort - the default is `TRUE`. Can also be set with the package option `AMR_include_PKPD`.

`breakpoint_type`
The type of breakpoints to use, either `"ECOFF"`, `"animal"`, or `"human"`. ECOFF stands for Epidemiological Cut-Off values. The default is `"human"`, which can also be set with the package option `AMR_breakpoint_type`. If `host` is set to values of veterinary species, this will automatically be set to `"animal"`.

`host` A vector (or column name) with characters to indicate the host. Only useful for veterinary breakpoints, as it requires `breakpoint_type = "animal"`. The values can be any text resembling the animal species, even in any of the 28 supported languages of this package. For foreign languages, be sure to set the language with `set_AMR_locale()` (though it will be automatically guessed based on the system language).

`language` Language to convert values set in `host` when using animal breakpoints. Use one of these supported language names or ISO 639-1 codes: English (en), Arabic (ar), Bengali (bn), Chinese (zh), Czech (cs), Danish (da), Dutch (nl), Finnish

(fi), French (fr), German (de), Greek (el), Hindi (hi), Indonesian (id), Italian (it), Japanese (ja), Korean (ko), Norwegian (no), Polish (pl), Portuguese (pt), Romanian (ro), Russian (ru), Spanish (es), Swahili (sw), Swedish (sv), Turkish (tr), Ukrainian (uk), Urdu (ur), or Vietnamese (vi).

`verbose` A logical to indicate that all notes should be printed during interpretation of MIC values or disk diffusion values.

`conserve_capped_values` Deprecated, use `capped_mic_handling` instead.

`col_mo` Column name of the names or codes of the microorganisms (see `as.mo()`) - the default is the first column of class `mo`. Values will be coerced using `as.mo()`.

`parallel` A logical to indicate if parallel computing must be used, defaults to `FALSE`. Requires the `future.apply` package. **A non-sequential `future::plan()` must already be active before setting** `parallel = TRUE` — for example, `future::plan(future::multisess`. An error is thrown if `parallel = TRUE` is used without a plan set by the user. Parallelism distributes columns (and optionally row batches) across workers; it is most beneficial when there are many antibiotic columns and a large number of rows.

`clean` A logical to indicate whether previously stored results should be forgotten after returning the 'logbook' with results.

## Details

*Note: The clinical breakpoints in this package were validated through, and imported from, WHONET. The public use of this `AMR` package has been endorsed by both CLSI and EUCAST. See clinical_breakpoints for more information.*

**How it Works:**

The `as.sir()` function can work in four ways:

1. For **cleaning raw / untransformed data**. The data will be cleaned to only contain valid values, namely: **S** for susceptible, **I** for intermediate or 'susceptible, increased exposure', **R** for resistant, **NI** for non-interpretable, and **SDD** for susceptible dose-dependent. Each of these can be set using a regular expression. Furthermore, `as.sir()` will try its best to clean with some intelligence. For example, mixed values with SIR interpretations and MIC values such as `"<0.25; S"` will be coerced to `"S"`. Combined interpretations for multiple test methods (as seen in laboratory records) such as `"S; S"` will be coerced to `"S"`, but a value like `"S; I"` will return `NA` with a warning that the input is invalid.
2. For **interpreting minimum inhibitory concentration (MIC) values** according to EUCAST or CLSI. You must clean your MIC values first using `as.mic()`, that also gives your columns the new data class `mic`. Also, be sure to have a column with microorganism names or codes. It will be found automatically, but can be set manually using the `mo` argument.
   - Example to apply using `dplyr`:

```
your_data %>% mutate_if(is.mic, as.sir)
your_data %>% mutate(across(where(is.mic), as.sir))
your_data %>% mutate_if(is.mic, as.sir, ab = "column_with_antibiotics", mo = "column_with_micr
your_data %>% mutate_if(is.mic, as.sir, ab = c("cipro", "ampicillin", ...), mo = c("E. coli", "

# for veterinary breakpoints, also set `host`:
```

```
your_data %>% mutate_if(is.mic, as.sir, host = "column_with_animal_species", guideline = "CLSI

# fast processing with parallel computing (requires future.apply):
as.sir(your_data, ..., parallel = TRUE)
```

- Operators like "<=" will be considered according to the `capped_mic_handling` setting. At default, an MIC value of e.g. ">=2" will return "NI" (non-interpretable) if the breakpoint is 4-8; the *true* MIC could be at either side of the breakpoint. This is to prevent that capped values from raw laboratory data would not be treated conservatively.
- **Note:** When using CLSI as the guideline, MIC values must be log2-based doubling dilutions. Values not in this format, will be automatically rounded up to the nearest log2 level as CLSI instructs, and a warning will be thrown.

3. For **interpreting disk diffusion diameters** according to EUCAST or CLSI. You must clean your disk zones first using `as.disk()`, that also gives your columns the new data class `disk`. Also, be sure to have a column with microorganism names or codes. It will be found automatically, but can be set manually using the `mo` argument.
   - Example to apply using `dplyr`:

```
your_data %>% mutate_if(is.disk, as.sir)
your_data %>% mutate(across(where(is.disk), as.sir))
your_data %>% mutate_if(is.disk, as.sir, ab = "column_with_antibiotics", mo = "column_with_mic
your_data %>% mutate_if(is.disk, as.sir, ab = c("cipro", "ampicillin", ...), mo = c("E. coli",

# for veterinary breakpoints, also set `host`:
your_data %>% mutate_if(is.disk, as.sir, host = "column_with_animal_species", guideline = "CLS

# fast processing with parallel computing (requires future.apply):
as.sir(your_data, ..., parallel = TRUE)
```

4. For **interpreting a complete data set**, with automatic determination of MIC values, disk diffusion diameters, microorganism names or codes, and antimicrobial test results. This is done very simply by running `as.sir(your_data)`.

**For points 2, 3 and 4: Use** `sir_interpretation_history()` to retrieve a data.frame with all results of all previous `as.sir()` calls. It also contains notes about interpretation, and the exact input and output values.

**Supported Guidelines:**

For interpreting MIC values as well as disk diffusion diameters, currently implemented guidelines are:

- For **clinical microbiology**: EUCAST 2011-2026 and CLSI 2011-2026;
- For **veterinary microbiology**: EUCAST 2021-2026 and CLSI 2019-2026;
- For **ECOFFs** (Epidemiological Cut-off Values): EUCAST 2020-2026 and CLSI 2022-2026.

The `guideline` argument must be set to e.g., `"EUCAST 2026"` or `"CLSI 2026"`. By simply using `"EUCAST"` (the default) or `"CLSI"` as input, the latest included version of that guideline will automatically be selected. Importantly, using a column name of your data instead, allows for straightforward interpretation of historical data that must be analysed in the context of, for example, different years.

You can set your own data set using the `reference_data` argument. The `guideline` argument will then be ignored.

It is also possible to set the default guideline with the package option AMR_guideline (e.g. in your .Rprofile file), such as:

```
options(AMR_guideline = "CLSI")
options(AMR_guideline = "CLSI 2018")
options(AMR_guideline = "EUCAST 2020")
# or to reset:
options(AMR_guideline = NULL)
```

**Working with Veterinary Breakpoints:**

When using veterinary breakpoints (i.e., setting breakpoint_type = "animal"), a column with animal species must be available or set manually using the host argument. The column must contain names like "dogs", "cats", "cattle", "swine", "horses", "poultry", or "aquatic". Other animal names like "goats", "rabbits", or "monkeys" are also recognised but may not be available in all guidelines. Matching is case-insensitive and accepts Latin-based synonyms (e.g., "bovine" for cattle and "canine" for dogs).

Regarding choice of veterinary guidelines, these might be the best options to set before analysis:

```
options(AMR_guideline = "CLSI")
options(AMR_breakpoint_type = "animal")
```

**After Interpretation:**

After using as.sir(), you can use the interpretive_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.

To determine which isolates are multi-drug resistant, be sure to run mdro() (which applies the MDR/PDR/XDR guideline from 2012 at default) on a data set that contains S/I/R values. Read more about detecting multidrug-resistant organisms here.

**Other:**

The function is.sir() detects if the input contains class sir. If the input is a data.frame or list, it iterates over all columns/items and returns a logical vector.

The base R function as.double() can be used to retrieve quantitative values from a sir object: "S" = 1, "I"/"SDD" = 2, "R" = 3. All other values are rendered NA. **Note:** Do not use as.integer(), since that (because of how R works internally) will return the factor level indices, and not these aforementioned quantitative values.

The function is_sir_eligible() returns TRUE when a column contains at most 5% potentially invalid antimicrobial interpretations, and FALSE otherwise. The threshold of 5% can be set with the threshold argument. If the input is a data.frame, it iterates over all columns and returns a logical vector.

NA_sir_ is a missing value of the new sir class, analogous to e.g. base R's NA_character_.

## Value

Ordered factor with new class sir

### Interpretation of SIR

In 2019, the European Committee on Antimicrobial Susceptibility Testing (EUCAST) has decided to change the definitions of susceptibility testing categories S, I, and R (https://www.eucast.org/bacteria/clinical-breakpoints-and-interpretation/definition-of-s-i-and-r/).

In aggregated analyses, such as proportions, counts, antibiograms, and plots, the 'I' category must be considered either susceptible or resistant. In epidemiological AMR data analysis, this follows the guideline: EUCAST considers 'I' susceptible, while CLSI considers 'I' resistant. The dose-dependent susceptibility of CLSI ('SDD') is always considered susceptible, as it equals what EUCAST defines as 'I'. All functions with a `guideline` argument for this purpose use these categories:

| `guideline` | Susceptible | Resistant |
|---|---|---|
| `"EUCAST"` | S, SDD, I, WT | R, NWT, NS |
| `"CLSI"` | S, SDD, WT | I, R, NWT, NS |

The values 'WT' (wild type) and 'NWT' (non-wild type) result from epidemiological cut-off values (ECOFFs) instead of clinical breakpoints. Since they are no clinical categories, 'WT' is always considered susceptible and 'NWT' always resistant, regardless of the guideline. The value 'NS' (non-susceptible) is always considered resistant, and 'NI' (non-interpretable) is always considered a missing result.

The `guideline` argument can be:

- **Left unset**, to use the package option `AMR_guideline`, or `"EUCAST"` if this option is not set. In the latter case, a message is shown once per session (but not in R Markdown or Quarto documents); set the option to silence it.
- `"EUCAST"`, or any EUCAST version such as `"EUCAST 2025"`, to consider 'I' susceptible, e.g., to return `"SI"` vs. `"R"` in `proportion_df()`.
- `"CLSI"`, or any CLSI version such as `"CLSI 2025"`, to consider 'I' resistant, e.g., to return `"S"` vs. `"IR"` in `proportion_df()`.
- `NULL` **or** `FALSE`, to keep S, I, and R separate. This is only possible in functions that return results per category: `count_df()`, `proportion_df()`, `sir_df()`, `ggplot_sir()`, `geom_sir()`, `labels_sir_count()`, and `bug_drug_combinations()`. Functions that return a single proportion or count, such as `susceptibility()`, `count_resistant()`, `antibiogram()`, `wisca()`, and `format()` on the result of `bug_drug_combinations()`, require 'I' to be considered either susceptible or resistant, and return an error.

Use `susceptibility()` and `resistance()` (or `count_susceptible()` and `count_resistant()`) to follow the guideline, or e.g. `proportion_SI()` and `proportion_R()` to be explicit regardless of the guideline.

The `combine_SI` argument of the functions that return results per category, `antibiogram()`, and `wisca()` is deprecated in favour of `guideline`. If set, it is followed literally and takes precedence over `guideline`.

### Download Our Reference Data

All reference data sets in the AMR package - including information on microorganisms, antimicrobials, and clinical breakpoints - are freely available for download in multiple formats: R, MS Excel, Apache Feather, Apache Parquet, SPSS, and Stata.

For maximum compatibility, we also provide machine-readable, tab-separated plain text files suitable for use in any software, including laboratory information systems.

Visit our website for direct download links, or explore the actual files in our GitHub repository.

## References

For interpretations of minimum inhibitory concentration (MIC) values and disk diffusion diameters:

- **CLSI M39: Analysis and Presentation of Cumulative Antimicrobial Susceptibility Test Data**, 2011-2026, *Clinical and Laboratory Standards Institute* (CLSI). https://clsi.org/standards/products/microbiology/documents/m39/.
- **CLSI M100: Performance Standard for Antimicrobial Susceptibility Testing**, 2011-2026, *Clinical and Laboratory Standards Institute* (CLSI). https://clsi.org/shop/standards/m100/.
- **CLSI VET01: Performance Standards for Antimicrobial Disk and Dilution Susceptibility Tests for Bacteria Isolated From Animals**, 2019-2026, *Clinical and Laboratory Standards Institute* (CLSI). https://clsi.org/standards/products/veterinary-medicine/documents/vet01/.
- **EUCAST Breakpoint tables for interpretation of MICs and zone diameters**, 2011-2026, *European Committee on Antimicrobial Susceptibility Testing* (EUCAST). https://www.eucast.org/bacteria/clinical-breakpoints-and-interpretation/clinical-breakpoint-tables/.
- **WHONET** as a source for machine-reading the clinical breakpoints (read more here), 1989-2026, *WHO Collaborating Centre for Surveillance of Antimicrobial Resistance*. https://whonet.org/.

## See Also

as.mic(), as.disk(), as.mo()

## Examples

```
example_isolates

summary(example_isolates[, 1:10]) # see all SIR results at a glance

# create some example data sets, with combined MIC values and disk zones
df_wide <- data.frame(
  microorganism = "Escherichia coli",
  amoxicillin = as.mic(8),
  cipro = as.mic(0.256),
  tobra = as.disk(16),
  genta = as.disk(18),
  ERY = "R"
)
df_long <- data.frame(
  bacteria = rep("Escherichia coli", 4),
  antibiotic = c("amoxicillin", "cipro", "tobra", "genta"),
  mics = as.mic(c(0.01, 1, 4, 8)),
  disks = as.disk(c(6, 10, 14, 18)),
  guideline = c("EUCAST 2021", "EUCAST 2022", "EUCAST 2023", "EUCAST 2024")
```

```
)
# and clean previous SIR interpretation logs
x <- sir_interpretation_history(clean = TRUE)


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

# most basic application:
as.sir(df_wide)

# return a 'logbook' about the results:
sir_interpretation_history()



## Using dplyr ----------------------------------------------------
if (require("dplyr")) {
  # approaches that all work without additional arguments:
  df_wide %>% mutate_if(is.mic, as.sir)
  df_wide %>% mutate_if(function(x) is.mic(x) | is.disk(x), as.sir)
  df_wide %>% mutate(across(where(is.mic), as.sir))

  df_wide %>% mutate_at(vars(amoxicillin:tobra), as.sir)
  df_wide %>% mutate(across(amoxicillin:tobra, as.sir))

  df_wide %>% mutate(across(aminopenicillins(), as.sir))

  # approaches that all work with additional arguments:
  df_long %>%
    # given a certain data type, e.g. MIC values
    mutate_if(is.mic, as.sir,
      mo = "bacteria",
      ab = "antibiotic",
      guideline = "guideline"
    )
  df_long %>%
    mutate(across(
      where(is.mic),
      function(x) {
        as.sir(x,
          mo = "bacteria",
          ab = "antibiotic",
          guideline = "CLSI"
        )
      }
    ))
  df_wide %>%
    # given certain columns, e.g. from 'cipro' to 'genta'
    mutate_at(vars(cipro:genta), as.sir,
      mo = "bacteria",
      guideline = "CLSI"
    )
  df_wide %>%
```

```
  mutate(across(
    cipro:genta,
    function(x) {
      as.sir(x,
        mo = "bacteria",
        guideline = "CLSI"
      )
    }
  ))

# for veterinary breakpoints, add 'host':
df_long$animal_species <- c("cats", "dogs", "horses", "cattle")
df_long %>%
  # given a certain data type, e.g. MIC values
  mutate_if(is.mic, as.sir,
    mo = "bacteria",
    ab = "antibiotic",
    host = "animal_species",
    guideline = "CLSI"
  )
df_long %>%
  mutate(across(
    where(is.mic),
    function(x) {
      as.sir(x,
        mo = "bacteria",
        ab = "antibiotic",
        host = "animal_species",
        guideline = "CLSI"
      )
    }
  ))
df_wide %>%
  mutate_at(vars(cipro:genta), as.sir,
    mo = "bacteria",
    ab = "antibiotic",
    host = "animal_species",
    guideline = "CLSI"
  )
df_wide %>%
  mutate(across(
    cipro:genta,
    function(x) {
      as.sir(x,
        mo = "bacteria",
        host = "animal_species",
        guideline = "CLSI"
      )
    }
  ))

# to include information about urinary tract infections (UTI)
data.frame(
```

```
    mo = "E. coli",
    nitrofuratoin = c("<= 2", 32),
    from_the_bladder = c(TRUE, FALSE)
  ) %>%
    as.sir(uti = "from_the_bladder")

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

  df_wide %>%
    mutate_at(vars(cipro:genta), as.sir, mo = "E. coli", uti = TRUE)
}


## Using base R ------------------------------------------------


# for single values
as.sir(
  x = as.mic(2),
  mo = as.mo("S. pneumoniae"),
  ab = "AMP",
  guideline = "EUCAST"
)

as.sir(
  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 SIR values ----------------------------------

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

as.sir(c(1, 2, 3))
as.sir(c(1, 2, 3), S = 3, I = 2, R = 1)

sir_data <- as.sir(c(rep("S", 474), rep("I", 36), rep("R", 370)))
is.sir(sir_data)
plot(sir_data) # for percentages
barplot(sir_data) # for frequencies

# as common in R, you can use as.integer() to return factor indices:
as.integer(as.sir(c("S", "SDD", "I", "R", "NI", NA)))
```

```
# but for computational use, as.double() will return 1 for S, 2 for I/SDD, and 3 for R:
as.double(as.sir(c("S", "SDD", "I", "R", "NI", NA)))

# the dplyr way
if (require("dplyr")) {
  example_isolates %>%
    mutate_at(vars(PEN:RIF), as.sir)
  # same:
  example_isolates %>%
    as.sir(PEN:RIF)

  # fastest way to transform all columns with already valid AMR results to class `sir`:
  example_isolates %>%
    mutate_if(is_sir_eligible, as.sir)

  # since dplyr 1.0.0, this can also be the more impractical:
  # example_isolates %>%
  #   mutate(across(where(is_sir_eligible), as.sir))
}
```

---

`atc_online_property` *Get ATC Properties from WHOCC Website*

---

## Description

Gets data from the WHOCC website to determine properties of an Anatomical Therapeutic Chemical (ATC) (e.g. an antimicrobial), such as the name, defined daily dose (DDD) or standard unit.

## Usage

```
atc_online_property(
  atc_code,
  property,
  administration = "O",
  url = "https://atcddd.fhi.no/atc_ddd_index/?code=%s&showdescription=no",
 url_vet = "https://atcddd.fhi.no/atcvet/atcvet_index/?code=%s&showdescription=no"
)

atc_online_groups(atc_code, ...)

atc_online_ddd(atc_code, ...)

atc_online_ddd_units(atc_code, ...)
```

## Arguments

| | |
|---|---|
| `atc_code` | A character (vector) with ATC code(s) of antimicrobials, will be coerced with `as.ab()` and `ab_atc()` internally if not a valid ATC code. |

| | |
|---|---|
| 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 WHOCC. The sign %s can be used as a placeholder for ATC codes. |
| url_vet | URL of website of the WHOCC for veterinary medicine. The sign %s can be used as a placeholder for ATC_vet codes (that all start with "Q"). |
| ... | Arguments to pass on to atc_property. |

## Details

Options for argument 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" = millilitre (e.g. eyedrops)

**N.B. This function requires an internet connection and only works if the following packages are installed:** curl**,** rvest**,** xml2**.**

## Source

https://atcddd.fhi.no/atc_ddd_alterations__cumulative/ddd_alterations/abbrevations/

## Examples

```
if (requireNamespace("curl") && requireNamespace("rvest") && requireNamespace("xml2")) {
  # oral DDD (Defined Daily Dose) of amoxicillin
  atc_online_property("J01CA04", "DDD", "O")
  atc_online_ddd(ab_atc("amox"))

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

 atc_online_property("J01CA04", property = "groups") # search hierarchical groups of amoxicillin
}
```

---

# 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

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

## Arguments

| | |
|---|---|
| `tbl` | A data.frame or list. |
| `width` | Number of characters to present the visual availability - the default is 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

## Examples

```
availability(example_isolates)

if (require("dplyr")) {
  example_isolates %>%
    filter(mo == as.mo("Escherichia coli")) %>%
```

```
    select_if(is.sir) %>%
    availability()
}
```

---

`av_from_text` *Retrieve Antiviral Drug Names and Doses from Clinical Text*

---

## Description

Use this function on e.g. clinical texts from health care records. It returns a list with all antiviral drugs, doses and forms of administration found in the texts.

## Usage

```
av_from_text(
  text,
  type = c("drug", "dose", "administration"),
  collapse = NULL,
  translate_av = FALSE,
  thorough_search = NULL,
  info = interactive(),
  ...
)
```

## Arguments

| | |
|---|---|
| `text` | Text to analyse. |
| `type` | Type of property to search for, either `"drug"`, `"dose"` or `"administration"`, see *Examples*. |
| `collapse` | A character to pass on to `paste(, collapse = ...)` to only return one character per element of `text`, see *Examples*. |
| `translate_av` | If `type = "drug"`: a column name of the antivirals data set to translate the antibiotic abbreviations to, using `av_property()`. The default is `FALSE`. Using `TRUE` is equal to using "name". |
| `thorough_search` | A logical to indicate whether the input must be extensively searched for misspelling and other faulty input values. Setting this to `TRUE` will take considerably more time than when using `FALSE`. At default, it will turn `TRUE` when all input elements contain a maximum of three words. |
| `info` | A logical to indicate whether a progress bar should be printed - the default is `TRUE` only in interactive mode. |
| `...` | Arguments passed on to `as.av()`. |

## Details

This function is also internally used by `as.av()`, although it then only searches for the first drug name and will throw a note if more drug names could have been returned. Note: the `as.av()` function may use very long regular expression to match brand names of antiviral drugs. This may fail on some systems.

**Argument `type`:**

At default, the function will search for antiviral drug names. All text elements will be searched for official names, ATC codes and brand names. As it uses `as.av()` internally, it will correct for misspelling.

With `type = "dose"` (or similar, like "dosing", "doses"), all text elements will be searched for numeric values that are higher than 100 and do not resemble years. The output will be numeric. It supports any unit (g, mg, IE, etc.) and multiple values in one clinical text, see *Examples*.

With `type = "administration"` (or abbreviations, like "admin", "adm"), all text elements will be searched for a form of drug administration. It supports the following forms (including common abbreviations): buccal, implant, inhalation, instillation, intravenous, nasal, oral, parenteral, rectal, sublingual, transdermal and vaginal. Abbreviations for oral (such as 'po', 'per os') will become "oral", all values for intravenous (such as 'iv', 'intraven') will become "iv". It supports multiple values in one clinical text, see *Examples*.

**Argument `collapse`:**

Without using `collapse`, this function will return a list. This can be convenient to use e.g. inside a `mutate()`):
`df %>% mutate(avx = av_from_text(clinical_text))`

The returned AV codes can be transformed to official names, groups, etc. with all `av_*` functions such as `av_name()` and `av_group()`, or by using the `translate_av` argument.

With using `collapse`, this function will return a character:
`df %>% mutate(avx = av_from_text(clinical_text, collapse = "|"))`

## Value

A list, or a character if `collapse` is not `NULL`

## Examples

```
av_from_text("28/03/2020 valaciclovir po tid")
av_from_text("28/03/2020 valaciclovir po tid", type = "admin")
```

---

`av_property` *Get Properties of an Antiviral Drug*

---

## Description

Use these functions to return a specific property of an antiviral drug from the antivirals data set. All input values will be evaluated internally with `as.av()`.

## Usage

```
av_name(x, language = get_AMR_locale(), tolower = FALSE, ...)

av_cid(x, ...)

av_synonyms(x, ...)

av_tradenames(x, ...)

av_group(x, language = get_AMR_locale(), ...)

av_atc(x, ...)

av_loinc(x, ...)

av_ddd(x, administration = "oral", ...)

av_ddd_units(x, administration = "oral", ...)

av_info(x, language = get_AMR_locale(), ...)

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

av_property(x, property = "name", language = get_AMR_locale(), ...)
```

## Arguments

| | |
|---|---|
| x | Any (vector of) text that can be coerced to a valid antiviral drug code with as.av(). |
| language | Language of the returned text - the default is system language (see get_AMR_locale()) and can also be set with the package option AMR_locale. Use language = NULL or language = "" to prevent translation. |
| tolower | A logical to indicate whether the first character of every output should be transformed to a lower case character. |
| ... | Other arguments passed on to as.av(). |
| administration | Way of administration, either "oral" or "iv". |
| open | Browse the URL using utils::browseURL(). |
| property | One of the column names of one of the antivirals data set: "av", "name", "atc", "cid", "atc_group", "synonyms", "oral_ddd", "oral_units", "iv_ddd", "iv_units", or "loinc". |

## Details

All output will be translated where possible.

The function av_url() will return the direct URL to the official WHO website. A warning will be returned if the required ATC code is not available.

## Value

- An integer in case of av_cid()
- A named list in case of av_info() and multiple av_atc()/av_synonyms()/av_tradenames()
- A double in case of av_ddd()
- A character in all other cases

## Source

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

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

## Download Our Reference Data

All reference data sets in the AMR package - including information on microorganisms, antimicrobials, and clinical breakpoints - are freely available for download in multiple formats: R, MS Excel, Apache Feather, Apache Parquet, SPSS, and Stata.

For maximum compatibility, we also provide machine-readable, tab-separated plain text files suitable for use in any software, including laboratory information systems.

Visit our website for direct download links, or explore the actual files in our GitHub repository.

## See Also

antivirals

## Examples

```
# all properties:
av_name("ACI")
av_atc("ACI")
av_cid("ACI")
av_synonyms("ACI")
av_tradenames("ACI")
av_group("ACI")
av_url("ACI")

# lowercase transformation
av_name(x = c("ACI", "VALA"))
av_name(x = c("ACI", "VALA"), tolower = TRUE)

# defined daily doses (DDD)
av_ddd("ACI", "oral")
av_ddd_units("ACI", "oral")
av_ddd("ACI", "iv")
av_ddd_units("ACI", "iv")

av_info("ACI") # all properties as a list
```

```
# all av_* functions use as.av() internally, so you can go from 'any' to 'any':
av_atc("ACI")
av_group("J05AB01")
av_loinc("abacavir")
av_name("29113-8")
av_name(135398513)
av_name("J05AB01")
```

---

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 publishable/printable format, see *Examples*.

## Usage

```
bug_drug_combinations(
  x,
  col_mo = NULL,
  FUN = mo_shortname,
  include_n_rows = FALSE,
  guideline = getOption("AMR_guideline", "EUCAST"),
  ...
)

## S3 method for class 'bug_drug_combinations'
format(
  x,
  translate_ab = "name (ab, atc)",
  language = get_AMR_locale(),
  minimum = 30,
  combine_SI,
  guideline = attr(x, "guideline"),
  add_ab_group = TRUE,
  remove_intrinsic_resistant = FALSE,
  decimal.mark = getOption("OutDec"),
  big.mark = ifelse(decimal.mark == ",", ".", ","),
  ...
)
```

## Arguments

| | |
|---|---|
| x | A data set with antimicrobials columns, such as amox, AMX and AMC. |
| col_mo | Column name of the names or codes of the microorganisms (see `as.mo()`) - the default is the first column of class `mo`. Values will be coerced using `as.mo()`. |

| | |
|---|---|
| FUN | The function to call on the mo column to transform the microorganism codes - the default is mo_shortname(). |
| include_n_rows | A logical to indicate if the total number of rows must be included in the output. |
| guideline | A guideline name, "EUCAST" or "CLSI" (or any version thereof, e.g. "CLSI 2025"), to determine whether the 'I' category is considered susceptible (EUCAST) or resistant (CLSI) when calculating resistance with format(). The default is the package option AMR_guideline, or "EUCAST" if it is not set. In bug_drug_combinations(), this can also be NULL or FALSE, after which format() requires its own guideline to be set. In format(), the default is the guideline used in bug_drug_combinations(), which can be overridden by setting it. See section *Interpretation of SIR* below. |
| ... | Arguments passed on to FUN. |
| translate_ab | A character of length 1 containing column names of the antimicrobials data set. |
| language | Language of the returned text - the default is the current system language (see get_AMR_locale()) and can also be set with the package option 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 | *(deprecated in favour of* guideline*)* A logical. If set, it takes precedence over guideline: TRUE bases resistance on R only, and FALSE on SDD, I, and R. |
| add_ab_group | A logical to indicate where the group of the antimicrobials must be included as a first column. |
| remove_intrinsic_resistant | logical to indicate that rows and columns 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 and returns a table ready for reporting/publishing. Following the guideline, resistance is based on R (EUCAST) or on I and R (CLSI), see section *Interpretation of SIR* below. This table can also directly be used in R Markdown / Quarto without the need for e.g. knitr::kable().

## Value

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

### Interpretation of SIR

In 2019, the European Committee on Antimicrobial Susceptibility Testing (EUCAST) has decided to change the definitions of susceptibility testing categories S, I, and R (https://www.eucast.org/bacteria/clinical-breakpoints-and-interpretation/definition-of-s-i-and-r/).

In aggregated analyses, such as proportions, counts, antibiograms, and plots, the 'I' category must be considered either susceptible or resistant. In epidemiological AMR data analysis, this follows the guideline: EUCAST considers 'I' susceptible, while CLSI considers 'I' resistant. The dose-dependent susceptibility of CLSI ('SDD') is always considered susceptible, as it equals what EUCAST defines as 'I'. All functions with a `guideline` argument for this purpose use these categories:

| `guideline` | Susceptible | Resistant |
|---|---|---|
| `"EUCAST"` | S, SDD, I, WT | R, NWT, NS |
| `"CLSI"` | S, SDD, WT | I, R, NWT, NS |

The values 'WT' (wild type) and 'NWT' (non-wild type) result from epidemiological cut-off values (ECOFFs) instead of clinical breakpoints. Since they are no clinical categories, 'WT' is always considered susceptible and 'NWT' always resistant, regardless of the guideline. The value 'NS' (non-susceptible) is always considered resistant, and 'NI' (non-interpretable) is always considered a missing result.

The `guideline` argument can be:

- **Left unset**, to use the package option `AMR_guideline`, or `"EUCAST"` if this option is not set. In the latter case, a message is shown once per session (but not in R Markdown or Quarto documents); set the option to silence it.
- `"EUCAST"`, or any EUCAST version such as `"EUCAST 2025"`, to consider 'I' susceptible, e.g., to return `"SI"` vs. `"R"` in `proportion_df()`.
- `"CLSI"`, or any CLSI version such as `"CLSI 2025"`, to consider 'I' resistant, e.g., to return `"S"` vs. `"IR"` in `proportion_df()`.
- `NULL` **or** `FALSE`, to keep S, I, and R separate. This is only possible in functions that return results per category: `count_df()`, `proportion_df()`, `sir_df()`, `ggplot_sir()`, `geom_sir()`, `labels_sir_count()`, and `bug_drug_combinations()`. Functions that return a single proportion or count, such as `susceptibility()`, `count_resistant()`, `antibiogram()`, `wisca()`, and `format()` on the result of `bug_drug_combinations()`, require 'I' to be considered either susceptible or resistant, and return an error.

Use `susceptibility()` and `resistance()` (or `count_susceptible()` and `count_resistant()`) to follow the guideline, or e.g. `proportion_SI()` and `proportion_R()` to be explicit regardless of the guideline.

The `combine_SI` argument of the functions that return results per category, `antibiogram()`, and `wisca()` is deprecated in favour of `guideline`. If set, it is followed literally and takes precedence over `guideline`.

### Examples

```
# example_isolates is a data set available in the AMR package.
# run ?example_isolates for more info.
```

```
example_isolates


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

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

bug_drug_combinations(example_isolates,
  FUN = function(x) {
    ifelse(x == as.mo("Escherichia coli"),
      "E. coli",
      "Others"
    )
  }
)
```

---

## clinical_breakpoints *Data Set with Clinical Breakpoints for SIR Interpretation*

---

### Description

Data set containing clinical breakpoints to interpret MIC and disk diffusion to SIR values, according to international guidelines. This data set contains breakpoints for humans, 7 different animal groups, and ECOFFs.

These breakpoints are currently implemented:

- For **clinical microbiology**: EUCAST 2011-2026 and CLSI 2011-2026;
- For **veterinary microbiology**: EUCAST 2021-2026 and CLSI 2019-2026;
- For **ECOFFs** (Epidemiological Cut-off Values): EUCAST 2020-2026 and CLSI 2022-2026.

Use `as.sir()` to transform MICs or disks measurements to SIR values.

### Usage

```
clinical_breakpoints
```

### Format

A tibble with 61 389 observations and 15 variables:

- `guideline`
  Name of the guideline

- `type`
  Breakpoint type, either `"ECOFF"`, `"animal"`, or `"human"`
- `host`
  Host of infectious agent. This is mostly useful for veterinary breakpoints and is either `"ECOFF"`, `"aquatic"`, `"cats"`, `"cattle"`, `"dogs"`, `"horse"`, `"human"`, `"poultry"`, or `"swine"`
- `method`
  Testing method, either `"DISK"` or `"MIC"`
- `site`
  Body site for which the breakpoint must be applied, e.g. "Oral" or "Respiratory"
- `mo`
  Microbial ID, see `as.mo()`
- `rank_index`
  Taxonomic rank index of `mo`, used to choose the most specific breakpoint (the lowest index): 1 for subspecies and infraspecies, 2 for species, 2.5 for species groups (e.g., *Streptococcus* group A), 3 for genus, 4 for family, 5 for order, 6 for other groups (e.g., Gram-negative or anaerobic bacteria), and 7 for non-species related (PK-PD) breakpoints. Blocking rows (see *Details*) have the rank index of their organism + 0.1, e.g., 2.6 for a species group, so that a breakpoint for the same organism always takes precedence.
- `ab`
  Antimicrobial code as used by this package, EARS-Net and WHONET, 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", is `NA` for blocking rows (see *Details*). EUCAST's off-scale MIC breakpoint of 0.001 mg/L, which indicates that there is no S category, is stored as 0.0001 to keep all isolates out of "S".
- `breakpoint_R`
  Highest MIC value or lowest number of millimetres that leads to "R", can be `NA`
- `uti`
  A logical value (`TRUE`/`FALSE`) to indicate whether the rule applies to a urinary tract infection (UTI)
- `is_SDD`
  A logical value (`TRUE`/`FALSE`) to indicate whether the intermediate range between "S" and "R" should be interpreted as "SDD", instead of "I". This currently applies to 72 breakpoints.
- `note`
  Notes from the guideline that apply to the breakpoint, if available

## Details

### Different Types of Breakpoints:

Supported types of breakpoints are ECOFF, animal, and human. ECOFF (Epidemiological cut-off) values are used in antimicrobial susceptibility testing to differentiate between wild-type and non-wild-type strains of bacteria or fungi.

The default is "human", which can also be set with the package option AMR_breakpoint_type. Use as.sir(..., breakpoint_type = ...) to interpret raw data using a specific breakpoint type, e.g. as.sir(..., breakpoint_type = "ECOFF") to use ECOFFs.

**Blocking Rows:**

Where EUCAST lists an organism but gives no breakpoint for it (e.g. "IE", "-", "Note", breakpoints in brackets, an organism excluded with "except", or no S category for a zone diameter), as.sir() must not fall back to the breakpoint of a broader taxon, such as the breakpoint for *Vibrio* spp. where EUCAST gives "IE" for *V. fluvialis*. Such cases are included as "blocking rows": rows in which breakpoint_S and breakpoint_R are both NA and the note starts with "[No breakpoint]" followed by the reason. as.sir() returns NA for these. This currently applies to 15 660 rows. Remove them with subset(clinical_breakpoints, !is.na(breakpoint_S)) if only actual breakpoints are needed.

Where another guideline has no S category for a zone diameter, such as CLSI for fluconazole in *C. glabrata* (SDD and R only), only breakpoint_R is given, and the note starts with "[No S category]". as.sir() then returns R where applicable, and NA otherwise.

**Imported From WHONET:**

Some breakpoints in this package were validated through and imported from WHONET, a free desktop Windows application developed and supported by the WHO Collaborating Centre for Surveillance of Antimicrobial Resistance. More can be read on their website. The developers of WHONET and this AMR package have been in contact about sharing their work. We highly appreciate their great development on the WHONET software.

From WHONET, imported were:

- All CLSI breakpoints, including ECOFF
- All non-human EUCAST breakpoints, including ECOFF
- Human EUCAST breakpoints between 2011 and 2018
- Human EUCAST antifungal breakpoints between 2019 and 2025

Human EUCAST breakpoints from 2019 onwards (antifungal breakpoints from 2026 onwards) were retrieved directly from the EUCAST Clinical Breakpoint Tables published on https://www.eucast.org, using this script.

Our WHONET import script can be found here.

**Response From CLSI and EUCAST:**

The CEO of CLSI and the chairman of EUCAST have endorsed the work and public use of this AMR package (and consequently the use of their breakpoints) in June 2023, when future development of distributing clinical breakpoints was discussed in a meeting between CLSI, EUCAST, WHO, developers of WHONET software, and developers of this AMR package.

**Download Note:**

This AMR package (and the WHONET software as well) contains rather complex internal methods to apply the guidelines. For example, some breakpoints must be applied on certain species groups (which are in case of this package available through the microorganisms.groups data set). It is important that this is considered when implementing the breakpoints for own use.

### Download Our Reference Data

All reference data sets in the AMR package - including information on microorganisms, antimicrobials, and clinical breakpoints - are freely available for download in multiple formats: R, MS Excel, Apache Feather, Apache Parquet, SPSS, and Stata.

For maximum compatibility, we also provide machine-readable, tab-separated plain text files suitable for use in any software, including laboratory information systems.

Visit our website for direct download links, or explore the actual files in our GitHub repository.

### See Also

intrinsic_resistant

### Examples

```
clinical_breakpoints
```

---

## 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()` from the `dplyr` package and also 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,
  guideline = getOption("AMR_guideline", "EUCAST")
)

count_susceptible(
  ...,
  only_all_tested = FALSE,
  guideline = getOption("AMR_guideline", "EUCAST")
)

count_S(..., only_all_tested = FALSE)

count_SI(..., only_all_tested = FALSE)

count_I(..., only_all_tested = FALSE)
```

```
count_IR(..., only_all_tested = FALSE)

count_R(..., only_all_tested = FALSE)

count_all(..., only_all_tested = FALSE)

n_sir(..., only_all_tested = FALSE)

count_df(
  data,
  translate_ab = "name",
  language = get_AMR_locale(),
  combine_SI,
  guideline = getOption("AMR_guideline", "EUCAST")
)
```

## Arguments

| | |
|---|---|
| `...` | One or more vectors (or columns) with antibiotic interpretations. They will be transformed internally with `as.sir()` 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 antimicrobials, see section *Combination Therapy* below. Use `sir_combine()` to create a new variable for a combination therapy. |
| `guideline` | A guideline name, `"EUCAST"` or `"CLSI"` (or any version thereof, e.g. `"CLSI 2025"`), to determine whether the 'I' category is considered susceptible (EUCAST) or resistant (CLSI). The default is the package option `AMR_guideline`, or `"EUCAST"` if it is not set. For functions that return results per category, i.e., `count_df()`, `proportion_df()`, `sir_df()`, `ggplot_sir()`, `geom_sir()`, and `labels_sir_count()`, this can also be `NULL` or `FALSE` to keep S, I, and R separate. See section *Interpretation of SIR* below. |
| `data` | A data.frame containing columns with class `sir` (see `as.sir()`). |
| `translate_ab` | A column name of the antimicrobials data set to translate the antibiotic abbreviations to, using `ab_property()`. |
| `language` | Language of the returned text - the default is the current system language (see `get_AMR_locale()`) and can also be set with the package option `AMR_locale`. Use `language = NULL` or `language = ""` to prevent translation. |
| `combine_SI` | *(deprecated in favour of* `guideline`*)* A logical. If set, it takes precedence over `guideline`: `TRUE` merges S, SDD, and I into `"SI"` vs. `"R"`, and `FALSE` keeps S, I, and R separate. |

## Details

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

The function n_sir() is an alias of count_all(). They can be used to count all available isolates, i.e. where the input antimicrobial has an available result (S, I or R). For multiple antimicrobials, these are isolates with a result for any of the antimicrobials, or with only_all_tested = TRUE, for all of the antimicrobials (see *Combination Therapy* below). Their use is equal to dplyr's n_distinct(). For one antimicrobial, their function is equal to count_susceptible(...) + count_resistant(...).

The function count_df() takes any variable from data that has an sir class (created with as.sir()) and counts the number of S's, I's and R's. It also supports grouped variables. The function sir_df() works exactly like count_df(), but adds the percentage of S, I and R.

## Value

An integer

## Interpretation of SIR

In 2019, the European Committee on Antimicrobial Susceptibility Testing (EUCAST) has decided to change the definitions of susceptibility testing categories S, I, and R (https://www.eucast.org/bacteria/clinical-breakpoints-and-interpretation/definition-of-s-i-and-r/).

In aggregated analyses, such as proportions, counts, antibiograms, and plots, the 'I' category must be considered either susceptible or resistant. In epidemiological AMR data analysis, this follows the guideline: EUCAST considers 'I' susceptible, while CLSI considers 'I' resistant. The dose-dependent susceptibility of CLSI ('SDD') is always considered susceptible, as it equals what EUCAST defines as 'I'. All functions with a guideline argument for this purpose use these categories:

| guideline | Susceptible | Resistant |
|---|---|---|
| "EUCAST" | S, SDD, I, WT | R, NWT, NS |
| "CLSI" | S, SDD, WT | I, R, NWT, NS |

The values 'WT' (wild type) and 'NWT' (non-wild type) result from epidemiological cut-off values (ECOFFs) instead of clinical breakpoints. Since they are no clinical categories, 'WT' is always considered susceptible and 'NWT' always resistant, regardless of the guideline. The value 'NS' (non-susceptible) is always considered resistant, and 'NI' (non-interpretable) is always considered a missing result.

The guideline argument can be:

- **Left unset**, to use the package option AMR_guideline, or "EUCAST" if this option is not set. In the latter case, a message is shown once per session (but not in R Markdown or Quarto documents); set the option to silence it.
- "EUCAST", or any EUCAST version such as "EUCAST 2025", to consider 'I' susceptible, e.g., to return "SI" vs. "R" in proportion_df().
- "CLSI", or any CLSI version such as "CLSI 2025", to consider 'I' resistant, e.g., to return "S" vs. "IR" in proportion_df().
- NULL **or** FALSE, to keep S, I, and R separate. This is only possible in functions that return results per category: count_df(), proportion_df(), sir_df(), ggplot_sir(), geom_sir(),

labels_sir_count(), and bug_drug_combinations(). Functions that return a single proportion or count, such as susceptibility(), count_resistant(), antibiogram(), wisca(), and format() on the result of bug_drug_combinations(), require 'I' to be considered either susceptible or resistant, and return an error.

Use susceptibility() and resistance() (or count_susceptible() and count_resistant()) to follow the guideline, or e.g. proportion_SI() and proportion_R() to be explicit regardless of the guideline.

The combine_SI argument of the functions that return results per category, antibiogram(), and wisca() is deprecated in favour of guideline. If set, it is followed literally and takes precedence over guideline.

### Combination Therapy

Results are ordered from susceptible to resistant: S (and WT), SDD, I, R (and NWT, NS). With method = "most_susceptible" (in sir_combine()), the most susceptible result of the antimicrobials is returned, and with method = "most_resistant", the most resistant result. The value "NI" (non-interpretable) is considered a missing result.

A missing result can only be ignored if it cannot change the conclusion: a susceptible result for "most_susceptible", and a non-susceptible result for "most_resistant". Under EUCAST, the 'I' category is considered susceptible (*susceptible, increased exposure*), and under CLSI, it is not. With only_all_tested = TRUE, any missing result leads to NA.

This is an example for two antimicrobials, Drug A and Drug B, under EUCAST (the order of the antimicrobials does not matter):

| | | Result of method = "most_susceptible" | | Result of method = "most_resistant" | |
|---|---|---|---|---|---|
| | | only_all_tested = | | only_all_tested = | |
| Drug A | Drug B | FALSE | TRUE | FALSE | TRUE |
| S | S | S | S | S | S |
| S | I | S | S | I | I |
| S | R | S | S | R | R |
| I | R | I | I | R | R |
| R | R | R | R | R | R |
| S | <NA> | S | <NA> | <NA> | <NA> |
| I | <NA> | I | <NA> | <NA> | <NA> |
| R | <NA> | <NA> | <NA> | R | <NA> |
| <NA> | <NA> | <NA> | <NA> | <NA> | <NA> |

Under CLSI, the combination of I with a missing result is <NA> for "most_susceptible" and I for "most_resistant".

The count_*() and proportion_*() functions apply these rules when more than one variable is used. With only_all_tested = FALSE (default), the functions for susceptibility, such as susceptibility()

and count_susceptible(), count an isolate as susceptible if any of the antimicrobials is susceptible ("most_susceptible"), and the functions for resistance, such as resistance() and count_resistant(), count an isolate as resistant if any of the antimicrobials is resistant ("most_resistant"). An isolate is considered tested if it has such a result, or if it was tested for all antimicrobials. Therefore, for combinations with only_all_tested = FALSE:

```
   count_S()    +   count_I()    +   count_R()    >= count_all()
proportion_S() + proportion_I() + proportion_R() >= 1
```

With only_all_tested = TRUE, all these functions count the result of the combination therapy ("most_susceptible") of isolates tested for all antimicrobials, so that:

```
   count_S()    +   count_I()    +   count_R()    = count_all()
proportion_S() + proportion_I() + proportion_R() = 1
```

In antibiogram() and wisca(), the result of the combination therapy ("most_susceptible") is used, the 'I' category is considered susceptible or not following the guideline argument ('SDD' is always considered susceptible), and the result is then reduced to S or R.

## See Also

proportion_* to calculate microbial resistance and susceptibility.

## Examples

```
# example_isolates is a data set available in the AMR package.
# run ?example_isolates for more info.

# base R ------------------------------------------------------------
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$AMX)
n_sir(example_isolates$AMX)

# n_sir() 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_sir(example_isolates$AMX)
```

```
# dplyr ------------------------------------------------------------

if (require("dplyr")) {
  example_isolates %>%
    group_by(ward) %>%
    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_sir(CIP), # same - analogous to n_distinct
      total = n()
    ) # NOT the number of tested isolates!

  # Number of available isolates for a whole antibiotic class
  # (i.e., in this data set columns GEN, TOB, AMK, KAN)
  example_isolates %>%
    group_by(ward) %>%
    summarise(across(aminoglycosides(), n_sir))

  # 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(ward, AMX, CIP) %>%
    group_by(ward) %>%
    count_df(translate = FALSE)
}
```

---

## custom_interpretive_rules

*Define Custom Interpretive Rules*

---

## Description

Define custom interpretive rules for your organisation or specific analysis and use the output of this function in `interpretive_rules()`.

## Usage

```
custom_interpretive_rules(...)
```

## Arguments

| | |
|---|---|
| ... | Rules in formula notation, see below for instructions, and in *Examples*. |

## Details

Some organisations have their own adoption of interpretive rules. This function can be used to define custom rules to be used in the `interpretive_rules()` function.

**Basics:**

If you are familiar with the `case_when()` function of the `dplyr` package, you will recognise the input method to set your own rules. Rules must be set using what R considers to be the 'formula notation'. The rule itself is written *before* the tilde (~) and the consequence of the rule is written *after* the tilde:

```
x <- custom_interpretive_rules(TZP == "S" ~ aminopenicillins == "S",
                               TZP == "R" ~ aminopenicillins == "R")
```

These are two custom interpretive rules: if TZP (piperacillin/tazobactam) is "S", all aminopenicillins (ampicillin and amoxicillin) must be made "S", and if TZP is "R", aminopenicillins must be made "R". These rules can also be printed to the console, so it is immediately clear how they work:

```
x
#> A set of custom interpretive rules:
#>
#>   1. If TZP is "S" then set to  S :
#>      amoxicillin (AMX), ampicillin (AMP)
#>
#>   2. If TZP is "R" then set to  R :
#>      amoxicillin (AMX), ampicillin (AMP)
```

The rules (the part *before* the tilde, in above example `TZP == "S"` and `TZP == "R"`) must be evaluable in your data set: it should be able to run as a filter in your data set without errors. This means for the above example that the column `TZP` must exist. We will create a sample data set and test the rules set:

```
df <- data.frame(mo = c("Escherichia coli", "Klebsiella pneumoniae"),
                 TZP = as.sir("R"),
                 ampi = as.sir("S"),
                 cipro = as.sir("S"))
df
#>                    mo TZP ampi cipro
```

```
#> 1      Escherichia coli   R    S     S
#> 2 Klebsiella pneumoniae   R    S     S

interpretive_rules(df,
                   rules = "custom",
                   custom_rules = x,
                   info = FALSE,
                   overwrite = TRUE)
#>                       mo TZP ampi cipro
#> 1      Escherichia coli   R    R     S
#> 2 Klebsiella pneumoniae   R    R     S
```

**Using taxonomic properties in rules:**

There is one exception in columns used for the rules: all column names of the microorganisms data set can also be used, but do not have to exist in the data set. These column names are: `"mo"`, `"fullname"`, `"status"`, `"domain"`, `"kingdom"`, `"phylum"`, `"class"`, `"order"`, `"family"`, `"genus"`, `"species"`, `"subspecies"`, `"rank"`, `"ref"`, `"oxygen_tolerance"`, `"morphology"`, `"source"`, `"lpsn"`, `"lpsn_parent"`, `"lpsn_renamed_to"`, `"mycobank"`, `"mycobank_parent"`, `"mycobank_renamed_to"`, `"gbif"`, `"gbif_parent"`, `"gbif_renamed_to"`, `"prevalence"`, and `"snomed"`. Thus, this next example will work as well, despite the fact that the `df` data set does not contain a column genus:

```
y <- custom_interpretive_rules(
  TZP == "S" & genus == "Klebsiella" ~ aminopenicillins == "S",
  TZP == "R" & genus == "Klebsiella" ~ aminopenicillins == "R"
)

interpretive_rules(df,
                   rules = "custom",
                   custom_rules = y,
                   info = FALSE,
                   overwrite = TRUE)
#>                       mo TZP ampi cipro
#> 1      Escherichia coli   R    S     S
#> 2 Klebsiella pneumoniae   R    R     S
```

**Sharing rules among multiple users:**

The rules set (the `y` object in this case) could be exported to a shared file location using `saveRDS()` if you collaborate with multiple users. The custom rules set could then be imported using `readRDS()`.

**Usage of multiple antimicrobials and antimicrobial group names:**

You can define antimicrobial groups instead of single antimicrobials for the rule consequence, which is the part *after* the tilde (~). In the examples above, the antimicrobial group `aminopenicillins` includes both ampicillin and amoxicillin.

Rules can also be applied to multiple antimicrobials and antimicrobial groups simultaneously. Use the `c()` function to combine multiple antimicrobials. For instance, the following example sets all aminopenicillins and ureidopenicillins to "R" if column TZP (piperacillin/tazobactam) is "R":

```
x <- custom_interpretive_rules(TZP == "R" ~ c(aminopenicillins, ureidopenicillins) == "R")
```

```
x
#> A set of custom interpretive rules:
#>
#>   1. If TZP is "R" then set to "R":
#>    amoxicillin (AMX), ampicillin (AMP), azlocillin (AZL), mezlocillin (MEZ), piperacillin (PIP), p
```

These 43 antimicrobial groups are allowed in the rules (case-insensitive) and can be used in any combination:

- aminocoumarins
  (clorobiocin, novobiocin, and penicillin/novobiocin)
- aminoglycosides
  (amikacin, amikacin/fosfomycin, apramycin, arbekacin, astromicin, bekanamycin, dibekacin, framycetin, gentamicin, gentamicin-high, habekacin, hygromycin, isepamicin, kanamycin, kanamycin-high, kanamycin/cephalexin, kasugamycin, micronomicin, neomycin, netilmicin, pentisomicin, plazomicin, propikacin, ribostamycin, sisomicin, streptoduocin, streptomycin, streptomycin-high, tobramycin, and tobramycin-high)
- aminopenicillins
  (amoxicillin, amoxicillin/clavulanic acid, and ampicillin)
- antifungals
  (amorolfine, amphotericin B, amphotericin B-high, anidulafungin, butoconazole, caspofungin, ciclopirox, clotrimazole, econazole, fluconazole, flucytosine, fosfluconazole, griseofulvin, hachimycin, ibrexafungerp, isavuconazole, isoconazole, itraconazole, ketoconazole, manogepix, micafungin, miconazole, nystatin, oteseconazole, pimaricin, posaconazole, rezafungin, ribociclib, sulconazole, terbinafine, terconazole, and voriconazole)
- antimycobacterials
  (4-aminosalicylic acid, calcium aminosalicylate, capreomycin, clofazimine, delamanid, enviomycin, ethambutol, ethambutol/isoniazid, ethionamide, isoniazid, isoniazid/sulfamethoxazole/trimethoprim/pyridox morinamide, p-aminosalicylic acid, pretomanid, protionamide, pyrazinamide, rifabutin, rifampicin, rifampicin/ethambutol/isoniazid, rifampicin/isoniazid, rifampicin/pyrazinamide/ethambutol/isoniazid, rifampicin/pyrazinamide/isoniazid, rifamycin, rifapentine, sodium aminosalicylate, streptomycin/isoniazid, terizidone, thioacetazone, thioacetazone/isoniazid, tiocarlide, and viomycin)
- betalactamase_inhibitors
  (amoxicillin/clavulanic acid, amoxicillin/sulbactam, ampicillin/sulbactam, avibactam, aztreonam/avibactam, aztreonam/nacubactam, cefepime/enmetazobactam, cefepime/nacubactam, cefepime/taniborbactam, cefepime/tazobactam, cefepime/zidebactam, cefoperazone/sulbactam, cefotaxime/sulbactam, ceftaroline/avibactam, ceftazidime/avibactam, ceftibuten/avibactam, ceftolozane/tazobactam, imipenem/relebactam, meropenem/nacubactam, meropenem/vaborbactam, mezlocillin/sulbactam, nacubactam, penicillin/sulbactam, piperacillin/sulbactam, piperacillin/tazobactam, sulbactam, sulbactam/durlobactam, sultamicillin, taniborbactam, tazobactam, ticarcillin/clavulanic acid, xeruborbactam, and zidebactam)
- betalactams
  (amoxicillin, amoxicillin/clavulanic acid, amoxicillin/sulbactam, ampicillin, ampicillin/sulbactam, apalcillin, aspoxicillin, azidocillin, azlocillin, aztreonam, aztreonam/avibactam, aztreonam/nacubactam, bacampicillin, benzathine benzylpenicillin, benzathine phenoxymethylpenicillin, benzylpenicillin, benzylpenicillin screening test, biapenem, carbenicillin, carindacillin, carumonam, cefacetrile, cefaclor, cefadroxil, cefalexin, cefaloridine, cefalotin, cefamandole, cefapirin, cefatrizine, cefazedone, cefazolin, cefcapene, cefcapene pivoxil, cefdinir, cefditoren, cefditoren pivoxil, cefepime, cefepime/amikacin, cefepime/clavulanic acid, cefepime/enmetazobactam,

cefepime/nacubactam, cefepime/taniborbactam, cefepime/tazobactam, cefepime/zidebactam, cefetamet, cefetamet pivoxil, cefetecol, cefetrizole, cefiderocol, cefixime, cefmenoxime, cefmetazole, cefodizime, cefonicid, cefoperazone, cefoperazone/sulbactam, ceforanide, cefoselis, cefotaxime, cefotaxime screening test, cefotaxime/clavulanic acid, cefotaxime/sulbactam, cefotetan, cefotiam, cefotiam hexetil, cefovecin, cefoxitin, cefoxitin screening test, cefozopran, cefpimizole, cefpiramide, cefpirome, cefpodoxime, cefpodoxime proxetil, cefpodoxime/clavulanic acid, cefprozil, cefquinome, cefroxadine, cefsulodin, cefsumide, ceftaroline, ceftaroline/avibactam, ceftazidime, ceftazidime/avibactam, ceftazidime/clavulanic acid, cefteram, cefteram pivoxil, ceftezole, ceftibuten, ceftibuten/avibactam, ceftiofur, ceftizoxime, ceftizoxime alapivoxil, ceftobiprole, ceftobiprole medocaril, ceftolozane/tazobactam, ceftriaxone, ceftriaxone/beta-lactamase inhibitor, cefuroxime, cefuroxime axetil, cephradine, ciclacillin, clometocillin, cloxacillin, dicloxacillin, doripenem, epicillin, ertapenem, flucloxacillin, hetacillin, imipenem, imipenem/EDTA, imipenem/relebactam, latamoxef, lenampicillin, loracarbef, mecillinam, meropenem, meropenem/nacubactam, meropenem/vaborbactam, metampicillin, meticillin, mezlocillin, mezlocillin/sulbactam, nafcillin, oxacillin, oxacillin screening test, panipenem, penamecillin, penicillin/novobiocin, penicillin/sulbactam, pheneticillin, phenoxymethylpenicillin, piperacillin, piperacillin/sulbactam, piperacillin/tazobactam, piridicillin, pivampicillin, pivmecillinam, procaine benzylpenicillin, propicillin, razupenem, ritipenem, ritipenem acoxil, sarmoxicillin, sulbenicillin, sultamicillin, talampicillin, taniborbactam, tebipenem, temocillin, ticarcillin, ticarcillin/clavulanic acid, and tigemonam)

- betalactams_with_inhibitor
(amoxicillin/clavulanic acid, amoxicillin/sulbactam, ampicillin/sulbactam, aztreonam/avibactam, aztreonam/nacubactam, cefepime/amikacin, cefepime/clavulanic acid, cefepime/enmetazobactam, cefepime/nacubactam, cefepime/taniborbactam, cefepime/tazobactam, cefepime/zidebactam, cefoperazone/sulbactam, cefotaxime/clavulanic acid, cefotaxime/sulbactam, cefpodoxime/clavulanic acid, ceftaroline/avibactam, ceftazidime/avibactam, ceftazidime/clavulanic acid, ceftibuten/avibactam, ceftolozane/tazobactam, ceftriaxone/beta-lactamase inhibitor, imipenem/relebactam, meropenem/nacubactam, meropenem/vaborbactam, mezlocillin/sulbactam, penicillin/novobiocin, penicillin/sulbactam, piperacillin/sulbactam, piperacillin/tazobactam, and ticarcillin/clavulanic acid)
- carbapenems
(biapenem, doripenem, ertapenem, imipenem, imipenem/EDTA, imipenem/relebactam, meropenem, meropenem/nacubactam, meropenem/vaborbactam, panipenem, razupenem, ritipenem, ritipenem acoxil, taniborbactam, and tebipenem)
- cephalosporins
(cefacetrile, cefaclor, cefadroxil, cefalexin, cefaloridine, cefalotin, cefamandole, cefapirin, cefatrizine, cefazedone, cefazolin, cefcapene, cefcapene pivoxil, cefdinir, cefditoren, cefditoren pivoxil, cefepime, cefepime/amikacin, cefepime/clavulanic acid, cefepime/enmetazobactam, cefepime/nacubactam, cefepime/taniborbactam, cefepime/tazobactam, cefepime/zidebactam, cefetamet, cefetamet pivoxil, cefetecol, cefetrizole, cefiderocol, cefixime, cefmenoxime, cefmetazole, cefodizime, cefonicid, cefoperazone, cefoperazone/sulbactam, ceforanide, cefoselis, cefotaxime, cefotaxime screening test, cefotaxime/clavulanic acid, cefotaxime/sulbactam, cefotetan, cefotiam, cefotiam hexetil, cefovecin, cefoxitin, cefoxitin screening test, cefozopran, cefpimizole, cefpiramide, cefpirome, cefpodoxime, cefpodoxime proxetil, cefpodoxime/clavulanic acid, cefprozil, cefquinome, cefroxadine, cefsulodin, cefsumide, ceftaroline, ceftaroline/avibactam, ceftazidime, ceftazidime/avibactam, ceftazidime/clavulanic acid, cefteram, cefteram pivoxil, ceftezole, ceftibuten, ceftibuten/avibactam, ceftiofur, ceftizoxime, ceftizoxime alapivoxil, ceftobiprole, ceftobiprole medocaril, ceftolozane/tazobactam, ceftriaxone, ceftriaxone/beta-lactamase inhibitor, cefuroxime, cefuroxime axetil, cephradine, lata-

moxef, and loracarbef)

- cephalosporins_1st
(cefacetrile, cefadroxil, cefalexin, cefaloridine, cefalotin, cefapirin, cefatrizine, cefazedone, cefazolin, cefroxadine, ceftezole, and cephradine)
- cephalosporins_2nd
(cefaclor, cefamandole, cefmetazole, cefonicid, ceforanide, cefotetan, cefotiam, cefoxitin, cefoxitin screening test, cefprozil, cefuroxime, cefuroxime axetil, and loracarbef)
- cephalosporins_3rd
(cefcapene, cefcapene pivoxil, cefdinir, cefditoren, cefditoren pivoxil, cefetamet, cefetamet pivoxil, cefixime, cefmenoxime, cefodizime, cefoperazone, cefoperazone/sulbactam, cefotaxime, cefotaxime screening test, cefotaxime/clavulanic acid, cefotaxime/sulbactam, cefotiam hexetil, cefovecin, cefpimizole, cefpiramide, cefpodoxime, cefpodoxime proxetil, cefpodoxime/clavulanic acid, cefsulodin, ceftazidime, ceftazidime/avibactam, ceftazidime/clavulanic acid, cefteram, cefteram pivoxil, ceftibuten, ceftibuten/avibactam, ceftiofur, ceftizoxime, ceftizoxime alapivoxil, ceftriaxone, ceftriaxone/beta-lactamase inhibitor, and latamoxef)
- cephalosporins_4th
(cefepime, cefepime/amikacin, cefepime/clavulanic acid, cefepime/enmetazobactam, cefepime/nacubactam, cefepime/taniborbactam, cefepime/tazobactam, cefepime/zidebactam, cefetecol, cefoselis, cefozopran, cefpirome, and cefquinome)
- cephalosporins_5th
(ceftaroline, ceftaroline/avibactam, ceftobiprole, ceftobiprole medocaril, and ceftolozane/tazobactam)
- cephalosporins_except_caz
(cefacetrile, cefaclor, cefadroxil, cefalexin, cefaloridine, cefalotin, cefamandole, cefapirin, cefatrizine, cefazedone, cefazolin, cefcapene, cefcapene pivoxil, cefdinir, cefditoren, cefditoren pivoxil, cefepime, cefepime/amikacin, cefepime/clavulanic acid, cefepime/enmetazobactam, cefepime/nacubactam, cefepime/taniborbactam, cefepime/tazobactam, cefepime/zidebactam, cefetamet, cefetamet pivoxil, cefetecol, cefetrizole, cefiderocol, cefixime, cefmenoxime, cefmetazole, cefodizime, cefonicid, cefoperazone, cefoperazone/sulbactam, ceforanide, cefoselis, cefotaxime, cefotaxime screening test, cefotaxime/clavulanic acid, cefotaxime/sulbactam, cefotetan, cefotiam, cefotiam hexetil, cefovecin, cefoxitin, cefoxitin screening test, cefozopran, cefpimizole, cefpiramide, cefpirome, cefpodoxime, cefpodoxime proxetil, cefpodoxime/clavulanic acid, cefprozil, cefquinome, cefroxadine, cefsulodin, cefsumide, ceftaroline, ceftaroline/avibactam, ceftazidime/avibactam, ceftazidime/clavulanic acid, cefteram, cefteram pivoxil, ceftezole, ceftibuten, ceftibuten/avibactam, ceftiofur, ceftizoxime, ceftizoxime alapivoxil, ceftobiprole, ceftobiprole medocaril, ceftolozane/tazobactam, ceftriaxone, ceftriaxone/beta-lactamase inhibitor, cefuroxime, cefuroxime axetil, cephradine, latamoxef, and loracarbef)
- fluoroquinolones
(besifloxacin, ciprofloxacin, ciprofloxacin/metronidazole, ciprofloxacin/ornidazole, ciprofloxacin/tinidazole, clinafloxacin, danofloxacin, delafloxacin, difloxacin, enoxacin, enrofloxacin, finafloxacin, fleroxacin, garenoxacin, gatifloxacin, gemifloxacin, grepafloxacin, lascufloxacin, levofloxacin, levofloxacin/ornidazole, levonadifloxacin, lomefloxacin, marbofloxacin, metioxate, miloxacin, moxifloxacin, nadifloxacin, nemonoxacin, nifuroquine, nitroxoline, norfloxacin, norfloxacin screening test, norfloxacin/metronidazole, norfloxacin/tinidazole, ofloxacin, ofloxacin/ornidazole, orbifloxacin, pazufloxacin, pefloxacin, pefloxacin screening test, pradofloxacin, premafloxacin, prulifloxacin, rufloxacin, sarafloxacin, sitafloxacin, sparfloxacin, temafloxacin, tilbroquinol, tioxacin, tosufloxacin, and trovafloxacin)

- fusidanes
  (fusidic acid)
- glycopeptides
  (avoparcin, bleomycin, dalbavancin, norvancomycin, oritavancin, ramoplanin, teicoplanin, teicoplanin-macromethod, telavancin, vancomycin, vancomycin-macromethod, and zorbamycin)
- glycopeptides_except_lipo
  (avoparcin, bleomycin, norvancomycin, ramoplanin, teicoplanin, teicoplanin-macromethod, vancomycin, vancomycin-macromethod, and zorbamycin)
- ionophores
  (lasalocid, monensin sodium, narasin, nystatin, and salinomycin)
- isoxazolylpenicillins
  (cloxacillin, dicloxacillin, flucloxacillin, meticillin, oxacillin, and oxacillin screening test)
- lincosamides
  (clindamycin, clindamycin inducible screening test, lincomycin, and pirlimycin)
- lipoglycopeptides
  (dalbavancin, oritavancin, and telavancin)
- macrolides
  (acetylmidecamycin, acetylspiramycin, azithromycin, clarithromycin, clindamycin inducible screening test, dirithromycin, erythromycin, flurithromycin, gamithromycin, josamycin, kitasamycin, meleumycin, midecamycin, miocamycin, nafithromycin, oleandomycin, pirlimycin, primycin, rokitamycin, roxithromycin, solithromycin, spiramycin, telithromycin, tildipirosin, tilmicosin, troleandomycin, tulathromycin, tylosin, and tylvalosin)
- monobactams
  (aztreonam, aztreonam/avibactam, aztreonam/nacubactam, carumonam, and tigemonam)
- nitrofurans
  (furazidin, furazolidone, nifurtoinol, nitrofurantoin, and nitrofurazone)
- oxazolidinones
  (cadazolid, cycloserine, linezolid, tedizolid, and thiacetazone)
- penicillins
  (amoxicillin, amoxicillin/clavulanic acid, amoxicillin/sulbactam, ampicillin, ampicillin/sulbactam, apalcillin, aspoxicillin, azidocillin, azlocillin, bacampicillin, benzathine benzylpenicillin, benzathine phenoxymethylpenicillin, benzylpenicillin, benzylpenicillin screening test, carbenicillin, carindacillin, ciclacillin, clometocillin, cloxacillin, dicloxacillin, epicillin, flucloxacillin, hetacillin, lenampicillin, mecillinam, metampicillin, meticillin, mezlocillin, mezlocillin/sulbactam, nafcillin, oxacillin, oxacillin screening test, penamecillin, penicillin/novobiocin, penicillin/sulbactam, pheneticillin, phenoxymethylpenicillin, piperacillin, piperacillin/sulbactam, piperacillin/tazobactam, piridicillin, pivampicillin, pivmecillinam, procaine benzylpenicillin, propicillin, sarmoxicillin, sulbenicillin, sultamicillin, talampicillin, temocillin, ticarcillin, and ticarcillin/clavulanic acid)
- peptides
  (avoparcin, bacitracin, bleomycin, dalbavancin, daptomycin, norvancomycin, oritavancin, ramoplanin, teicoplanin, teicoplanin-macromethod, telavancin, thiostrepton, vancomycin, vancomycin-macromethod, and zorbamycin)
- phenicols
  (chloramphenicol, florfenicol, and thiamphenicol)
- phosphonics
  (amikacin/fosfomycin and fosfomycin)

- pleuromutilins
  (retapamulin and tiamulin)
- polymyxins
  (colistin, polymyxin B, and polymyxin B/polysorbate 80)
- quinolones
  (besifloxacin, cinoxacin, ciprofloxacin, ciprofloxacin/metronidazole, ciprofloxacin/ornidazole, ciprofloxacin/tinidazole, clinafloxacin, danofloxacin, delafloxacin, difloxacin, enoxacin, enrofloxacin, finafloxacin, fleroxacin, flumequine, garenoxacin, gatifloxacin, gemifloxacin, grepafloxacin, lascufloxacin, levofloxacin, levofloxacin/ornidazole, levonadifloxacin, lomefloxacin, marbofloxacin, metioxate, miloxacin, moxifloxacin, nadifloxacin, nalidixic acid, nalidixic acid screening test, nemonoxacin, nifuroquine, nitroxoline, norfloxacin, norfloxacin screening test, norfloxacin/metronidazole, norfloxacin/tinidazole, ofloxacin, ofloxacin/ornidazole, orbifloxacin, oxolinic acid, ozenoxacin, pazufloxacin, pefloxacin, pefloxacin screening test, pipemidic acid, piromidic acid, pradofloxacin, premafloxacin, prulifloxacin, rosoxacin, rufloxacin, sarafloxacin, sitafloxacin, sparfloxacin, temafloxacin, tilbroquinol, tioxacin, tosufloxacin, and trovafloxacin)
- rifamycins
  (rifabutin, rifampicin, rifampicin/ethambutol/isoniazid, rifampicin/isoniazid, rifampicin/pyrazinamide/ethambutol/ison
  rifampicin/pyrazinamide/isoniazid, rifamycin, and rifapentine)
- spiropyrimidinetriones
  (zoliflodacin)
- streptogramins
  (ostreogrycin, pristinamycin, quinupristin/dalfopristin, and virginiamycine)
- sulfonamides
  (isoniazid/sulfamethoxazole/trimethoprim/pyridoxine, ormetroprim/sulfamethoxazole, sulfachlorpyridazine, sulfadiazine, sulfadiazine/tetroxoprim, sulfadiazine/trimethoprim, sulfadimethoxine, sulfadimidine, sulfadimidine/trimethoprim, sulfafurazole, sulfaisodimidine, sulfalene, sulfamazone, sulfamerazine, sulfamerazine/trimethoprim, sulfamethazine, sulfamethizole, sulfamethoxazole, sulfamethoxypyridazine, sulfametomidine, sulfametoxydiazine, sulfametrole/trimethoprim, sulfamoxole, sulfamoxole/trimethoprim, sulfanilamide, sulfaperin, sulfaphenazole, sulfapyridine, sulfasuccinamide, sulfathiazole, sulfathiourea, sulfisoxazole, sulfonamide, and trimethoprim/sulfamethoxazole)
- tetracyclines
  (cetocycline, chlortetracycline, clomocycline, demeclocycline, doxycycline, eravacycline, lymecycline, metacycline, minocycline, omadacycline, oxytetracycline, penimepicycline, rolitetracycline, sarecycline, tetracycline, tetracycline screening test, tetracycline/oleandomycin, and tigecycline)
- tetracyclines_except_tgc
  (cetocycline, chlortetracycline, clomocycline, demeclocycline, doxycycline, eravacycline, lymecycline, metacycline, minocycline, omadacycline, oxytetracycline, penimepicycline, rolitetracycline, sarecycline, tetracycline, tetracycline screening test, and tetracycline/oleandomycin)
- trimethoprims
  (brodimoprim, iclaprim, isoniazid/sulfamethoxazole/trimethoprim/pyridoxine, ormetroprim/sulfamethoxazole, sulfadiazine, sulfadiazine/tetroxoprim, sulfadiazine/trimethoprim, sulfadimethoxine, sulfadimidine, sulfadimidine/trimethoprim, sulfafurazole, sulfaisodimidine, sulfalene, sulfamazone, sulfamerazine, sulfamerazine/trimethoprim, sulfamethizole, sulfamethoxazole, sulfamethoxypyridazine, sulfametomidine, sulfametoxydiazine, sulfametrole/trimethoprim, sulfamoxole, sulfamoxole/trimethoprim, sulfanilamide, sulfaperin, sulfaphenazole, sulfapyridine, sulfathiazole, sulfathiourea, trimethoprim, and trimethoprim/sulfamethoxazole)

- ureidopenicillins
  (azlocillin, mezlocillin, piperacillin, and piperacillin/tazobactam)

## Value

A list containing the custom rules

## Examples

```
x <- custom_interpretive_rules(
  AMC == "R" & genus == "Klebsiella" ~ aminopenicillins == "R",
  AMC == "I" & genus == "Klebsiella" ~ aminopenicillins == "I"
)
x

# run the custom rule set (verbose = TRUE will return a logbook instead of the data set):
eucast_rules(example_isolates,
  rules = "custom",
  custom_rules = x,
  info = FALSE,
  overwrite = TRUE,
  verbose = TRUE
)

# combine rule sets
x2 <- c(
  x,
  custom_interpretive_rules(TZP == "R" ~ carbapenems == "R")
)
x2
```

---

# custom_mdro_guideline *Define Custom MDRO Guideline*

---

## Description

Define custom a MDRO guideline for your organisation or specific analysis and use the output of this function in `mdro()`.

## Usage

```
custom_mdro_guideline(..., as_factor = TRUE)

## S3 method for class 'custom_mdro_guideline'
c(x, ..., as_factor = NULL)
```

## Arguments

| | |
|---|---|
| ... | Guideline rules in formula notation, see below for instructions, and in *Examples*. |
| as_factor | A logical to indicate whether the returned value should be an ordered factor (TRUE, default), or otherwise a character vector. For combining rules sets (using c()) this value will be inherited from the first set at default. |
| x | Existing custom MDRO rules |

## Details

Using a custom MDRO guideline is of importance if you have custom rules to determine MDROs in your hospital, e.g., rules that are dependent on ward, state of contact isolation or other variables in your data.

**Basics:**

If you are familiar with the case_when() function of the dplyr package, you will recognise the input method to set your own rules. Rules must be set using what R considers to be the 'formula notation'. The rule itself is written *before* the tilde (~) and the consequence of the rule is written *after* the tilde:

```
custom <- custom_mdro_guideline(CIP == "R" & age > 60 ~ "Elderly Type A",
                                ERY == "R" & age > 60 ~ "Elderly Type B")
```

If a row/an isolate matches the first rule, the value after the first ~ (in this case *'Elderly Type A'*) will be set as MDRO value. Otherwise, the second rule will be tried and so on. The number of rules is unlimited.

You can print the rules set in the console for an overview. Colours will help reading it if your console supports colours.

```
custom
#> A set of custom MDRO rules:
#>   1. If CIP is R and age is higher than 60 then: Elderly Type A
#>   2. If ERY is R and age is higher than 60 then: Elderly Type B
#>   3. Otherwise: Negative

#> Unmatched rows will return NA.
#> Results will be of class 'factor', with ordered levels: Negative < Elderly Type A < Elderly Type B
```

The outcome of the function can be used for the guideline argument in the mdro() function:

```
x <- mdro(example_isolates, guideline = custom)
#> Determining MDROs based on custom rules, resulting in factor levels: Negative < Elderly Type A < Elde
#> - Custom MDRO rule 1: CIP == "R" & age > 60 (198 rows matched)
#> - Custom MDRO rule 2: ERY == "R" & age > 60 (732 rows matched)
#> => Found 930 custom defined MDROs out of 2000 isolates (46.5%)

table(x)
#> x
#>       Negative  Elderly Type A  Elderly Type B
#>           1070             198             732
```

Rules can also be combined with other custom rules by using `c()`:

```
x <- mdro(example_isolates,
          guideline = c(custom,
                custom_mdro_guideline(ERY == "R" & age > 50 ~ "Elderly Type C")))
#> Determining MDROs based on custom rules, resulting in factor levels: Negative < Elderly Type A < Elde
#> - Custom MDRO rule 1: CIP == "R" & age > 60 (198 rows matched)
#> - Custom MDRO rule 2: ERY == "R" & age > 60 (732 rows matched)
#> - Custom MDRO rule 3: ERY == "R" & age > 50 (109 rows matched)
#> => Found 1039 custom defined MDROs out of 2000 isolates (52.0%)

table(x)
#> x
#>       Negative  Elderly Type A  Elderly Type B  Elderly Type C
#>            961             198             732             109
```

**Sharing rules among multiple users:**

The rules set (the `custom` object in this case) could be exported to a shared file location using `saveRDS()` if you collaborate with multiple users. The custom rules set could then be imported using `readRDS()`.

**Usage of multiple antimicrobials and antimicrobial group names:**

You can define antimicrobial groups instead of single antimicrobials for the rule itself, which is the part *before* the tilde (~). Use `any()` or `all()` to specify the scope of the antimicrobial group:

```
custom_mdro_guideline(
  AMX == "R"                         ~ "My MDRO #1",
  any(cephalosporins_2nd() == "R") ~ "My MDRO #2",
  all(glycopeptides() == "R")      ~ "My MDRO #3"
)
```

All 43 antimicrobial selectors are supported for use in the rules:

- `aminoglycosides()` can select:
  amikacin, amikacin/fosfomycin, apramycin, arbekacin, astromicin, bekanamycin, dibekacin, framycetin, gentamicin, gentamicin-high, habekacin, hygromycin, isepamicin, kanamycin, kanamycin-high, kanamycin/cephalexin, kasugamycin, micronomicin, neomycin, netilmicin, pentisomicin, plazomicin, propikacin, ribostamycin, sisomicin, streptoduocin, streptomycin, streptomycin-high, tobramycin, and tobramycin-high
- `aminopenicillins()` can select:
  amoxicillin, amoxicillin/clavulanic acid, and ampicillin
- `antifungals()` can select:
  amorolfine, amphotericin B, amphotericin B-high, anidulafungin, butoconazole, caspofungin, ciclopirox, clotrimazole, econazole, fluconazole, flucytosine, fosfluconazole, griseofulvin, hachimycin, ibrexafungerp, isavuconazole, isoconazole, itraconazole, ketoconazole, manogepix, micafungin, miconazole, nystatin, oteseconazole, pimaricin, posaconazole, rezafungin, ribociclib, sulconazole, terbinafine, terconazole, and voriconazole
- `antimycobacterials()` can select:
  4-aminosalicylic acid, calcium aminosalicylate, capreomycin, clofazimine, delamanid, enviomycin, ethambutol, ethambutol/isoniazid, ethionamide, isoniazid, isoniazid/sulfamethoxazole/trimethoprim/pyrido

morinamide, p-aminosalicylic acid, pretomanid, protionamide, pyrazinamide, rifabutin, rifampicin, rifampicin/ethambutol/isoniazid, rifampicin/isoniazid, rifampicin/pyrazinamide/ethambutol/isoniazid, rifampicin/pyrazinamide/isoniazid, rifamycin, rifapentine, sodium aminosalicylate, streptomycin/isoniazid, terizidone, thioacetazone, thioacetazone/isoniazid, tiocarlide, and viomycin

- `betalactams()` can select:
  amoxicillin, amoxicillin/clavulanic acid, amoxicillin/sulbactam, ampicillin, ampicillin/sulbactam, apalcillin, aspoxicillin, azidocillin, azlocillin, aztreonam, aztreonam/avibactam, aztreonam/nacubactam, bacampicillin, benzathine benzylpenicillin, benzathine phenoxymethylpenicillin, benzylpenicillin, benzylpenicillin screening test, biapenem, carbenicillin, carindacillin, carumonam, cefacetrile, cefaclor, cefadroxil, cefalexin, cefaloridine, cefalotin, cefamandole, cefapirin, cefatrizine, cefazedone, cefazolin, cefcapene, cefcapene pivoxil, cefdinir, cefditoren, cefditoren pivoxil, cefepime, cefepime/amikacin, cefepime/clavulanic acid, cefepime/enmetazobactam, cefepime/nacubactam, cefepime/taniborbactam, cefepime/tazobactam, cefepime/zidebactam, cefetamet, cefetamet pivoxil, cefetecol, cefetrizole, cefiderocol, cefixime, cefmenoxime, cefmetazole, cefodizime, cefonicid, cefoperazone, cefoperazone/sulbactam, ceforanide, cefoselis, cefotaxime, cefotaxime screening test, cefotaxime/clavulanic acid, cefotaxime/sulbactam, cefotetan, cefotiam, cefotiam hexetil, cefovecin, cefoxitin, cefoxitin screening test, cefozopran, cefpimizole, cefpiramide, cefpirome, cefpodoxime, cefpodoxime proxetil, cefpodoxime/clavulanic acid, cefprozil, cefquinome, cefroxadine, cefsulodin, cefsumide, ceftaroline, ceftaroline/avibactam, ceftazidime, ceftazidime/avibactam, ceftazidime/clavulanic acid, cefteram, cefteram pivoxil, ceftezole, ceftibuten, ceftibuten/avibactam, ceftiofur, ceftizoxime, ceftizoxime alapivoxil, ceftobiprole, ceftobiprole medocaril, ceftolozane/tazobactam, ceftriaxone, ceftriaxone/beta-lactamase inhibitor, cefuroxime, cefuroxime axetil, cephradine, ciclacillin, clometocillin, cloxacillin, dicloxacillin, doripenem, epicillin, ertapenem, flucloxacillin, hetacillin, imipenem, imipenem/EDTA, imipenem/relebactam, latamoxef, lenampicillin, loracarbef, mecillinam, meropenem, meropenem/nacubactam, meropenem/vaborbactam, metampicillin, meticillin, mezlocillin, mezlocillin/sulbactam, nafcillin, oxacillin, oxacillin screening test, panipenem, penamecillin, penicillin/novobiocin, penicillin/sulbactam, pheneticillin, phenoxymethylpenicillin, piperacillin, piperacillin/sulbactam, piperacillin/tazobactam, piridicillin, pivampicillin, pivmecillinam, procaine benzylpenicillin, propicillin, razupenem, ritipenem, ritipenem acoxil, sarmoxicillin, sulbenicillin, sultamicillin, talampicillin, taniborbactam, tebipenem, temocillin, ticarcillin, ticarcillin/clavulanic acid, and tigemonam
- `betalactams_with_inhibitor()` can select:
  amoxicillin/clavulanic acid, amoxicillin/sulbactam, ampicillin/sulbactam, aztreonam/avibactam, aztreonam/nacubactam, cefepime/amikacin, cefepime/clavulanic acid, cefepime/enmetazobactam, cefepime/nacubactam, cefepime/taniborbactam, cefepime/tazobactam, cefepime/zidebactam, cefoperazone/sulbactam, cefotaxime/clavulanic acid, cefotaxime/sulbactam, cefpodoxime/clavulanic acid, ceftaroline/avibactam, ceftazidime/avibactam, ceftazidime/clavulanic acid, ceftibuten/avibactam, ceftolozane/tazobactam, ceftriaxone/beta-lactamase inhibitor, imipenem/relebactam, meropenem/nacubactam, meropenem/vaborbactam, mezlocillin/sulbactam, penicillin/novobiocin, penicillin/sulbactam, piperacillin/sulbactam, piperacillin/tazobactam, and ticarcillin/clavulanic acid
- `carbapenems()` can select:
  biapenem, doripenem, ertapenem, imipenem, imipenem/EDTA, imipenem/relebactam, meropenem, meropenem/nacubactam, meropenem/vaborbactam, panipenem, razupenem, ritipenem, ritipenem acoxil, taniborbactam, and tebipenem
- `cephalosporins()` can select:
  cefacetrile, cefaclor, cefadroxil, cefalexin, cefaloridine, cefalotin, cefamandole, cefapirin, cefatrizine, cefazedone, cefazolin, cefcapene, cefcapene pivoxil, cefdinir, cefditoren, cefditoren

pivoxil, cefepime, cefepime/amikacin, cefepime/clavulanic acid, cefepime/enmetazobactam, cefepime/nacubactam, cefepime/taniborbactam, cefepime/tazobactam, cefepime/zidebactam, cefetamet, cefetamet pivoxil, cefetecol, cefetrizole, cefiderocol, cefixime, cefmenoxime, cefmetazole, cefodizime, cefonicid, cefoperazone, cefoperazone/sulbactam, ceforanide, cefoselis, cefotaxime, cefotaxime screening test, cefotaxime/clavulanic acid, cefotaxime/sulbactam, cefotetan, cefotiam, cefotiam hexetil, cefovecin, cefoxitin, cefoxitin screening test, cefozopran, cefpimizole, cefpiramide, cefpirome, cefpodoxime, cefpodoxime proxetil, cefpodoxime/clavulanic acid, cefprozil, cefquinome, cefroxadine, cefsulodin, cefsumide, ceftaroline, ceftaroline/avibactam, ceftazidime, ceftazidime/avibactam, ceftazidime/clavulanic acid, cefteram, cefteram pivoxil, ceftezole, ceftibuten, ceftibuten/avibactam, ceftiofur, ceftizoxime, ceftizoxime alapivoxil, ceftobiprole, ceftobiprole medocaril, ceftolozane/tazobactam, ceftriaxone, ceftriaxone/beta-lactamase inhibitor, cefuroxime, cefuroxime axetil, cephradine, latamoxef, and loracarbef

- `cephalosporins_1st()` can select:
  cefacetrile, cefadroxil, cefalexin, cefaloridine, cefalotin, cefapirin, cefatrizine, cefazedone, cefazolin, cefroxadine, ceftezole, and cephradine
- `cephalosporins_2nd()` can select:
  cefaclor, cefamandole, cefmetazole, cefonicid, ceforanide, cefotetan, cefotiam, cefoxitin, cefoxitin screening test, cefprozil, cefuroxime, cefuroxime axetil, and loracarbef
- `cephalosporins_3rd()` can select:
  cefcapene, cefcapene pivoxil, cefdinir, cefditoren, cefditoren pivoxil, cefetamet, cefetamet pivoxil, cefixime, cefmenoxime, cefodizime, cefoperazone, cefoperazone/sulbactam, cefotaxime, cefotaxime screening test, cefotaxime/clavulanic acid, cefotaxime/sulbactam, cefotiam hexetil, cefovecin, cefpimizole, cefpiramide, cefpodoxime, cefpodoxime proxetil, cefpodoxime/clavulanic acid, cefsulodin, ceftazidime, ceftazidime/avibactam, ceftazidime/clavulanic acid, cefteram, cefteram pivoxil, ceftibuten, ceftibuten/avibactam, ceftiofur, ceftizoxime, ceftizoxime alapivoxil, ceftriaxone, ceftriaxone/beta-lactamase inhibitor, and latamoxef
- `cephalosporins_4th()` can select:
  cefepime, cefepime/amikacin, cefepime/clavulanic acid, cefepime/enmetazobactam, cefepime/nacubactam, cefepime/taniborbactam, cefepime/tazobactam, cefepime/zidebactam, cefetecol, cefoselis, cefozopran, cefpirome, and cefquinome
- `cephalosporins_5th()` can select:
  ceftaroline, ceftaroline/avibactam, ceftobiprole, ceftobiprole medocaril, and ceftolozane/tazobactam
- `fluoroquinolones()` can select:
  besifloxacin, ciprofloxacin, ciprofloxacin/metronidazole, ciprofloxacin/ornidazole, ciprofloxacin/tinidazole, clinafloxacin, danofloxacin, delafloxacin, difloxacin, enoxacin, enrofloxacin, finafloxacin, fleroxacin, garenoxacin, gatifloxacin, gemifloxacin, grepafloxacin, lascufloxacin, levofloxacin, levofloxacin/ornidazole, levonadifloxacin, lomefloxacin, marbofloxacin, metioxate, miloxacin, moxifloxacin, nadifloxacin, nemonoxacin, nifuroquine, nitroxoline, norfloxacin, norfloxacin screening test, norfloxacin/metronidazole, norfloxacin/tinidazole, ofloxacin, ofloxacin/ornidazole, orbifloxacin, pazufloxacin, pefloxacin, pefloxacin screening test, pradofloxacin, premafloxacin, prulifloxacin, rufloxacin, sarafloxacin, sitafloxacin, sparfloxacin, temafloxacin, tilbroquinol, tioxacin, tosufloxacin, and trovafloxacin
- `glycopeptides()` can select:
  avoparcin, bleomycin, dalbavancin, norvancomycin, oritavancin, ramoplanin, teicoplanin, teicoplanin-macromethod, telavancin, vancomycin, vancomycin-macromethod, and zorbamycin
- `ionophores()` can select:
  lasalocid, monensin sodium, narasin, nystatin, and salinomycin

- isoxazolylpenicillins() can select:
  cloxacillin, dicloxacillin, flucloxacillin, meticillin, oxacillin, and oxacillin screening test
- lincosamides() can select:
  clindamycin, clindamycin inducible screening test, lincomycin, and pirlimycin
- lipoglycopeptides() can select:
  dalbavancin, oritavancin, and telavancin
- macrolides() can select:
  acetylmidecamycin, acetylspiramycin, azithromycin, clarithromycin, clindamycin inducible screening test, dirithromycin, erythromycin, flurithromycin, gamithromycin, josamycin, kitasamycin, meleumycin, midecamycin, miocamycin, nafithromycin, oleandomycin, pirlimycin, primycin, rokitamycin, roxithromycin, solithromycin, spiramycin, telithromycin, tildipirosin, tilmicosin, troleandomycin, tulathromycin, tylosin, and tylvalosin
- monobactams() can select:
  aztreonam, aztreonam/avibactam, aztreonam/nacubactam, carumonam, and tigemonam
- nitrofurans() can select:
  furazidin, furazolidone, nifurtoinol, nitrofurantoin, and nitrofurazone
- oxazolidinones() can select:
  cadazolid, cycloserine, linezolid, tedizolid, and thiacetazone
- penicillins() can select:
  amoxicillin, amoxicillin/clavulanic acid, amoxicillin/sulbactam, ampicillin, ampicillin/sulbactam, apalcillin, aspoxicillin, azidocillin, azlocillin, bacampicillin, benzathine benzylpenicillin, benzathine phenoxymethylpenicillin, benzylpenicillin, benzylpenicillin screening test, carbenicillin, carindacillin, ciclacillin, clometocillin, cloxacillin, dicloxacillin, epicillin, flucloxacillin, hetacillin, lenampicillin, mecillinam, metampicillin, meticillin, mezlocillin, mezlocillin/sulbactam, nafcillin, oxacillin, oxacillin screening test, penamecillin, penicillin/novobiocin, penicillin/sulbactam, pheneticillin, phenoxymethylpenicillin, piperacillin, piperacillin/sulbactam, piperacillin/tazobactam, piridicillin, pivampicillin, pivmecillinam, procaine benzylpenicillin, propicillin, sarmoxicillin, sulbenicillin, sultamicillin, talampicillin, temocillin, ticarcillin, and ticarcillin/clavulanic acid
- peptides() can select:
  avoparcin, bacitracin, bleomycin, dalbavancin, daptomycin, norvancomycin, oritavancin, ramoplanin, teicoplanin, teicoplanin-macromethod, telavancin, thiostrepton, vancomycin, vancomycin-macromethod, and zorbamycin
- phenicols() can select:
  chloramphenicol, florfenicol, and thiamphenicol
- phosphonics() can select:
  amikacin/fosfomycin and fosfomycin
- polymyxins() can select:
  colistin, polymyxin B, and polymyxin B/polysorbate 80
- quinolones() can select:
  besifloxacin, cinoxacin, ciprofloxacin, ciprofloxacin/metronidazole, ciprofloxacin/ornidazole, ciprofloxacin/tinidazole, clinafloxacin, danofloxacin, delafloxacin, difloxacin, enoxacin, enrofloxacin, finafloxacin, fleroxacin, flumequine, garenoxacin, gatifloxacin, gemifloxacin, grepafloxacin, lascufloxacin, levofloxacin, levofloxacin/ornidazole, levonadifloxacin, lomefloxacin, marbofloxacin, metioxate, miloxacin, moxifloxacin, nadifloxacin, nalidixic acid, nalidixic acid screening test, nemonoxacin, nifuroquine, nitroxoline, norfloxacin, norfloxacin screening

test, norfloxacin/metronidazole, norfloxacin/tinidazole, ofloxacin, ofloxacin/ornidazole, orbifloxacin, oxolinic acid, ozenoxacin, pazufloxacin, pefloxacin, pefloxacin screening test, pipemidic acid, piromidic acid, pradofloxacin, premafloxacin, prulifloxacin, rosoxacin, rufloxacin, sarafloxacin, sitafloxacin, sparfloxacin, temafloxacin, tilbroquinol, tioxacin, tosufloxacin, and trovafloxacin

- `rifamycins()` can select:
  rifabutin, rifampicin, rifampicin/ethambutol/isoniazid, rifampicin/isoniazid, rifampicin/pyrazinamide/ethambutol/isoni
  rifampicin/pyrazinamide/isoniazid, rifamycin, and rifapentine
- `spiropyrimidinetriones()` can select:
  zoliflodacin
- `streptogramins()` can select:
  ostreogrycin, pristinamycin, quinupristin/dalfopristin, and virginiamycine
- `sulfonamides()` can select:
  isoniazid/sulfamethoxazole/trimethoprim/pyridoxine, ormetroprim/sulfamethoxazole, sulfachlorpyridazine, sulfadiazine, sulfadiazine/tetroxoprim, sulfadiazine/trimethoprim, sulfadimethoxine, sulfadimidine, sulfadimidine/trimethoprim, sulfafurazole, sulfaisodimidine, sulfalene, sulfamazone, sulfamerazine, sulfamerazine/trimethoprim, sulfamethazine, sulfamethizole, sulfamethoxazole, sulfamethoxypyridazine, sulfametomidine, sulfametoxydiazine, sulfametrole/trimethoprim, sulfamoxole, sulfamoxole/trimethoprim, sulfanilamide, sulfaperin, sulfaphenazole, sulfapyridine, sulfasuccinamide, sulfathiazole, sulfathiourea, sulfisoxazole, sulfonamide, and trimethoprim/sulfamethoxazole
- `tetracyclines()` can select:
  cetocycline, chlortetracycline, clomocycline, demeclocycline, doxycycline, eravacycline, lymecycline, metacycline, minocycline, omadacycline, oxytetracycline, penimepicycline, rolitetracycline, sarecycline, tetracycline, tetracycline screening test, tetracycline/oleandomycin, and tigecycline
- `trimethoprims()` can select:
  brodimoprim, iclaprim, isoniazid/sulfamethoxazole/trimethoprim/pyridoxine, ormetroprim/sulfamethoxazole, sulfadiazine, sulfadiazine/tetroxoprim, sulfadiazine/trimethoprim, sulfadimethoxine, sulfadimidine, sulfadimidine/trimethoprim, sulfafurazole, sulfaisodimidine, sulfalene, sulfamazone, sulfamerazine, sulfamerazine/trimethoprim, sulfamethizole, sulfamethoxazole, sulfamethoxypyridazine, sulfametomidine, sulfametoxydiazine, sulfametrole/trimethoprim, sulfamoxole, sulfamoxole/trimethoprim, sulfanilamide, sulfaperin, sulfaphenazole, sulfapyridine, sulfathiazole, sulfathiourea, trimethoprim, and trimethoprim/sulfamethoxazole
- `ureidopenicillins()` can select:
  azlocillin, mezlocillin, piperacillin, and piperacillin/tazobactam

## Value

A list containing the custom rules

## Examples

```
x <- custom_mdro_guideline(
  CIP == "R" & age > 60 ~ "Elderly Type A",
  ERY == "R" & age > 60 ~ "Elderly Type B"
)
x
```

```
# run the custom rule set (verbose = TRUE will return a logbook instead of the data set):
out <- mdro(example_isolates, guideline = x)
table(out)

out <- mdro(example_isolates, guideline = x, verbose = TRUE)
head(out)

# you can create custom guidelines using selectors (see ?antimicrobial_selectors)
my_guideline <- custom_mdro_guideline(
  AMX == "R" ~ "Custom MDRO 1",
  all(cephalosporins_2nd() == "R") ~ "Custom MDRO 2"
)
my_guideline

out <- mdro(example_isolates, guideline = my_guideline)
table(out)
```

---

dosage *Data Set with Treatment Dosages as Defined by EUCAST*

---

## Description

EUCAST breakpoints used in this package are based on the dosages in this data set. They can be retrieved with `eucast_dosage()`.

## Usage

```
dosage
```

## Format

A tibble with 759 observations and 9 variables:

- `ab`
  Antimicrobial ID as used in this package (such as `AMC`), using the official EARS-Net (European Antimicrobial Resistance Surveillance Network) codes where available
- `name`
  Official name of the antimicrobial drug as used by WHONET/EARS-Net or the WHO
- `type`
  Type of the dosage, either `"high_dosage"`, `"standard_dosage"`, or `"uncomplicated_uti"`
- `dose`
  Dose, such as "2 g" or "25 mg/kg"
- `dose_times`
  Number of times a dose must be administered
- `administration`
  Route of administration, either `""`, `"im"`, `"iv"`, `"oral"`, or NA

- `notes`
  Additional dosage notes
- `original_txt`
  Original text in the PDF file of EUCAST
- `eucast_version`
  Version number of the EUCAST Clinical Breakpoints guideline to which these dosages apply, either 15, 14, 13.1, 12, or 11

### Download Our Reference Data

All reference data sets in the AMR package - including information on microorganisms, antimicrobials, and clinical breakpoints - are freely available for download in multiple formats: R, MS Excel, Apache Feather, Apache Parquet, SPSS, and Stata.

For maximum compatibility, we also provide machine-readable, tab-separated plain text files suitable for use in any software, including laboratory information systems.

Visit our website for direct download links, or explore the actual files in our GitHub repository.

### Examples

```
dosage
```

---

## `esbl_isolates` *Data Set with 500 ESBL Isolates*

---

### Description

A data set containing 500 microbial isolates with MIC values of common antibiotics and a binary `esbl` column for extended-spectrum beta-lactamase (ESBL) production. This data set contains randomised fictitious data but reflects reality and can be used to practise AMR-related machine learning, e.g., classification modelling with tidymodels.

### Usage

```
esbl_isolates
```

### Format

A tibble with 500 observations and 19 variables:

- `esbl`
  Logical indicator if the isolate is ESBL-producing
- `genus`
  Genus of the microorganism
- `AMC:COL`
  MIC values for 17 antimicrobial drugs, transformed to class `mic` (see `as.mic()`)

### Details

See our tidymodels integration for an example using this data set.

### Examples

```
esbl_isolates
```

---

`example_isolates` *Data Set with 2 000 Example Isolates*

---

### Description

A data set containing 2 000 microbial isolates with their full antibiograms. This data set contains randomised fictitious data, but reflects reality and can be used to practise AMR data analysis. For examples, please read the tutorial on our website.

### Usage

```
example_isolates
```

### Format

A tibble with 2 000 observations and 46 variables:

- `date`
  Date of receipt at the laboratory
- `patient`
  ID of the patient
- `age`
  Age of the patient
- `gender`
  Gender of the patient, either `"F"` or `"M"`
- `ward`
  Ward type where the patient was admitted, either `"Clinical"`, `"ICU"`, or `"Outpatient"`
- `mo`
  ID of microorganism created with `as.mo()`, see also the microorganisms data set
- `PEN:RIF`
  40 different antimicrobials with class `sir` (see `as.sir()`); these column names occur in the antimicrobials data set and can be translated with `set_ab_names()` or `ab_name()`

### Download Our Reference Data

All reference data sets in the AMR package - including information on microorganisms, antimicrobials, and clinical breakpoints - are freely available for download in multiple formats: R, MS Excel, Apache Feather, Apache Parquet, SPSS, and Stata.

For maximum compatibility, we also provide machine-readable, tab-separated plain text files suitable for use in any software, including laboratory information systems.

Visit our website for direct download links, or explore the actual files in our GitHub repository.

## Examples

```
example_isolates
```

---

`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 data analysis. This data set can be used for practice.

## Usage

```
example_isolates_unclean
```

## Format

A tibble 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 antimicrobials that have to be transformed with `as.sir()`

## Download Our Reference Data

All reference data sets in the AMR package - including information on microorganisms, antimicrobials, and clinical breakpoints - are freely available for download in multiple formats: R, MS Excel, Apache Feather, Apache Parquet, SPSS, and Stata.

For maximum compatibility, we also provide machine-readable, tab-separated plain text files suitable for use in any software, including laboratory information systems.

Visit our website for direct download links, or explore the actual files in our GitHub repository.

## Examples

```
example_isolates_unclean
```

`first_isolate` *Determine First Isolates*

## Description

Determine first isolates of all microorganisms of every patient per episode and (if needed) per specimen type. These functions support all four methods as summarised by Hindler *et al.* in 2007 (doi:10.1086/511864). To determine patient episodes not necessarily based on microorganisms, use `is_new_episode()` that also supports grouping with the `dplyr` package.

## Usage

```
first_isolate(
  x = NULL,
  col_date = NULL,
  col_patient_id = NULL,
  col_mo = NULL,
  col_testcode = NULL,
  col_specimen = NULL,
  col_icu = NULL,
  col_keyantimicrobials = NULL,
  episode_days = 365,
  testcodes_exclude = NULL,
  icu_exclude = FALSE,
  specimen_group = NULL,
  type = "points",
 method = c("phenotype-based", "episode-based", "patient-based", "isolate-based"),
  ignore_I = TRUE,
  points_threshold = 2,
  info = interactive(),
  include_unknown = FALSE,
  include_untested_sir = TRUE,
  ...
)

filter_first_isolate(
  x = NULL,
  col_date = NULL,
  col_patient_id = NULL,
  col_mo = NULL,
  episode_days = 365,
 method = c("phenotype-based", "episode-based", "patient-based", "isolate-based"),
  ...
)
```

## Arguments

| | |
|---|---|
| x | A data.frame containing isolates. Can be left blank for automatic determination, see *Examples*. |
| col_date | Column name of the result date (or date that is was received on the lab) - the default is the first column with a date class. |
| col_patient_id | Column name of the unique IDs of the patients - the default is the first column that starts with 'patient' or 'patid' (case insensitive). |
| col_mo | Column name of the names or codes of the microorganisms (see as.mo()) - the default is 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 (such as 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). This can also be a logical vector with the same length as rows in x. |
| col_keyantimicrobials | (only useful when method = "phenotype-based") column name of the key antimicrobials to determine first isolates, see key_antimicrobials(). The default is the first column that starts with 'key' followed by 'ab' or 'antibiotics' or 'antimicrobials' (case insensitive). Use col_keyantimicrobials = FALSE to prevent this. Can also be the output of key_antimicrobials(). |
| 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 | A character vector with test codes that should be excluded (case-insensitive). |
| icu_exclude | A logical to indicate whether ICU isolates should be excluded (rows with value TRUE in the column set with col_icu). |
| specimen_group | Value in the column set with col_specimen to filter on. |
| type | Type to determine weighed isolates; can be "keyantimicrobials" or "points", see *Details*. |
| method | The method to apply, either "phenotype-based", "episode-based", "patient-based" or "isolate-based" (can be abbreviated), see *Details*. The default is "phenotype-based" if antimicrobial test results are present in the data, and "episode-based" otherwise. |
| ignore_I | logical to indicate whether antibiotic interpretations with "I" will be ignored when type = "keyantimicrobials", see *Details*. |
| points_threshold | Minimum number of points to require before differences in the antibiogram will lead to inclusion of an isolate when type = "points", see *Details*. |
| info | A logical to indicate info should be printed - the default is TRUE only in interactive mode. |

include_unknown
: A logical to indicate 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.

include_untested_sir
: A logical to indicate whether also rows without antibiotic results are still eligible for becoming a first isolate. Use include_untested_sir = FALSE to always return FALSE for such rows. This checks the data set for columns of class sir and consequently requires transforming columns with antibiotic results using as.sir() first.

...
: Arguments passed on to first_isolate() when using filter_first_isolate(), otherwise arguments passed on to key_antimicrobials() (such as universal, gram_negative, gram_positive).

## Details

The methodology implemented in these functions is strictly based on the recommendations outlined in CLSI Guideline M39 and the research overview by Hindler *et al.* (2007, doi:10.1086/511864).

To conduct epidemiological analyses on antimicrobial resistance data, only so-called first isolates should be included to prevent overestimation and underestimation of antimicrobial resistance. Different methods can be used to do so, see below.

These functions are context-aware. This means that the x argument can be left blank if used inside a data.frame call, see *Examples*.

The first_isolate() function is a wrapper around the is_new_episode() function, but more efficient for data sets containing microorganism codes or names.

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

### Different methods:

According to previously-mentioned sources, there are different methods (algorithms) to select first isolates with increasing reliability: isolate-based, patient-based, episode-based and phenotype-based. All methods select on a combination of the taxonomic genus and species (not subspecies).

All mentioned methods are covered in the first_isolate() function:

| Method | Function to apply |
|---|---|
| **Isolate-based**<br>*(= all isolates)* | first_isolate(x, method = "isolate-based") |
| **Patient-based**<br>*(= first isolate per patient)* | first_isolate(x, method = "patient-based") |
| **Episode-based**<br>*(= first isolate per episode)*<br>- 7-Day interval from initial isolate | first_isolate(x, method = "episode-based"), or:<br>- first_isolate(x, method = "e", episode_days = 7) |

| | |
|---|---|
| - 30-Day interval from initial isolate | - `first_isolate(x, method = "e", episode_days = 30)` |
| **Phenotype-based** | `first_isolate(x, method = "phenotype-based")`, or: |
| *(= first isolate per phenotype)* | |
| - Major difference in any antimicrobial result | - `first_isolate(x, type = "points")` |
| - Any difference in key antimicrobial results | - `first_isolate(x, type = "keyantimicrobials")` |

**Isolate-based**

*Minimum variables required: Microorganism identifier*

This method does not require any selection, as all isolates should be included. It does, however, respect all arguments set in the `first_isolate()` function. For example, the default setting for `include_unknown` (`FALSE`) will omit selection of rows without a microbial ID.

**Patient-based**

*Minimum variables required: Microorganism identifier, Patient identifier*

This method includes every genus-species combination per patient once. This method makes sure that no duplicate isolates are selected from the same patient. This method is preferred to e.g. identify the first MRSA finding of each patient to determine the incidence. Conversely, in a large longitudinal data set, this could mean that isolates are *excluded* that were found years after the initial isolate.

**Episode-based**

*Minimum variables required: Microorganism identifier, Patient identifier, Date*

To include every genus-species combination per patient episode once, set the `episode_days` to a sensible number of days. Depending on the type of analysis, this could be e.g., 14, 30, 60 or 365. Short episodes are common for analysing specific hospital or ward data or ICU cases, long episodes are common for analysing regional and national data.

This is the most common method to correct for duplicate isolates. Patients are categorised into episodes based on their ID and dates (e.g., the date of specimen receipt or laboratory result). While this is a common method, it does not take into account antimicrobial test results. This means that e.g. a methicillin-resistant *Staphylococcus aureus* (MRSA) isolate cannot be differentiated from a wildtype *Staphylococcus aureus* isolate.

**Phenotype-based**

*Minimum variables required: Microorganism identifier, Patient identifier, Date, Antimicrobial test results*

This is a more reliable method, since it also *weighs* the antibiogram (antimicrobial test results) yielding so-called 'first weighted isolates'. There are two different methods to weigh the antibiogram:

1. Using `type = "points"` and argument `points_threshold` (default)
   This method weighs *all* antimicrobial drugs available in the data set. Any difference from I to S or R (or vice versa) counts as `0.5` points, a difference from S to R (or vice versa) counts as `1` point. When the sum of points exceeds `points_threshold`, which defaults to `2`, an isolate will be selected as a first weighted isolate.
   All antimicrobials are internally selected using the `all_antimicrobials()` function. The output of this function does not need to be passed to the `first_isolate()` function.

2. Using `type = "keyantimicrobials"` and argument `ignore_I`
   This method only weighs specific antimicrobial drugs, called *key antimicrobials*. Any difference from S to R (or vice versa) in these key antimicrobials will select an isolate as a first weighted isolate. With `ignore_I = FALSE`, also differences from I to S or R (or vice versa) will lead to this.
   Key antimicrobials are internally selected using the `key_antimicrobials()` function, but can also be added manually as a variable to the data and set in the `col_keyantimicrobials` argument. Another option is to pass the output of the `key_antimicrobials()` function directly to the `col_keyantimicrobials` argument.

The default method is phenotype-based (using `type = "points"`) and episode-based (using `episode_days = 365`). This makes sure that every genus-species combination is selected per patient once per year, while taking into account all antimicrobial test results. If no antimicrobial test results are available in the data set, only the episode-based method is applied at default.

## Value

A logical vector

## References

Methodology of these functions is strictly based on:


- **M39 Analysis and Presentation of Cumulative Antimicrobial Susceptibility Test Data, 5th Edition**, 2022, *Clinical and Laboratory Standards Institute (CLSI)*. https://clsi.org/standards/products/microbiology/documents/m39/.
- Hindler JF and Stelling J (2007). **Analysis and Presentation of Cumulative Antibiograms: A New Consensus Guideline from the Clinical and Laboratory Standards Institute.** Clinical Infectious Diseases, 44(6), 867-873. doi:10.1086/511864


## See Also

`key_antimicrobials()`

## Examples

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

example_isolates[first_isolate(info = TRUE), ]

# get all first Gram-negatives
example_isolates[which(first_isolate(info = FALSE) & mo_is_gram_negative()), ]

if (require("dplyr")) {
  # filter on first isolates using dplyr:
  example_isolates %>%
    filter(first_isolate(info = TRUE))
}
if (require("dplyr")) {
  # short-hand version:
```

```
  example_isolates %>%
    filter_first_isolate(info = FALSE)
}
if (require("dplyr")) {
  # flag the first isolates per group:
  example_isolates %>%
    group_by(ward) %>%
    mutate(first = first_isolate(info = TRUE)) %>%
    select(ward, date, patient, mo, first)
}
```

---

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

---

## Description

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

## Usage

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

## Arguments

| | |
|---|---|
| `x` | a numeric vector or matrix. `x` and `y` can also both be factors. |
| `y` | a numeric vector; ignored if `x` is a matrix. If `x` is a factor, `y` should be a factor of the same length. |
| `p` | a vector of probabilities of the same length as `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 (such as 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:

$G = 2 * sum(x * log(x/E))$

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

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

where df are the degrees of freedom.

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

## Value

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

- statistic: the value of the 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 was used
- data.name: a character string giving the name(s) of the data
- observed: the observed counts
- expected: the expected counts under the null hypothesis
- residuals: the Pearson residuals, (observed - expected) / sqrt(expected)
- stdres: standardised residuals, (observed - expected) / sqrt(V), where V is the residual cell variance (Agresti 2007, section 2.4.5) for the case where x is a matrix, n * p * (1 - p) otherwise

## Source

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

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

## References

1. McDonald, J.H. 2014. **Handbook of Biological Statistics (3rd ed.)**. Sparky House Publishing, Baltimore, Maryland.
2. Agresti, A. 2007. **An Introduction to Categorical Data Analysis (2nd ed.)**. Wiley, New York.

## See Also

chisq.test()

## Examples

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

x <- c(772, 1611, 737)
g.test(x, p = c(1, 2, 1) / 4)

# 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)

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

---

# get_episode *Determine Clinical or Epidemic Episodes*

---

## Description

These functions determine which items in a vector can be considered (the start of) a new episode. This can be used to determine clinical episodes for any epidemiological analysis. The `get_episode()` function returns the index number of the episode per group, while the `is_new_episode()` function returns TRUE for every new `get_episode()` index. Both absolute and relative episode determination are supported.

## Usage

```
get_episode(x, episode_days = NULL, case_free_days = NULL, ...)

is_new_episode(x, episode_days = NULL, case_free_days = NULL, ...)
```

## Arguments

| | |
|---|---|
| `x` | Vector of dates (class `Date` or `POSIXt`), will be sorted internally to determine episodes. |
| `episode_days` | Episode length in days to specify the time period after which a new episode begins, can also be less than a day or `Inf`, see *Details*. |
| `case_free_days` | (inter-epidemic) interval length in days after which a new episode will start, can also be less than a day or `Inf`, see *Details*. |
| `...` | Ignored, only in place to allow future extensions. |

## Details

Episodes can be determined in two ways: absolute and relative.

1. Absolute

   This method uses `episode_days` to define an episode length in days, after which a new episode will start. A common use case in AMR data analysis is microbial epidemiology: episodes of *S. aureus* bacteraemia in ICU patients for example. The episode length could then be 30 days, so that new *S. aureus* isolates after an ICU episode of 30 days will be considered a different (or new) episode.

   Thus, this method counts **since the start of the previous episode**.

2. Relative

   This method uses `case_free_days` to quantify the duration of case-free days (the inter-epidemic interval), after which a new episode will start. A common use case is infectious disease epidemiology: episodes of norovirus outbreaks in a hospital for example. The case-free period could then be 14 days, so that new norovirus cases after that time will be considered a different (or new) episode.

   Thus, this methods counts **since the last case in the previous episode**.

In a table:

| Date | Using `episode_days = 7` | Using `case_free_days = 7` |
|---|---|---|
| 2023-01-01 | 1 | 1 |
| 2023-01-02 | 1 | 1 |
| 2023-01-05 | 1 | 1 |
| 2023-01-08 | 2** | 1 |
| 2023-02-21 | 3 | 2*** |
| 2023-02-22 | 3 | 2 |
| 2023-02-23 | 3 | 2 |
| 2023-02-24 | 3 | 2 |
| 2023-03-01 | 4 | 2 |

** This marks the start of a new episode, because 8 January 2023 is more than 7 days since the start of the previous episode (1 January 2023).
*** This marks the start of a new episode, because 21 January 2023 is more than 7 days since the last case in the previous episode (8 January 2023).

Either `episode_days` or `case_free_days` must be provided in the function.

**Difference between `get_episode()` and `is_new_episode()`:**

The `get_episode()` function returns the index number of the episode, so all cases/patients/isolates in the first episode will have the number 1, all cases/patients/isolates in the second episode will have the number 2, etc.

The `is_new_episode()` function on the other hand, returns `TRUE` for every new `get_episode()` index.

To specify, when setting `episode_days = 365` (using method 1 as explained above), this is how the two functions differ:

| patient | date | get_episode() | is_new_episode() |
|---|---|---|---|
| A | 2019-01-01 | 1 | TRUE |
| A | 2019-03-01 | 1 | FALSE |
| A | 2021-01-01 | 2 | TRUE |
| B | 2008-01-01 | 1 | TRUE |
| B | 2008-01-01 | 1 | FALSE |
| C | 2020-01-01 | 1 | TRUE |

**Other:**

The `first_isolate()` function is a wrapper around the `is_new_episode()` function, but is more efficient for data sets containing microorganism codes or names and allows for different isolate selection methods.

The `dplyr` package is not required for these functions to work, but these episode functions do support variable grouping and work conveniently inside `dplyr` verbs such as `filter()`, `mutate()` and `summarise()`.

## Value

- `get_episode()`: an integer vector
- `is_new_episode()`: a logical vector

## See Also

`first_isolate()`

## Examples

```
# difference between absolute and relative determination of episodes:
x <- data.frame(dates = as.Date(c(
  "2021-01-01",
  "2021-01-02",
  "2021-01-05",
  "2021-01-08",
  "2021-02-21",
  "2021-02-22",
  "2021-02-23",
  "2021-02-24",
  "2021-03-01",
  "2021-03-01"
```

```
)))
x$absolute <- get_episode(x$dates, episode_days = 7)
x$relative <- get_episode(x$dates, case_free_days = 7)
x


# `example_isolates` is a data set available in the AMR package.
# See ?example_isolates
df <- example_isolates[sample(seq_len(2000), size = 100), ]

get_episode(df$date, episode_days = 60) # indices
is_new_episode(df$date, episode_days = 60) # TRUE/FALSE

# filter on results from the third 60-day episode only, using base R
df[which(get_episode(df$date, 60) == 3), ]

# the functions also work for less than a day, e.g. to include one per hour:
get_episode(
  c(
    Sys.time(),
    Sys.time() + 60 * 60
  ),
  episode_days = 1 / 24
)


if (require("dplyr")) {
  # is_new_episode() can also be used in dplyr verbs to determine patient
  # episodes based on any (combination of) grouping variables:
  df %>%
    mutate(condition = sample(
      x = c("A", "B", "C"),
      size = 100,
      replace = TRUE
    )) %>%
    group_by(patient, condition) %>%
    mutate(new_episode = is_new_episode(date, 365)) %>%
    select(patient, date, condition, new_episode) %>%
    arrange(patient, condition, date)
}

if (require("dplyr")) {
  df %>%
    group_by(ward, patient) %>%
    transmute(date,
      patient,
      new_index = get_episode(date, 60),
      new_logical = is_new_episode(date, 60)
    ) %>%
    arrange(patient, ward, date)
}

if (require("dplyr")) {
```

```
  df %>%
    group_by(ward) %>%
    summarise(
      n_patients = n_distinct(patient),
      n_episodes_365 = sum(is_new_episode(date, episode_days = 365)),
      n_episodes_60 = sum(is_new_episode(date, episode_days = 60)),
      n_episodes_30 = sum(is_new_episode(date, episode_days = 30))
    )
}

# grouping on patients and microorganisms leads to the same
# results as first_isolate() when using 'episode-based':
if (require("dplyr")) {
  x <- df %>%
    filter_first_isolate(
      include_unknown = TRUE,
      method = "episode-based"
    )

  y <- df %>%
    group_by(patient, mo) %>%
    filter(is_new_episode(date, 365)) %>%
    ungroup()

  identical(x, y)
}

# but is_new_episode() has a lot more flexibility than first_isolate(),
# since you can now group on anything that seems relevant:
if (require("dplyr")) {
  df %>%
    group_by(patient, mo, ward) %>%
    mutate(flag_episode = is_new_episode(date, 365)) %>%
    select(group_vars(.), flag_episode)
}
```

---

## 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

```
ggplot_pca(
  x,
  choices = 1:2,
```

```
  scale = 1,
  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_textangled = TRUE,
  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 ^ scale and the observations are scaled by lambda ^ (1-scale) where lambda are the singular values as computed by princomp. Normally 0 <= scale <= 1, and a warning will be issued if the specified scale is outside this range. |
| 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. |
| labels | An optional vector of labels for the observations. If set, the labels will be placed below their respective points. When using the pca() function as input for x, this will be determined automatically based on the attribute non_numeric_cols, see pca(). |
| labels_textsize | The size of the text used for the labels. |
| labels_text_placement | Adjustment factor the placement of the variable names (>=1 means further away from the arrow head). |
| groups | An optional vector of groups for the labels, with the same length as labels. If set, the points and labels will be coloured according to these groups. When using the pca() function as input for x, this will be determined automatically based on the attribute non_numeric_cols, see pca(). |

| | |
|---|---|
| ellipse | A logical to indicate whether a normal data ellipse should be drawn for each group (set with groups). |
| ellipse_prob | Statistical size of the ellipse in normal probability. |
| ellipse_size | The size of the ellipse line. |
| ellipse_alpha | The alpha (transparency) of the ellipse line. |
| points_size | The size of the points. |
| points_alpha | The alpha (transparency) of the points. |
| arrows | A logical to indicate whether arrows should be drawn. |
| arrows_colour | The colour of the arrow and their text. |
| arrows_size | The size (thickness) of the arrow lines. |
| arrows_textsize | The size of the text at the end of the arrows. |
| arrows_textangled | A logical whether the text at the end of the arrows should be angled. |
| arrows_alpha | The alpha (transparency) of the arrows and their text. |
| base_textsize | The text size for all plot elements except the labels and arrows. |
| ... | Arguments passed on to functions. |

## Details

The colours for labels and points can be changed by adding another scale layer for colour, such as scale_colour_viridis_d() and scale_colour_brewer().

## 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. Hardened all input possibilities by defining the exact type of user input for every argument
4. Added total amount of explained variance as a caption in the plot
5. Cleaned all syntax based on the lintr package, fixed grammatical errors and added integrity checks
6. Updated documentation

## References

Gabriel, K.R. 1971. **The biplot graphical display of matrices with application to principal component analysis**. *Biometrika* 58(3):453-467. doi:10.1093/biomet/58.3.453

## Examples

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


if (require("dplyr")) {
  # calculate the resistance per group first
  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;
    filter(n() >= 30) %>% # filter on only 30 results per group
    summarise_if(is.sir, resistance) # then get resistance of all drugs

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

  summary(pca_result)

  # old base R plotting method:
  biplot(pca_result, main = "Base R biplot")

  # new ggplot2 plotting method using this package:
  if (require("ggplot2")) {
    ggplot_pca(pca_result) +
      labs(title = "ggplot2 biplot")
  }
  if (require("ggplot2")) {
    # still extendible with any ggplot2 function
    ggplot_pca(pca_result) +
      scale_colour_viridis_d() +
      labs(title = "ggplot2 biplot")
  }
}
```

---

# ggplot_sir *AMR Plots with* ggplot2

---

## Description

Use these functions to create bar plots for AMR data analysis. All functions rely on ggplot2 functions.

## Usage

```
ggplot_sir(
```

```
  data,
  position = NULL,
  x = "antibiotic",
  fill = "interpretation",
  facet = NULL,
  breaks = seq(0, 1, 0.1),
  limits = NULL,
  translate_ab = "name",
  combine_SI,
  minimum = 30,
  language = get_AMR_locale(),
  guideline = getOption("AMR_guideline", "EUCAST"),
  nrow = NULL,
  colours = c(S = "#3CAEA3", SDD = "#8FD6C4", SI = "#3CAEA3", I = "#F6D55C", IR =
    "#ED553B", R = "#ED553B"),
  datalabels = TRUE,
  datalabels.size = 2.5,
  datalabels.colour = "grey15",
  title = NULL,
  subtitle = NULL,
  caption = NULL,
  x.title = "Antimicrobial",
  y.title = "Proportion",
  ...
)

geom_sir(
  position = NULL,
  x = c("antibiotic", "interpretation"),
  fill = "interpretation",
  translate_ab = "name",
  minimum = 30,
  language = get_AMR_locale(),
  combine_SI,
  guideline = getOption("AMR_guideline", "EUCAST"),
  ...
)

labels_sir_count(
  position = NULL,
  x = "antibiotic",
  translate_ab = "name",
  minimum = 30,
  language = get_AMR_locale(),
  combine_SI,
  guideline = getOption("AMR_guideline", "EUCAST"),
  datalabels.size = 3,
  datalabels.colour = "grey15"
```

)

## Arguments

| | |
|---|---|
| data | A data.frame with column(s) of class sir (see as.sir()). |
| 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 | A numeric vector of positions. |
| limits | A 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 antimicrobials data set to translate the antibiotic abbreviations to, using ab_property(). |
| combine_SI | *(deprecated in favour of* guideline*)* A logical. If set, it takes precedence over guideline: TRUE merges S, SDD, and I into "SI" vs. "R", and FALSE keeps S, I, and R separate. |
| 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*. |
| language | Language of the returned text - the default is the current system language (see get_AMR_locale()) and can also be set with the package option AMR_locale. Use language = NULL or language = "" to prevent translation. |
| guideline | A guideline name, "EUCAST" or "CLSI" (or any version thereof, e.g. "CLSI 2025"), to determine whether the 'I' category is considered susceptible (EUCAST) or resistant (CLSI). The default is the package option AMR_guideline, or "EUCAST" if it is not set. For functions that return results per category, i.e., count_df(), proportion_df(), sir_df(), ggplot_sir(), geom_sir(), and labels_sir_count(), this can also be NULL or FALSE to keep S, I, and R separate. See section *Interpretation of SIR* below. |
| nrow | (when using facet) number of rows. |
| colours | A named vactor with colour to be used for filling. The default colours are colour-blind friendly. |
| datalabels | Show datalabels using labels_sir_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 arguments passed on to geom_sir() or, in case of scale_sir_colours(), named values to set colours. The default colours are colour-blind friendly, while maintaining the convention that e.g. 'susceptible' should be green and 'resistant' should be red. See *Examples*. |

## Details

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

geom_sir() will take any variable from the data that has an sir class (created with as.sir()) using sir_df() and will plot bars with the percentage S, I, and R. The default behaviour is to have the bars stacked and to have the different antimicrobials on the x axis.

Additional functions include:

- facet_sir() 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_y_continuous().
- scale_sir_colours() sets colours to the bars (green for S, yellow for I, and red for R). with multilingual support. The default colours are colour-blind friendly, while maintaining the convention that e.g. 'susceptible' should be green and 'resistant' should be red.
- theme_sir() is a ggplot2 theme with minimal distraction.
- labels_sir_count() print datalabels on the bars with percentage and amount of isolates using ggplot2::geom_text().

ggplot_sir() 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*.

## Interpretation of SIR

In 2019, the European Committee on Antimicrobial Susceptibility Testing (EUCAST) has decided to change the definitions of susceptibility testing categories S, I, and R (https://www.eucast.org/bacteria/clinical-breakpoints-and-interpretation/definition-of-s-i-and-r/).

In aggregated analyses, such as proportions, counts, antibiograms, and plots, the 'I' category must be considered either susceptible or resistant. In epidemiological AMR data analysis, this follows the guideline: EUCAST considers 'I' susceptible, while CLSI considers 'I' resistant. The dose-dependent susceptibility of CLSI ('SDD') is always considered susceptible, as it equals what EUCAST defines as 'I'. All functions with a guideline argument for this purpose use these categories:

| guideline | Susceptible | Resistant |
|---|---|---|
| "EUCAST" | S, SDD, I, WT | R, NWT, NS |
| "CLSI" | S, SDD, WT | I, R, NWT, NS |

The values 'WT' (wild type) and 'NWT' (non-wild type) result from epidemiological cut-off values (ECOFFs) instead of clinical breakpoints. Since they are no clinical categories, 'WT' is always

considered susceptible and 'NWT' always resistant, regardless of the guideline. The value 'NS' (non-susceptible) is always considered resistant, and 'NI' (non-interpretable) is always considered a missing result.

The `guideline` argument can be:

- **Left unset**, to use the package option `AMR_guideline`, or `"EUCAST"` if this option is not set. In the latter case, a message is shown once per session (but not in R Markdown or Quarto documents); set the option to silence it.
- `"EUCAST"`, or any EUCAST version such as `"EUCAST 2025"`, to consider 'I' susceptible, e.g., to return `"SI"` vs. `"R"` in `proportion_df()`.
- `"CLSI"`, or any CLSI version such as `"CLSI 2025"`, to consider 'I' resistant, e.g., to return `"S"` vs. `"IR"` in `proportion_df()`.
- `NULL` **or** `FALSE`, to keep S, I, and R separate. This is only possible in functions that return results per category: `count_df()`, `proportion_df()`, `sir_df()`, `ggplot_sir()`, `geom_sir()`, `labels_sir_count()`, and `bug_drug_combinations()`. Functions that return a single proportion or count, such as `susceptibility()`, `count_resistant()`, `antibiogram()`, `wisca()`, and `format()` on the result of `bug_drug_combinations()`, require 'I' to be considered either susceptible or resistant, and return an error.

Use `susceptibility()` and `resistance()` (or `count_susceptible()` and `count_resistant()`) to follow the guideline, or e.g. `proportion_SI()` and `proportion_R()` to be explicit regardless of the guideline.

The `combine_SI` argument of the functions that return results per category, `antibiogram()`, and `wisca()` is deprecated in favour of `guideline`. If set, it is followed literally and takes precedence over `guideline`.

## Examples

```
if (require("ggplot2") && require("dplyr")) {
  # get antimicrobial results for drugs against a UTI:
  ggplot(example_isolates %>% select(AMX, NIT, FOS, TMP, CIP)) +
    geom_sir()
}
if (require("ggplot2") && require("dplyr")) {
  # prettify the plot using some additional functions:
  df <- example_isolates %>% select(AMX, NIT, FOS, TMP, CIP)
  ggplot(df) +
    geom_sir() +
    scale_y_percent() +
    scale_sir_colours(aesthetics = "fill") +
    labels_sir_count() +
    theme_sir()
}
if (require("ggplot2") && require("dplyr")) {
  # or better yet, simplify this using the wrapper function - a single command:
  example_isolates %>%
    select(AMX, NIT, FOS, TMP, CIP) %>%
    ggplot_sir()
}
if (require("ggplot2") && require("dplyr")) {
```

```
  # get only proportions and no counts:
  example_isolates %>%
    select(AMX, NIT, FOS, TMP, CIP) %>%
    ggplot_sir(datalabels = FALSE)
}
if (require("ggplot2") && require("dplyr")) {
  # add other ggplot2 arguments as you like:
  example_isolates %>%
    select(AMX, NIT, FOS, TMP, CIP) %>%
    ggplot_sir(
      width = 0.5,
      colour = "black",
      size = 1,
      linetype = 2,
      alpha = 0.25
    )
}
if (require("ggplot2") && require("dplyr")) {
  # you can alter the colours with colour names:
  example_isolates %>%
    select(AMX) %>%
    ggplot_sir(colours = c(SI = "yellow"))
}
if (require("ggplot2") && require("dplyr")) {
  # but you can also use the built-in colour-blind friendly colours for
  # your plots, where "S" is green, "I" is yellow and "R" is red:
  data.frame(
    x = c("Value1", "Value2", "Value3"),
    y = c(1, 2, 3),
    z = c("Value4", "Value5", "Value6")
  ) %>%
    ggplot() +
    geom_col(aes(x = x, y = y, fill = z)) +
    scale_sir_colours(
      aesthetics = "fill",
      Value4 = "S", Value5 = "I", Value6 = "R"
    )
}
if (require("ggplot2") && require("dplyr")) {
  # resistance of ciprofloxacine per age group
  example_isolates %>%
    mutate(first_isolate = first_isolate()) %>%
    filter(
      first_isolate == TRUE,
      mo == as.mo("Escherichia coli")
    ) %>%
    # age_groups() is also a function in this AMR package:
    group_by(age_group = age_groups(age)) %>%
    select(age_group, CIP) %>%
    ggplot_sir(x = "age_group")
}
if (require("ggplot2") && require("dplyr")) {
  # a shorter version which also adjusts data label colours:
```

```
  example_isolates %>%
    select(AMX, NIT, FOS, TMP, CIP) %>%
    ggplot_sir(colours = FALSE)
}
if (require("ggplot2") && require("dplyr")) {
  # it also supports groups (don't forget to use the group var on `x` or `facet`):
  example_isolates %>%
    filter(mo_is_gram_negative(), ward != "Outpatient") %>%
    # select only UTI-specific drugs
    select(ward, AMX, NIT, FOS, TMP, CIP) %>%
    group_by(ward) %>%
    ggplot_sir(
      x = "ward",
      facet = "antibiotic",
      nrow = 1,
      title = "AMR of Anti-UTI Drugs Per Ward",
      x.title = "Ward",
      datalabels = FALSE
    )
}
```

---

`guess_ab_col` *Guess Antibiotic Column*

---

## Description

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

## Usage

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

## 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. |
| `only_sir_columns` | A logical to indicate whether only antimicrobial columns must be included that were transformed to class sir on beforehand. Defaults to `FALSE` if no columns of `x` have a class sir. |

### Details

You can look for an antibiotic (trade) name or abbreviation and it will search x and the antimicrobials data set for any column containing a name or code of that antibiotic.

### Value

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

### Examples

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

guess_ab_col(df, "amoxicillin")
guess_ab_col(df, "J01AA07") # ATC code of tetracycline

guess_ab_col(df, "J01AA07", verbose = TRUE)

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

---

## interpretive_rules *Apply Interpretive Rules*

---

### Description

Apply rules from clinical breakpoints notes and expected resistant phenotypes as defined by e.g. the European Committee on Antimicrobial Susceptibility Testing (EUCAST, https://www.eucast.org), see *Source*. Use eucast_dosage() to get a data.frame with advised dosages of a certain bug-drug combination, which is based on the dosage data set.

To improve the interpretation of the antibiogram before CLSI/EUCAST interpretive rules are applied, some AMR-specific rules can be applied at default, see *Details*.

### Usage

```
interpretive_rules(
  x,
  col_mo = NULL,
  info = interactive(),
  rules = getOption("AMR_interpretive_rules", default = c("breakpoints",
```

```
    "expected_phenotypes")),
  guideline = getOption("AMR_guideline", "EUCAST"),
  verbose = FALSE,
  version_breakpoints = 16,
  version_expected_phenotypes = 1.2,
  version_expertrules = 3.3,
  ampc_cephalosporin_resistance = NA,
  only_sir_columns = any(is.sir(x)),
  custom_rules = NULL,
  overwrite = FALSE,
  add_if_missing = TRUE,
  ...
)

eucast_rules(
  x,
  col_mo = NULL,
  info = interactive(),
  rules = getOption("AMR_interpretive_rules", default = c("breakpoints",
    "expected_phenotypes")),
  ...
)

clsi_rules(
  x,
  col_mo = NULL,
  info = interactive(),
  rules = getOption("AMR_interpretive_rules", default = c("breakpoints",
    "expected_phenotypes")),
  ...
)

eucast_dosage(ab, administration = "iv", version_breakpoints = 15)
```

## Arguments

| | |
|---|---|
| x | A data set with antimicrobials columns, such as amox, AMX and AMC. |
| col_mo | Column name of the names or codes of the microorganisms (see as.mo()) - the default is 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 - the default is only print while in interactive sessions. |
| rules | A character vector that specifies which rules should be applied. Must be one or more of "breakpoints", "expected_phenotypes", "expert", "other", "custom", "all", and defaults to c("breakpoints", "expected_phenotypes"). The default value can be set to another value using the package option AMR_interpretive_rules: options(AMR_interpretive_rules = "all"). If using "custom", be sure to fill in argument custom_rules too. Custom rules can be created with custom_interpretive_rules(). |

`guideline` A guideline name, either "EUCAST" (default) or "CLSI". This can be set with the package option `AMR_guideline`.

`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. Using Verbose mode takes a lot more time.

`version_breakpoints` The version number to use for the EUCAST Clinical Breakpoints guideline. Can be `"16.0"`, `"15.0"`, `"14.0"`, `"13.1"`, `"12.0"`, `"11.0"`, or `"10.0"`.

`version_expected_phenotypes` The version number to use for the EUCAST Expected Phenotypes. Can be `"1.2"`.

`version_expertrules` The version number to use for the EUCAST Expert Rules and Intrinsic Resistance guideline. Can be `"3.3"`.

`ampc_cephalosporin_resistance` (only applies when `rules` contains `"expert"` or `"all"`) a character value that should be applied to cefotaxime, ceftriaxone and ceftazidime for AmpC de-repressed cephalosporin-resistant mutants - the default is `NA`. Currently only works when `version_expertrules` is `3.2` and higher; these versions of '*EUCAST Expert Rules on Enterobacterales*' state that results of cefotaxime, ceftriaxone and ceftazidime should be reported with a note, or results should be suppressed (emptied) for these three drugs. A value of `NA` (the default) for this argument will remove results for these three drugs, while e.g. a value of `"R"` will make the results for these drugs resistant. Use `NULL` or `FALSE` to not alter results for these three drugs of AmpC de-repressed cephalosporin-resistant mutants. Using `TRUE` is equal to using `"R"`.
For *EUCAST Expert Rules* v3.2, this rule applies to: *Citrobacter braakii*, *Citrobacter freundii*, *Citrobacter gillenii*, *Citrobacter murliniae*, *Citrobacter rodenticum*, *Citrobacter sedlakii*, *Citrobacter werkmanii*, *Citrobacter youngae*, *Enterobacter*, *Hafnia alvei*, and *Klebsiella aerogenes*.

`only_sir_columns` A logical to indicate whether only antimicrobial columns must be included that were transformed to class sir on beforehand. Defaults to `FALSE` if no columns of `x` have a class sir.

`custom_rules` Custom rules to apply, created with `custom_interpretive_rules()`.

`overwrite` A logical indicating whether to overwrite existing SIR values (default: `FALSE`). When `FALSE`, only non-SIR values are modified (i.e., any value that is not already S, I or R). To ensure compliance with EUCAST guidelines, **this should remain** `FALSE`, as EUCAST notes often state that an organism "should be tested for susceptibility to individual agents or be reported resistant".

`add_if_missing` A logical indicating whether rules should also be applied to missing (`NA`) values (default: `TRUE`). When `FALSE`, rules are only applied to cells that already contain an SIR value; cells with `NA` are left untouched. This is particularly useful when using `overwrite = TRUE` with custom rules and you want to update reported results without imputing values for untested drugs.

| | |
|---|---|
| ... | Column names of antimicrobials. To automatically detect antimicrobial column names, do not provide any named arguments; `guess_ab_col()` will then be used for detection. To manually specify a column, provide its name (case-insensitive) as an argument, e.g. `AMX = "amoxicillin"`. To skip a specific antimicrobial, set it to `NULL`, e.g. `TIC = NULL` to exclude ticarcillin. If a manually defined column does not exist in the data, it will be skipped with a warning. |
| ab | Any (vector of) text that can be coerced to a valid antimicrobial drug code with `as.ab()`. |
| administration | Route of administration, either `""`, `"im"`, `"iv"`, `"oral"`, or NA. |

## Details

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

The file containing all interpretive rules is located here: https://github.com/msberends/AMR/blob/main/data-raw/interpretive_rules.tsv. **Note:** Old taxonomic names are replaced with the current taxonomy where applicable. For example, *Ochrobactrum anthropi* was renamed to *Brucella anthropi* in 2020; older interpretive rules may not yet contain this new taxonomic name. The `AMR` package contains the full microbial taxonomy updated until 5th of October, 2026, see microorganisms.

### Custom Rules:

Custom rules can be created using `custom_interpretive_rules()`, e.g.:

```
x <- custom_interpretive_rules(AMC == "R" & genus == "Klebsiella" ~ aminopenicillins == "R",
                               AMC == "I" & genus == "Klebsiella" ~ aminopenicillins == "I")

interpretive_rules(example_isolates, rules = "custom", custom_rules = x)
```

### 'Other' Rules:

Before further processing, two non-EUCAST rules about drug combinations can be applied to improve the efficacy of the EUCAST rules, and the reliability of your data (analysis). These rules are:

1. A drug **with** enzyme inhibitor will be set to S if the same drug **without** enzyme inhibitor is S
2. A drug **without** enzyme inhibitor will be set to R if the same drug **with** enzyme inhibitor is R

Important examples include amoxicillin and amoxicillin/clavulanic acid, and trimethoprim and trimethoprim/sulfamethoxazole. Needless to say, for these rules to work, both drugs must be available in the data set.

Since these rules are not officially approved by EUCAST, they are not applied at default. To use these rules, include `"other"` to the `rules` argument, or use `interpretive_rules(..., rules = "all")`. You can also set the package option `AMR_interpretive_rules`, i.e. run `options(AMR_interpretive_rules = "all")`.

## Value

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

## Download Our Reference Data

All reference data sets in the AMR package - including information on microorganisms, antimicrobials, and clinical breakpoints - are freely available for download in multiple formats: R, MS Excel, Apache Feather, Apache Parquet, SPSS, and Stata.

For maximum compatibility, we also provide machine-readable, tab-separated plain text files suitable for use in any software, including laboratory information systems.

Visit our website for direct download links, or explore the actual files in our GitHub repository.

## References


- EUCAST Expert Rules. Version 2.0, 2012.
  Leclercq et al. **EUCAST expert rules in antimicrobial susceptibility testing.** *Clin Microbiol Infect.* 2013;19(2):141-60; doi:10.1111/j.14690691.2011.03703.x
- EUCAST Expected Phenotypes. (link)
- EUCAST Breakpoint tables for interpretation of MICs and zone diameters. (link)


## Examples

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

head(a)


# apply EUCAST rules: some results wil be changed
b <- eucast_rules(a, overwrite = TRUE)

head(b)
```

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


# Dosage guidelines:

eucast_dosage(c("tobra", "genta", "cipro"), "iv")

eucast_dosage(c("tobra", "genta", "cipro"), "iv", version_breakpoints = 10)
```

---

`intrinsic_resistant` *Data Set Denoting Bacterial Intrinsic Resistance*

---

## Description

Data set containing 'EUCAST Expected Resistant Phenotypes' of *all* bug-drug combinations between the microorganisms and antimicrobials data sets.

## Usage

```
intrinsic_resistant
```

## Format

A tibble with 307 684 observations and 3 variables:

- `mo`
  Microorganism ID which occurs in `microorganisms$mo`. Names can be retrieved using `mo_name()`.
- `ab`
  Antimicrobial ID which occurs in `antimicrobials$ab`. Names can be retrieved using `ab_name()`.
- `version`
  Guideline name and version.

## Details

This data set is currently strictly follows 'EUCAST Expected Resistant Phenotypes' v1.2 (2023). The

This data set is internally used by:

- `not_intrinsic_resistant()` (an antimicrobial selector)
- `mo_is_intrinsic_resistant()`
- `wisca()` to model $\beta(1, 9999)$ for resistant bug-drug combinations, per doi:10.1093/jac/dkv397

### Download Our Reference Data

All reference data sets in the AMR package - including information on microorganisms, antimicrobials, and clinical breakpoints - are freely available for download in multiple formats: R, MS Excel, Apache Feather, Apache Parquet, SPSS, and Stata.

For maximum compatibility, we also provide machine-readable, tab-separated plain text files suitable for use in any software, including laboratory information systems.

Visit our website for direct download links, or explore the actual files in our GitHub repository.

### Examples

```
intrinsic_resistant
```

---

## italicise_taxonomy *Italicise Taxonomic Families, Genera, Species, Subspecies*

---

### Description

According to the binomial nomenclature, the lowest four taxonomic levels (family, genus, species, subspecies) should be printed in italics. This function finds taxonomic names within strings and makes them italic.

### Usage

```
italicise_taxonomy(string, type = c("markdown", "ansi", "html"))

italicize_taxonomy(string, type = c("markdown", "ansi", "html"))
```

### Arguments

| | |
|---|---|
| `string` | A character (vector). |
| `type` | Type of conversion of the taxonomic names, either "markdown", "html" or "ansi", see *Details*. |

### Details

This function finds the taxonomic names and makes them italic based on the microorganisms data set.

The taxonomic names can be italicised using markdown (the default) by adding `*` before and after the taxonomic names, or `<i>` and `</i>` when using html. When using 'ansi', ANSI colours will be added using `\033[3m` before and `\033[23m` after the taxonomic names. If multiple ANSI colours are not available, no conversion will occur.

This function also supports abbreviation of the genus if it is followed by a species, such as "E. coli" and "K. pneumoniae ozaenae".

## Examples

```
italicise_taxonomy("An overview of Staphylococcus aureus isolates")
italicise_taxonomy("An overview of S. aureus isolates")

cat(italicise_taxonomy("An overview of S. aureus isolates", type = "ansi"))
```

---

`join` *Join microorganisms to a Data Set*

---

## Description

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

## Usage

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

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

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

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

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

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

## Arguments

| | |
|---|---|
| `x` | Existing data set to join, or character vector. In case of a character vector, the resulting data.frame will contain a column 'x' with these values. |
| `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` (such as by = `"bacteria_id"`), or another column in microorganisms (but then it should be named, like by = `c("bacteria_id" = "fullname")`). |
| `suffix` | If there are non-joined duplicate variables in `x` and `y`, these suffixes will be added to the output to disambiguate them. Should be a character vector of length 2. |
| `...` | Ignored, only in place to allow future extensions. |

## Details

**Note:** As opposed to the `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.

If the `dplyr` package is installed, their join functions will be used. Otherwise, the much slower `merge()` and `interaction()` functions from base R will be used.

## Value

a data.frame

## Examples

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

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

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


if (require("dplyr")) {
  example_isolates %>%
    left_join_microorganisms() %>%
    colnames()
}
```

---

# key_antimicrobials *(Key) Antimicrobials for First Weighted Isolates*

---

## Description

These functions can be used to determine first weighted isolates by considering the phenotype for isolate selection (see first_isolate()). Using a phenotype-based method to determine first isolates is more reliable than methods that disregard phenotypes.

## Usage

```
key_antimicrobials(
  x = NULL,
  col_mo = NULL,
  universal = c("ampicillin", "amoxicillin/clavulanic acid", "cefuroxime",
   "piperacillin/tazobactam", "ciprofloxacin", "trimethoprim/sulfamethoxazole"),
```

```
  gram_negative = c("gentamicin", "tobramycin", "colistin", "cefotaxime", "ceftazidime",
    "meropenem"),
  gram_positive = c("vancomycin", "teicoplanin", "tetracycline", "erythromycin",
    "oxacillin", "rifampin"),
  antifungal = c("anidulafungin", "caspofungin", "fluconazole", "miconazole", "nystatin",
    "voriconazole"),
  only_sir_columns = any(is.sir(x)),
  ...
)

all_antimicrobials(x = NULL, only_sir_columns = any(is.sir(x)), ...)

antimicrobials_equal(
  y,
  z,
  type = c("points", "keyantimicrobials"),
  ignore_I = TRUE,
  points_threshold = 2,
  ...
)
```

## Arguments

| | |
|---|---|
| x | A data.frame with antimicrobials columns, like AMX or amox. Can be left blank to determine automatically. |
| col_mo | Column name of the names or codes of the microorganisms (see as.mo()) - the default is the first column of class mo. Values will be coerced using as.mo(). |
| universal | Names of **broad-spectrum** antimicrobial drugs, case-insensitive. Set to NULL to ignore. See *Details* for the default antimicrobial drugs. |
| gram_negative | Names of antibiotic drugs for **Gram-positives**, case-insensitive. Set to NULL to ignore. See *Details* for the default antibiotic drugs. |
| gram_positive | Names of antibiotic drugs for **Gram-negatives**, case-insensitive. Set to NULL to ignore. See *Details* for the default antibiotic drugs. |
| antifungal | Names of antifungal drugs for **fungi**, case-insensitive. Set to NULL to ignore. See *Details* for the default antifungal drugs. |
| only_sir_columns | A logical to indicate whether only antimicrobial columns must be included that were transformed to class sir on beforehand. Defaults to FALSE if no columns of x have a class sir. |
| ... | Ignored, only in place to allow future extensions. |
| y, z | character vectors to compare. |
| type | Type to determine weighed isolates; can be "keyantimicrobials" or "points", see *Details*. |
| ignore_I | logical to indicate whether antibiotic interpretations with "I" will be ignored when type = "keyantimicrobials", see *Details*. |

`points_threshold`
: Minimum number of points to require before differences in the antibiogram will lead to inclusion of an isolate when `type = "points"`, see *Details*.

## Details

The `key_antimicrobials()` and `all_antimicrobials()` functions are context-aware. This means that the `x` argument can be left blank if used inside a data.frame call, see *Examples*.

The function `key_antimicrobials()` returns a character vector with 12 antimicrobial results for every isolate. The function `all_antimicrobials()` returns a character vector with all antimicrobial drug results for every isolate. These vectors can then be compared using `antimicrobials_equal()`, to check if two isolates have generally the same antibiogram. Missing and invalid values are replaced with a dot (`"."`) by `key_antimicrobials()` and ignored by `antimicrobials_equal()`.

Please see the `first_isolate()` function how these important functions enable the 'phenotype-based' method for determination of first isolates.

The default antimicrobial drugs used for **all rows** (set in `universal`) are:

- Ampicillin
- Amoxicillin/clavulanic acid
- Cefuroxime
- Ciprofloxacin
- Piperacillin/tazobactam
- Trimethoprim/sulfamethoxazole

The default antimicrobial drugs used for **Gram-negative bacteria** (set in `gram_negative`) are:

- Cefotaxime
- Ceftazidime
- Colistin
- Gentamicin
- Meropenem
- Tobramycin

The default antimicrobial drugs used for **Gram-positive bacteria** (set in `gram_positive`) are:

- Erythromycin
- Oxacillin
- Rifampin
- Teicoplanin
- Tetracycline
- Vancomycin

The default antimicrobial drugs used for **fungi** (set in `antifungal`) are:

- Anidulafungin

- Caspofungin
- Fluconazole
- Miconazole
- Nystatin
- Voriconazole

## See Also

first_isolate()

## Examples

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

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

# those strings can be compared with:
antimicrobials_equal(strainA, strainB, type = "keyantimicrobials")
# TRUE, because I is ignored (as well as missing values)

antimicrobials_equal(strainA, strainB, type = "keyantimicrobials", ignore_I = FALSE)
# FALSE, because I is not ignored and so the 4th [character] differs


if (require("dplyr")) {
  # set key antimicrobials to a new variable
  my_patients <- example_isolates %>%
    mutate(keyab = key_antimicrobials(antifungal = NULL)) %>% # no need to define `x`
    mutate(
      # now calculate first isolates
      first_regular = first_isolate(col_keyantimicrobials = FALSE),
      # and first WEIGHTED isolates
      first_weighted = first_isolate(col_keyantimicrobials = "keyab")
    )

  # Check the difference in this data set, 'weighted' results in more isolates:
  sum(my_patients$first_regular, na.rm = TRUE)
  sum(my_patients$first_weighted, na.rm = TRUE)
}
```

---

kurtosis *Kurtosis of the Sample*

---

### Description

Kurtosis is a measure of the "tailedness" of the probability distribution of a real-valued random variable. A normal distribution has a kurtosis of 3 and a excess kurtosis of 0.

### Usage

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

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

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

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

### Arguments

| | |
|---|---|
| `x` | A vector of values, a matrix or a data.frame. |
| `na.rm` | A logical to indicate whether `NA` values should be stripped before the computation proceeds. |
| `excess` | A logical to indicate whether the *excess kurtosis* should be returned, defined as the kurtosis minus 3. |

### See Also

`skewness()`

### Examples

```
kurtosis(rnorm(10000))
kurtosis(rnorm(10000), excess = TRUE)
```

---

## `like` *Vectorised Pattern Matching with Keyboard Shortcut*

---

### Description

Convenient wrapper around `grepl()` 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 %unlike% pattern

x %like_case% pattern

x %unlike_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 vector containing regular expressions (or a character string for fixed = TRUE) to be matched in the given character vector. Coerced by as.character() to a character string if possible. |
| ignore.case | If FALSE, the pattern matching is *case sensitive* and if TRUE, case is ignored during matching. |

## Details

These like() and %like%/%unlike% functions:

- Are case-insensitive (use %like_case%/%unlike_case% for case-sensitive matching)
- Support multiple patterns
- Check if pattern is a valid regular expression and sets fixed = TRUE if not, to greatly improve speed (vectorised over pattern)
- Always use compatibility with Perl unless fixed = TRUE, to greatly improve speed

Using RStudio? The %like%/%unlike% functions can also be directly inserted in your code from the Addins menu and can have its own keyboard shortcut like Shift+Ctrl+L or Shift+Cmd+L (see menu Tools > Modify Keyboard Shortcuts...). If you keep pressing your shortcut, the inserted text will be iterated over %like% -> %unlike% -> %like_case% -> %unlike_case%.

## Value

A logical vector

## Source

Idea from the like function from the data.table package, although altered as explained in *Details*.

## See Also

grepl()

## Examples

```
# data.table has a more limited version of %like%, so detach it
# (without unloading it, as data.table does not support being unloaded):
try(detach("package:data.table"), silent = TRUE)

a <- "This is a test"
b <- "TEST"
a %like% b
b %like% a

# also supports multiple patterns
a <- c("Test case", "Something different", "Yet another thing")
b <- c("case", "diff", "yet")
a %like% b
a %unlike% b

a[1] %like% b
a %like% b[1]


# get isolates whose name start with 'Entero' (case-insensitive)
example_isolates[which(mo_name() %like% "^entero"), ]

if (require("dplyr")) {
  example_isolates %>%
    filter(mo_name() %like% "^ent")
}
```

---

# mdro *Determine Multidrug-Resistant Organisms (MDRO)*

---

## Description

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

## Usage

```
mdro(
  x = NULL,
  guideline = "CMI 2012",
  col_mo = NULL,
  esbl = NA,
  carbapenemase = NA,
  mecA = NA,
  mecC = NA,
  vanA = NA,
  vanB = NA,
```

```
  info = interactive(),
  pct_required_classes = 0.5,
  combine_SI = TRUE,
  verbose = FALSE,
  only_sir_columns = any(is.sir(x)),
  infer_from_combinations = TRUE,
  ...
)

brmo(x = NULL, only_sir_columns = any(is.sir(x)), ...)

mrgn(x = NULL, only_sir_columns = any(is.sir(x)), verbose = FALSE, ...)

mdr_tb(x = NULL, only_sir_columns = any(is.sir(x)), verbose = FALSE, ...)

mdr_cmi2012(x = NULL, only_sir_columns = any(is.sir(x)), verbose = FALSE, ...)

eucast_exceptional_phenotypes(
  x = NULL,
  only_sir_columns = any(is.sir(x)),
  verbose = FALSE,
  ...
)
```

## Arguments

| | |
|---|---|
| x | A data.frame with antimicrobials columns, like AMX or amox. Can be left blank for automatic determination. |
| guideline | A specific guideline to follow, see sections *Supported International / National Guidelines* and *Using Custom Guidelines* below. When left empty, the publication by Magiorakos *et al.* (see below) will be followed. |
| col_mo | Column name of the names or codes of the microorganisms (see as.mo()) - the default is the first column of class mo. Values will be coerced using as.mo(). |
| esbl | logical values, or a column name containing logical values, indicating the presence of an ESBL gene (or production of its proteins). |
| carbapenemase | logical values, or a column name containing logical values, indicating the presence of a carbapenemase gene (or production of its proteins). |
| mecA | logical values, or a column name containing logical values, indicating the presence of a *mecA* gene (or production of its proteins). |
| mecC | logical values, or a column name containing logical values, indicating the presence of a *mecC* gene (or production of its proteins). |
| vanA | logical values, or a column name containing logical values, indicating the presence of a *vanA* gene (or production of its proteins). |
| vanB | logical values, or a column name containing logical values, indicating the presence of a *vanB* gene (or production of its proteins). |
| info | A logical to indicate whether progress should be printed to the console - the default is only print while in interactive sessions. |

`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. In a future version, the handling of 'I' will follow the selected MDRO guideline (#323).

`verbose`
: A logical to turn Verbose mode on and off (default is off). In Verbose mode, the function returns a data set with the MDRO results in logbook form with extensive info about which isolates would be MDRO-positive, or why they are not.

`only_sir_columns`
: A logical to indicate whether only antimicrobial columns must be included that were transformed to class sir on beforehand. Defaults to `FALSE` if no columns of `x` have a class sir.

`infer_from_combinations`
: A logical to indicate whether resistance for a missing base beta-lactam drug should be inferred from an available drug+inhibitor combination (e.g., piperacillin from piperacillin/tazobactam). The clinical basis is that resistance in a combination always implies resistance in the base drug, since the enzyme inhibitor provides no benefit when the organism is truly resistant. Only resistance is inferred; susceptibility in a combination does **not** imply susceptibility in the base drug (the inhibitor may be responsible). Defaults to `TRUE`.

`...`
: Column names of antimicrobials. To automatically detect antimicrobial column names, do not provide any named arguments; `guess_ab_col()` will then be used for detection. To manually specify a column, provide its name (case-insensitive) as an argument, e.g. `AMX = "amoxicillin"`. To skip a specific antimicrobial, set it to `NULL`, e.g. `TIC = NULL` to exclude ticarcillin. If a manually defined column does not exist in the data, it will be skipped with a warning.

## Details

These functions are context-aware. This means that the `x` argument can be left blank if used inside a data.frame call, see *Examples*.

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`).

**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.

**Supported International / National Guidelines:**

Please suggest to implement guidelines by letting us know.

Currently supported guidelines are (case-insensitive):

- `guideline = "CMI 2012"` (default)
  Magiorakos AP, Srinivasan A *et al.* "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) (doi:10.1111/j.1469-0691.2011.03570.x)
- `guideline = "EUCAST 3.3"` (or simply `guideline = "EUCAST"`)
  The European international guideline - EUCAST Expert Rules Version 3.3 "Intrinsic Resistance and Unusual Phenotypes" (link)
  Also:
  - `guideline = "EUCAST 3.2"`
    The former European international guideline - EUCAST Expert Rules Version 3.2 "Intrinsic Resistance and Unusual Phenotypes" (link)
  - `guideline = "EUCAST 3.1"`
    The former 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"`
  The German national guideline - Mueller et al. (2015) Antimicrobial Resistance and Infection Control 4:7; doi:10.1186/s1375601500476
- `guideline = "BRMO 2024"` (or simply `guideline = "BRMO"`)
  The Dutch national guideline - Samenwerkingverband Richtlijnen Infectiepreventie (SRI) (2024) "Bijzonder Resistente Micro-Organismen (BRMO)" (link)
  Also:
  - `guideline = "BRMO 2017"`
    The former Dutch national guideline - Werkgroep Infectiepreventie (WIP), RIVM, last revision as of 2017: "Bijzonder Resistente Micro-Organismen (BRMO)"

**Using Custom Guidelines:**

Using a custom MDRO guideline is of importance if you have custom rules to determine MDROs in your hospital, e.g., rules that are dependent on ward, state of contact isolation or other variables in your data.

Custom guidelines can be set with the `custom_mdro_guideline()` function.

## Value

- If `verbose` is set to `TRUE`:
  A data.frame containing columns `row_number`, `microorganism`, `MDRO`, `reason`, `all_nonsusceptible_columns`, `guideline`

- 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, except for custom guidelines:
  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. genotypic) tests

### Interpretation of SIR

In 2019, the European Committee on Antimicrobial Susceptibility Testing (EUCAST) has decided to change the definitions of susceptibility testing categories S, I, and R (https://www.eucast.org/bacteria/clinical-breakpoints-and-interpretation/definition-of-s-i-and-r/).

In aggregated analyses, such as proportions, counts, antibiograms, and plots, the 'I' category must be considered either susceptible or resistant. In epidemiological AMR data analysis, this follows the guideline: EUCAST considers 'I' susceptible, while CLSI considers 'I' resistant. The dose-dependent susceptibility of CLSI ('SDD') is always considered susceptible, as it equals what EUCAST defines as 'I'. All functions with a `guideline` argument for this purpose use these categories:

| `guideline` | Susceptible | Resistant |
|---|---|---|
| `"EUCAST"` | S, SDD, I, WT | R, NWT, NS |
| `"CLSI"` | S, SDD, WT | I, R, NWT, NS |

The values 'WT' (wild type) and 'NWT' (non-wild type) result from epidemiological cut-off values (ECOFFs) instead of clinical breakpoints. Since they are no clinical categories, 'WT' is always considered susceptible and 'NWT' always resistant, regardless of the guideline. The value 'NS' (non-susceptible) is always considered resistant, and 'NI' (non-interpretable) is always considered a missing result.

The `guideline` argument can be:

- **Left unset**, to use the package option `AMR_guideline`, or `"EUCAST"` if this option is not set. In the latter case, a message is shown once per session (but not in R Markdown or Quarto documents); set the option to silence it.
- `"EUCAST"`, or any EUCAST version such as `"EUCAST 2025"`, to consider 'I' susceptible, e.g., to return `"SI"` vs. `"R"` in `proportion_df()`.
- `"CLSI"`, or any CLSI version such as `"CLSI 2025"`, to consider 'I' resistant, e.g., to return `"S"` vs. `"IR"` in `proportion_df()`.
- `NULL` **or** `FALSE`, to keep S, I, and R separate. This is only possible in functions that return results per category: `count_df()`, `proportion_df()`, `sir_df()`, `ggplot_sir()`, `geom_sir()`,

labels_sir_count(), and bug_drug_combinations(). Functions that return a single proportion or count, such as susceptibility(), count_resistant(), antibiogram(), wisca(), and format() on the result of bug_drug_combinations(), require 'I' to be considered either susceptible or resistant, and return an error.

Use susceptibility() and resistance() (or count_susceptible() and count_resistant()) to follow the guideline, or e.g. proportion_SI() and proportion_R() to be explicit regardless of the guideline.

The combine_SI argument of the functions that return results per category, antibiogram(), and wisca() is deprecated in favour of guideline. If set, it is followed literally and takes precedence over guideline.

## See Also

custom_mdro_guideline()

## Examples

```
out <- mdro(example_isolates)
str(out)
table(out)

out <- mdro(example_isolates, guideline = "EUCAST 3.3")
table(out)


if (require("dplyr")) {
  # no need to define `x` when used inside dplyr verbs:
  example_isolates %>%
    mutate(MDRO = mdro()) %>%
    count(MDRO)
}
```

---

# mean_amr_distance *Calculate the Mean AMR Distance*

---

## Description

Calculates a normalised mean for antimicrobial resistance between multiple observations, to help to identify similar isolates without comparing antibiograms by hand.

## Usage

```
mean_amr_distance(x, ...)

## S3 method for class 'sir'
mean_amr_distance(x, ..., combine_SI = TRUE)
```

```
## S3 method for class 'data.frame'
mean_amr_distance(x, ..., combine_SI = TRUE)

amr_distance_from_row(amr_distance, row)
```

## Arguments

| | |
|---|---|
| x | A vector of class sir, mic or disk, or a data.frame containing columns of any of these classes. |
| ... | Variables to select. Supports tidyselect language such as `where(is.mic)`, `starts_with(...)`, or `column1:column4`, and can thus also be antimicrobial selectors. |
| combine_SI | A logical to indicate whether all values of S, SDD, and I must be merged into one, so the input only consists of S+I vs. R (susceptible vs. resistant) - the default is `TRUE`. |
| amr_distance | The outcome of `mean_amr_distance()`. |
| row | An index, such as a row number. |

## Details

The mean AMR distance is effectively the Z-score; a normalised numeric value to compare AMR test results which can help to identify similar isolates, without comparing antibiograms by hand.

MIC values (see `as.mic()`) are transformed with `log2()` first; their distance is thus calculated as `(log2(x) - mean(log2(x))) / sd(log2(x))`.

SIR values (see `as.sir()`) are transformed using `"S"` = 1, `"I"` = 2, and `"R"` = 3. If `combine_SI` is `TRUE` (default), the `"I"` will be considered to be 1.

For data sets, the mean AMR distance will be calculated per column, after which the mean per row will be returned, see *Examples*.

Use `amr_distance_from_row()` to subtract distances from the distance of one row, see *Examples*.

## Interpretation

Isolates with distances less than 0.01 difference from each other should be considered similar. Differences lower than 0.025 should be considered suspicious.

## Examples

```
sir <- random_sir(10)
sir
mean_amr_distance(sir)

mic <- random_mic(10)
mic
mean_amr_distance(mic)
# equal to the Z-score of their log2:
(log2(mic) - mean(log2(mic))) / sd(log2(mic))

disk <- random_disk(10)
disk
```

```
mean_amr_distance(disk)

y <- data.frame(
  id = LETTERS[1:10],
  amox = random_sir(10, ab = "amox", mo = "Escherichia coli"),
  cipr = random_disk(10, ab = "cipr", mo = "Escherichia coli"),
  gent = random_mic(10, ab = "gent", mo = "Escherichia coli"),
  tobr = random_mic(10, ab = "tobr", mo = "Escherichia coli")
)
y
mean_amr_distance(y)
y$amr_distance <- mean_amr_distance(y, is.mic(y))
y[order(y$amr_distance), ]

if (require("dplyr")) {
  y %>%
    mutate(
      amr_distance = mean_amr_distance(y),
      check_id_C = amr_distance_from_row(amr_distance, id == "C")
    ) %>%
    arrange(check_id_C)
}
if (require("dplyr")) {
  # support for groups
  example_isolates %>%
    filter(mo_genus() == "Enterococcus" & mo_species() != "") %>%
    select(mo, TCY, carbapenems()) %>%
    group_by(mo) %>%
    mutate(dist = mean_amr_distance(.)) %>%
    arrange(mo, dist)
}
```

---

`microorganisms` *Data Set with 103 222 Taxonomic Records of Microorganisms*

---

## Description

A data set containing the full microbial taxonomy (**last updated: 5th of October, 2026**) of seven domains. This data set is the backbone of this `AMR` package. MO codes can be looked up using `as.mo()` and microorganism properties can be looked up using any of the `mo_*` functions.

This data set is carefully crafted, yet made 100% reproducible from public and authoritative taxonomic sources (using this script), namely: *List of Prokaryotic names with Standing in Nomenclature (LPSN)* for bacteria, *MycoBank* for fungi, and *Global Biodiversity Information Facility (GBIF), via Catalogue of Life (COL)* for all others taxons.

## Usage

```
microorganisms
```

## Format

A tibble with 103 222 observations and 28 variables:

- `mo`
  ID of microorganism as used by this package. **This is a unique identifier.**
- `fullname`
  Full name, like `"Escherichia coli"`. For the taxonomic ranks genus, species and subspecies, this is the 'pasted' text of genus, species, and subspecies. For all taxonomic ranks higher than genus, this is the name of the taxon. **This is a unique identifier.**
- `status`
  Status of the taxon, either `"accepted"`, `"synonym"`, or `"unknown"`
- `domain`, `kingdom`, `phylum`, `class`, `order`, `family`, `genus`, `species`, `subspecies`
  Taxonomic rank of the microorganism. Note that for fungi, *phylum* is used for their taxonomic *division*. Also, for fungi, *subkingdom* and *subdivision* were left out since they do not occur in the bacterial taxonomy. For all species outside the domains of Bacteria and Archaea, the `domain` and `kingdom` are identical.
- `rank`
  Text of the taxonomic rank of the microorganism, such as `"species"` or `"genus"`
- `ref`
  Abbreviated authority citation for the nomenclatural act that established the current name combination, following ICNP conventions. For species described in their current genus (*sp. nov.*), this is the original description author(s) and year. For species transferred to a different genus (*comb. nov.*), this is the reclassification author(s) and year. Emendations are excluded. For synonyms, this is the authority under which the synonym was originally published. This field is directly retrieved from the source specified in the column `source`. Diacritics were removed to comply with CRAN, that only allows ASCII characters.
- `oxygen_tolerance`
  Oxygen tolerance, either `"aerobe"`, `"anaerobe"`, `"anaerobe/microaerophile"`, `"facultative anaerobe"`, `"likely facultative anaerobe"`, `"microaerophile"`, or NA. These data were retrieved from BacDive (see *Source*). Items that contain "likely" are missing from BacDive and were extrapolated from other species within the same genus to guess the oxygen tolerance. Currently 59.5% of all ~48 000 bacteria in the data set contain an oxygen tolerance.
- `morphology`
  Morphology (cell shape), either `"cocci"`, `"coccobacilli"`, `"filamentous"`, `"likely cocci"`, `"likely coccobacilli"`, `"likely filamentous"`, `"likely rods"`, `"likely spirilla"`, `"rods"`, `"spirilla"`, or NA. These data were retrieved from BacDive (see *Source*). Genera that are clinically established as coccobacilli (the HACEK group and beyond, such as *Haemophilus* and *Acinetobacter*) are classified as such regardless of BacDive majority vote. Items that contain "likely" are missing from BacDive and were extrapolated from other species within the same genus. Currently 57.4% of all ~48 000 bacteria in the data set contain a morphology.
- `source`
  Either `"GBIF"`, `"LPSN"`, `"MycoBank"`, or `"manually added"` (see *Source*)
- `lpsn`
  Identifier ('Record number') of List of Prokaryotic names with Standing in Nomenclature (LPSN). This will be the first/highest LPSN identifier to keep one identifier per row. For example, *Acetobacter ascendens* has LPSN Record number 7864 and 11011. Only the first is

available in the `microorganisms` data set. **This is a unique identifier**, though available for only ~37 000 records.

- `lpsn_parent`
  LPSN identifier of the parent taxon
- `lpsn_renamed_to`
  LPSN identifier of the currently valid taxon
- `mycobank`
  Identifier ('MycoBank #') of MycoBank. **This is a unique identifier**, though available for only ~25 000 records.
- `mycobank_parent`
  MycoBank identifier of the parent taxon
- `mycobank_renamed_to`
  MycoBank identifier of the currently valid taxon
- `gbif`
  Identifier ('taxonID') of Global Biodiversity Information Facility (GBIF), via Catalogue of Life (COL). **This is a unique identifier**, though available for only ~79 000 records.
- `gbif_parent`
  GBIF identifier of the parent taxon
- `gbif_renamed_to`
  GBIF identifier of the currently valid taxon
- `prevalence`
  Prevalence of the microorganism based on Bartlett *et al.* (2022, doi:10.1099/mic.0.001269), see `mo_matching_score()` for the full explanation
- `snomed`
  Systematized Nomenclature of Medicine (SNOMED) code of the microorganism, version of 16th of July, 2024 (see *Source*). Use `mo_snomed()` to retrieve it quickly, see `mo_property()`.

## Details

Please note that entries are only based on LPSN, MycoBank, and GBIF (see below). Since these sources incorporate entries based on (recent) publications in the International Journal of Systematic and Evolutionary Microbiology (IJSEM), it can happen that the year of publication is sometimes later than one might expect.

For example, *Staphylococcus pettenkoferi* was described for the first time in Diagnostic Microbiology and Infectious Disease in 2002 (doi:10.1016/s07328893(02)003991), but it was not until 2007 that a publication in IJSEM followed (doi:10.1099/ijs.0.643810). Consequently, the `AMR` package returns 2007 for `mo_year("S. pettenkoferi")`.

## Included Taxa

Included taxonomic data from LPSN, MycoBank, and GBIF are:

- All ~48 000 (sub)species from the domains of Archaea and Bacteria
- ~46 000 species from the kingdom of Fungi. 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. Only

relevant fungi are covered (such as all species of *Aspergillus*, *Candida*, *Cryptococcus*, *Histoplasma*, *Pneumocystis*, *Saccharomyces* and *Trichophyton*).

- ~5 300 (sub)species from the kingdom of Protozoa
- ~2 500 (sub)species from ~130 other relevant genera from the kingdom of Animalia (such as *Strongyloides* and *Taenia*)
- All ~33 000 previously accepted names of all included (sub)species (these were taxonomically renamed)
- The complete taxonomic tree of all included (sub)species: from domain to subspecies
- The identifier of the parent taxons
- The year and first author of the related scientific publication

**Manual additions:**

For convenience, some entries were added manually:

- All 37 groups and complexes of the microorganisms.groups data set, for cross-reference (examples include beta-haemolytic *Streptococcus* groups A to K, coagulase-negative *Staphylococcus* (CoNS), *Mycobacterium tuberculosis* complex, etc.)
- ~1 500 entries of *Salmonella*, such as the city-like serovars and groups A to H
- 1 entry of *Blastocystis* (*B. hominis*), although it officially does not exist (Noel *et al.* 2005, PMID 15634993)
- 9 other 'undefined' entries (unknown, unknown Gram-negatives, unknown Gram-positives, unknown yeast, unknown fungus, and unknown anaerobic Gram-pos/Gram-neg bacteria)

The syntax used to transform the original data to a cleansed R format, can be found here.

### Download Our Reference Data

All reference data sets in the AMR package - including information on microorganisms, antimicrobials, and clinical breakpoints - are freely available for download in multiple formats: R, MS Excel, Apache Feather, Apache Parquet, SPSS, and Stata.

For maximum compatibility, we also provide machine-readable, tab-separated plain text files suitable for use in any software, including laboratory information systems.

Visit our website for direct download links, or explore the actual files in our GitHub repository.

### References

Taxonomic entries were imported in this order of importance:

1. List of Prokaryotic names with Standing in Nomenclature (LPSN):

   Freese, HM *et al.* (2026). **TYGS and LPSN in 2025: a Global Core Biodata Resource for genome-based classification and nomenclature of prokaryotes within DSMZ Digital Diversity.** Nucleic Acids Research, 54, D884–D891; doi:10.1093/nar/gkaf1110. Accessed from `https://lpsn.dsmz.de` on 5th of October, 2026.

2. MycoBank:

   Vincent, R *et al* (2013). **MycoBank gearing up for new horizons.** IMA Fungus, 4(2), 371-9; doi:10.5598/imafungus.2013.04.02.16. Accessed from `https://www.mycobank.org` on 7th of January, 2026.

3. Global Biodiversity Information Facility (GBIF), via Catalogue of Life (COL):

   Banki, O. *et al.* (2026). Catalogue of Life (2026-04-18 XR). Catalogue of Life Foundation, Amsterdam, Netherlands. doi:10.48580/dgxjw. Accessed from https://www.gbif.org on 30th of April, 2026.

Furthermore, these sources were used for additional details:

- BacDive:

  Reimer, LC *et al.* (2022). ***BacDive* in 2022: the knowledge base for standardized bacterial and archaeal data.** Nucleic Acids Res., 50(D1):D741-D74; doi:10.1093/nar/gkab961. Accessed from https://bacdive.dsmz.de on 5th of October, 2026.

- Systematized Nomenclature of Medicine - Clinical Terms (SNOMED-CT):

  Public Health Information Network Vocabulary Access and Distribution System (PHIN VADS). US Edition of SNOMED CT from 1 September 2020. Value Set Name 'Microorganism', OID 2.16.840.1.114222.4.11.1009 (v12). Accessed from https://www.cdc.gov/phin/php/phinvads/ on 16th of July, 2024.

- Grimont *et al.* (2007). Antigenic Formulae of the Salmonella Serovars, 9th Edition. WHO Collaborating Centre for Reference and Research on *Salmonella* (WHOCC-SALM).

- Bartlett *et al.* (2022). **A comprehensive list of bacterial pathogens infecting humans** *Microbiology* 168:001269; doi:10.1099/mic.0.001269

## See Also

`as.mo()`, `mo_property()`, microorganisms.groups, microorganisms.codes, intrinsic_resistant

## Examples

```
microorganisms
```

---

# microorganisms.codes *Data Set with 6 293 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.

## Usage

```
microorganisms.codes
```

## Format

A tibble with 6 293 observations and 3 variables:

- `code`
  Commonly used code of a microorganism. **This is a unique identifier.**
- `mo`
  ID of the microorganism in the microorganisms data set. This is the record of the name that the code was named after, also if that name is outdated, so that the code and the microorganism ID remain relatable: e.g., the WHONET code `EAE` refers to *Enterobacter aerogenes* (`B_ENTRBC_AERG`). `as.mo()` returns the current name, here *Klebsiella aerogenes*. Codes for organisms that are not in the microorganisms data set, such as viruses, refer to `UNKNOWN`.
- `source`
  Source of the code, with the name it refers to, e.g. `"WHONET (Enterobacter aerogenes)"`, or `NA` if the source is unknown

## Download Our Reference Data

All reference data sets in the AMR package - including information on microorganisms, antimicrobials, and clinical breakpoints - are freely available for download in multiple formats: R, MS Excel, Apache Feather, Apache Parquet, SPSS, and Stata.

For maximum compatibility, we also provide machine-readable, tab-separated plain text files suitable for use in any software, including laboratory information systems.

Visit our website for direct download links, or explore the actual files in our GitHub repository.

## See Also

`as.mo()` microorganisms

## Examples

```
microorganisms.codes

# 'ECO' or 'eco' is the WHONET code for E. coli:
microorganisms.codes[microorganisms.codes$code == "ECO", ]

# and therefore, 'eco' will be understood as E. coli in this package:
mo_info("eco")

# works for all AMR functions:
mo_is_intrinsic_resistant("eco", ab = "vancomycin")
```

`microorganisms.groups` *Data Set with 466 Microorganisms In Species Groups*

## Description

A data set containing species groups and microbiological complexes, which are used in the clinical breakpoints table.

## Usage

```
microorganisms.groups
```

## Format

A tibble with 466 observations and 4 variables:

- `mo_group`
  ID of the species group / microbiological complex
- `mo`
  ID of the microorganism belonging in the species group / microbiological complex
- `mo_group_name`
  Name of the species group / microbiological complex, as retrieved with `mo_name()`
- `mo_name`
  Name of the microorganism belonging in the species group / microbiological complex, as retrieved with `mo_name()`

## Download Our Reference Data

All reference data sets in the AMR package - including information on microorganisms, antimicrobials, and clinical breakpoints - are freely available for download in multiple formats: R, MS Excel, Apache Feather, Apache Parquet, SPSS, and Stata.

For maximum compatibility, we also provide machine-readable, tab-separated plain text files suitable for use in any software, including laboratory information systems.

Visit our website for direct download links, or explore the actual files in our GitHub repository.

## See Also

`as.mo()` microorganisms

## Examples

```
microorganisms.groups

# these are all species in the Bacteroides fragilis group, as per WHONET:
microorganisms.groups[microorganisms.groups$mo_group == "B_BCTRD_FRGL-C", ]
```

`mo_matching_score` *Calculate the Matching Score for Microorganisms*

## Description

This algorithm is used by `as.mo()` and all the `mo_*` functions to determine the most probable match of taxonomic records based on user input.

## Usage

```
mo_matching_score(x, n)
```

## Arguments

| | |
|---|---|
| `x` | Any user input value(s). |
| `n` | A full taxonomic name, that exists in `microorganisms$fullname`. |

## Matching Score for Microorganisms

With ambiguous user input in `as.mo()` and all the `mo_*` functions, the returned results are chosen based on their matching score using `mo_matching_score()`. This matching score $m$, is calculated as:

$$m_{(x,n)} = \frac{l_n - 0.5 \cdot \min \begin{cases} l_n \\ \text{lev}(x, n) \end{cases}}{l_n \cdot p_n \cdot k_n}$$

where:

- $x$ is the user input;
- $n$ is a taxonomic name (genus, species, and subspecies);
- $l_n$ is the length of $n$;
- $lev$ is the Levenshtein distance function (counting any insertion as 1, and any deletion or substitution as 2) that is needed to change $x$ into $n$;
- $p_n$ is the human pathogenic prevalence group of $n$, as described below;
- $k_n$ is the taxonomic domain ('kingdom' until taxonomic reclassification of 2024) of $n$, set as Bacteria = 1, Fungi = 1.25, Protozoa = 1.5, Chromista = 1.75, Archaea = 2, others = 3.

The grouping into human pathogenic prevalence $p$ is based on recent work from Bartlett *et al.* (2022, doi:10.1099/mic.0.001269) who extensively studied medical-scientific literature to categorise all bacterial species into these groups:

- **Established**, if a taxonomic species has infected at least three persons in three or more references. These records have `prevalence = 1.15` in the microorganisms data set;
- **Putative**, if a taxonomic species has fewer than three known cases. These records have `prevalence = 1.25` in the microorganisms data set.

Furthermore,

- Genera from the World Health Organization's (WHO) Priority Pathogen List have `prevalence = 1.0` in the microorganisms data set;
- Any genus present in the **established** list also has `prevalence = 1.15` in the microorganisms data set;
- Any other genus present in the **putative** list has `prevalence = 1.25` in the microorganisms data set;
- Any other species or subspecies of which the genus is present in the two aforementioned groups, has `prevalence = 1.5` in the microorganisms data set;
- Any *non-bacterial* genus, species or subspecies of which the genus is present in the following list, has `prevalence = 1.25` in the microorganisms data set: *Absidia*, *Acanthamoeba*, *Acremonium*, *Actinomucor*, *Aedes*, *Alternaria*, *Amoeba*, *Ancylostoma*, *Angiostrongylus*, *Anisakis*, *Anopheles*, *Apiotrichum*, *Apophysomyces*, *Arthroderma*, *Ascaris*, *Aspergillus*, *Aureobasidium*, *Babesia*, *Balamuthia*, *Balantidium*, *Balantioides*, *Basidiobolus*, *Beauveria*, *Bipolaris*, *Blastobotrys*, *Blastocystis*, *Blastomyces*, *Brugia*, *Candida*, *Candidozyma*, *Capillaria*, *Chaetomium*, *Chilomastix*, *Chrysonilia*, *Chrysosporium*, *Cladophialophora*, *Cladosporium*, *Clavispora*, *Clonorchis*, *Coccidioides*, *Cokeromyces*, *Conidiobolus*, *Coniochaeta*, *Contracaecum*, *Cordylobia*, *Cryptococcus*, *Cryptosporidium*, *Cunninghamella*, *Curvularia*, *Cutaneotrichosporon*, *Cyberlindnera*, *Cyclospora*, *Cystoisospora*, *Debaryomyces*, *Debaryozyma*, *Demodex*, *Dermatobia*, *Dientamoeba*, *Diphyllobothrium*, *Dirofilaria*, *Diutina*, *Echinococcus*, *Echinostoma*, *Emergomyces*, *Emmonsia*, *Encephalitozoon*, *Entamoeba*, *Enterobius*, *Enterocytozoon*, *Epidermophyton*, *Exidia*, *Exophiala*, *Exserohilum*, *Fasciola*, *Fasciolopsis*, *Fonsecaea*, *Fusarium*, *Geotrichum*, *Giardia*, *Graphium*, *Haloarcula*, *Halobacterium*, *Halococcus*, *Hansenula*, *Hendersonula*, *Heterophyes*, *Histomonas*, *Histoplasma*, *Hortaea*, *Hymenolepis*, *Hypomyces*, *Hysterothylacium*, *Isospora*, *Kazachstania*, *Kloeckera*, *Kluyveromyces*, *Kodamaea*, *Lacazia*, *Leishmania*, *Lichtheimia*, *Loa*, *Lodderomyces*, *Lomentospora*, *Madurella*, *Magnusiomyces*, *Malassezia*, *Malbranchea*, *Mansonella*, *Metagonimus*, *Meyerozyma*, *Microascus*, *Microsporidium*, *Microsporum*, *Millerozyma*, *Mortierella*, *Mucor*, *Mycocentrospora*, *Naegleria*, *Nakaseomyces*, *Nannizzia*, *Necator*, *Nectria*, *Neocosmospora*, *Ochroconis*, *Oesophagostomum*, *Oidiodendron*, *Onchocerca*, *Opisthorchis*, *Paecilomyces*, *Paracoccidioides*, *Paragonimus*, *Pediculus*, *Penicillium*, *Phaeoacremonium*, *Phaeomoniella*, *Phialophora*, *Phlebotomus*, *Phoma*, *Pichia*, *Piedraia*, *Pithomyces*, *Pityrosporum*, *Plasmodium*, *Pneumocystis*, *Pseudallescheria*, *Pseudoscopulariopsis*, *Pseudoterranova*, *Pulex*, *Purpureocillium*, *Quambalaria*, *Rasamsonia*, *Rhinocladiella*, *Rhizomucor*, *Rhizopus*, *Rhodotorula*, *Saccharomyces*, *Saksenaea*, *Saprochaete*, *Sarcocystis*, *Sarcoptes*, *Scedosporium*, *Schistosoma*, *Schizophyllum*, *Schizosaccharomyces*, *Scolecobasidium*, *Scopulariopsis*, *Scytalidium*, *Spirometra*, *Sporobolomyces*, *Sporopachydermia*, *Sporothrix*, *Sporotrichum*, *Stachybotrys*, *Strongyloides*, *Syncephalastrum*, *Syngamus*, *Taenia*, *Talaromyces*, *Toxocara*, *Toxoplasma*, *Trichinella*, *Trichobilharzia*, *Trichoderma*, *Trichomonas*, *Trichophyton*, *Trichosporon*, *Trichostrongylus*, *Trichuris*, *Tritirachium*, *Trombicula*, *Trypanosoma*, *Tunga*, *Ulocladium*, *Ustilago*, *Verticillium*, *Wallemia*, *Wangiella*, *Wickerhamomyces*, *Wuchereria*, *Yarrowia*, or *Zygosaccharomyces*;
- All other records have `prevalence = 2.0` in the microorganisms data set.

When calculating the matching score, all characters in $x$ and $n$ are ignored that are other than A-Z, a-z, 0-9, spaces and parentheses.

All matches are sorted descending on their matching score and for all user input values, the top match will be returned. This will lead to the effect that e.g., `"E. coli"` will return the microbial

ID of *Escherichia coli* ($m = 0.688$, a highly prevalent microorganism found in humans) and not *Entamoeba coli* ($m = 0.381$, a less prevalent microorganism in humans), although the latter would alphabetically come first.

### Download Our Reference Data

All reference data sets in the AMR package - including information on microorganisms, antimicrobials, and clinical breakpoints - are freely available for download in multiple formats: R, MS Excel, Apache Feather, Apache Parquet, SPSS, and Stata.

For maximum compatibility, we also provide machine-readable, tab-separated plain text files suitable for use in any software, including laboratory information systems.

Visit our website for direct download links, or explore the actual files in our GitHub repository.

### Note

This algorithm was originally developed in 2018 and subsequently described in: Berends MS *et al.* (2022). **AMR: An R Package for Working with Antimicrobial Resistance Data**. *Journal of Statistical Software*, 104(3), 1-31; doi:10.18637/jss.v104.i03.

Later, the work of Bartlett A *et al.* about bacterial pathogens infecting humans (2022, doi:10.1099/mic.0.001269) was incorporated, and optimalisations to the algorithm were made.

### Examples

```
mo_reset_session()

as.mo("E. coli")
mo_uncertainties()

mo_matching_score(
  x = "E. coli",
  n = c("Escherichia coli", "Entamoeba coli")
)
```

---

## mo_property *Get Properties of a Microorganism*

---

### Description

Use these functions to return a specific property of a microorganism based on the latest accepted taxonomy. All input values will be evaluated internally with `as.mo()`, which makes it possible to use microbial abbreviations, codes and names as input. See *Examples*.

## Usage

```
mo_name(
  x,
  language = get_AMR_locale(),
  keep_synonyms = getOption("AMR_keep_synonyms", FALSE),
  ...
)

mo_fullname(
  x,
  language = get_AMR_locale(),
  keep_synonyms = getOption("AMR_keep_synonyms", FALSE),
  ...
)

mo_shortname(
  x,
  language = get_AMR_locale(),
  keep_synonyms = getOption("AMR_keep_synonyms", FALSE),
  ...
)

mo_subspecies(
  x,
  language = get_AMR_locale(),
  keep_synonyms = getOption("AMR_keep_synonyms", FALSE),
  ...
)

mo_species(
  x,
  language = get_AMR_locale(),
  keep_synonyms = getOption("AMR_keep_synonyms", FALSE),
  ...
)

mo_genus(
  x,
  language = get_AMR_locale(),
  keep_synonyms = getOption("AMR_keep_synonyms", FALSE),
  ...
)

mo_family(
  x,
  language = get_AMR_locale(),
  keep_synonyms = getOption("AMR_keep_synonyms", FALSE),
  ...
```

```
)

mo_order(
  x,
  language = get_AMR_locale(),
  keep_synonyms = getOption("AMR_keep_synonyms", FALSE),
  ...
)

mo_class(
  x,
  language = get_AMR_locale(),
  keep_synonyms = getOption("AMR_keep_synonyms", FALSE),
  ...
)

mo_phylum(
  x,
  language = get_AMR_locale(),
  keep_synonyms = getOption("AMR_keep_synonyms", FALSE),
  ...
)

mo_kingdom(
  x,
  language = get_AMR_locale(),
  keep_synonyms = getOption("AMR_keep_synonyms", FALSE),
  ...
)

mo_domain(
  x,
  language = get_AMR_locale(),
  keep_synonyms = getOption("AMR_keep_synonyms", FALSE),
  ...
)

mo_type(
  x,
  language = get_AMR_locale(),
  keep_synonyms = getOption("AMR_keep_synonyms", FALSE),
  ...
)

mo_status(
  x,
  language = get_AMR_locale(),
  keep_synonyms = getOption("AMR_keep_synonyms", FALSE),
```

```
  ...
)

mo_pathogenicity(
  x,
  language = get_AMR_locale(),
  keep_synonyms = getOption("AMR_keep_synonyms", FALSE),
  ...
)

mo_gramstain(
  x,
  language = get_AMR_locale(),
  keep_synonyms = getOption("AMR_keep_synonyms", FALSE),
  add_morphology = FALSE,
  ...
)

mo_is_gram_negative(
  x,
  language = get_AMR_locale(),
  keep_synonyms = getOption("AMR_keep_synonyms", FALSE),
  ...
)

mo_is_gram_positive(
  x,
  language = get_AMR_locale(),
  keep_synonyms = getOption("AMR_keep_synonyms", FALSE),
  ...
)

mo_is_yeast(
  x,
  language = get_AMR_locale(),
  keep_synonyms = getOption("AMR_keep_synonyms", FALSE),
  ...
)

mo_is_intrinsic_resistant(
  x,
  ab,
  language = get_AMR_locale(),
  keep_synonyms = getOption("AMR_keep_synonyms", FALSE),
  ...
)

mo_oxygen_tolerance(
```

```
  x,
  language = get_AMR_locale(),
  keep_synonyms = getOption("AMR_keep_synonyms", FALSE),
  ...
)

mo_is_anaerobic(
  x,
  language = get_AMR_locale(),
  keep_synonyms = getOption("AMR_keep_synonyms", FALSE),
  ...
)

mo_morphology(
  x,
  language = get_AMR_locale(),
  keep_synonyms = getOption("AMR_keep_synonyms", FALSE),
  ...
)

mo_snomed(
  x,
  language = get_AMR_locale(),
  keep_synonyms = getOption("AMR_keep_synonyms", FALSE),
  ...
)

mo_ref(
  x,
  language = get_AMR_locale(),
  keep_synonyms = getOption("AMR_keep_synonyms", FALSE),
  ...
)

mo_authors(
  x,
  language = get_AMR_locale(),
  keep_synonyms = getOption("AMR_keep_synonyms", FALSE),
  ...
)

mo_year(
  x,
  language = get_AMR_locale(),
  keep_synonyms = getOption("AMR_keep_synonyms", FALSE),
  ...
)
```

```
mo_lpsn(
  x,
  language = get_AMR_locale(),
  keep_synonyms = getOption("AMR_keep_synonyms", FALSE),
  ...
)

mo_mycobank(
  x,
  language = get_AMR_locale(),
  keep_synonyms = getOption("AMR_keep_synonyms", FALSE),
  ...
)

mo_gbif(
  x,
  language = get_AMR_locale(),
  keep_synonyms = getOption("AMR_keep_synonyms", FALSE),
  ...
)

mo_rank(
  x,
  language = get_AMR_locale(),
  keep_synonyms = getOption("AMR_keep_synonyms", FALSE),
  ...
)

mo_taxonomy(
  x,
  language = get_AMR_locale(),
  keep_synonyms = getOption("AMR_keep_synonyms", FALSE),
  ...
)

mo_synonyms(
  x,
  language = get_AMR_locale(),
  keep_synonyms = getOption("AMR_keep_synonyms", FALSE),
  ...
)

mo_current(x, language = get_AMR_locale(), ...)

mo_group_members(
  x,
  language = get_AMR_locale(),
  keep_synonyms = getOption("AMR_keep_synonyms", FALSE),
```

```
  ...
)

mo_info(
  x,
  language = get_AMR_locale(),
  keep_synonyms = getOption("AMR_keep_synonyms", FALSE),
  ...
)

mo_url(
  x,
  open = FALSE,
  language = get_AMR_locale(),
  keep_synonyms = getOption("AMR_keep_synonyms", FALSE),
  ...
)

mo_property(
  x,
  property = "fullname",
  language = get_AMR_locale(),
  keep_synonyms = getOption("AMR_keep_synonyms", FALSE),
  ...
)
```

## Arguments

| | |
|---|---|
| x | Any character (vector) that can be coerced to a valid microorganism code with as.mo(). Can be left blank for auto-guessing the column containing microorganism codes if used in a data set, see *Examples*. |
| language | Language to translate text like "no growth", which defaults to the system language (see get_AMR_locale()). |
| keep_synonyms | A logical to indicate if outdated, previously valid taxonomic names must be preserved and not be corrected to currently accepted names. Do note that the term "synonym" is in this case jargon from the field of microbial taxonomy - it is not in place to denote that e.g. "Streptococcus Group A" is a synonym of *S. pyogenes*. Though this is practically the case, taxonomically it is not as "Streptococcus Group A" is not even a valid taxonomic name.<br>The default is FALSE, which will return a note if outdated taxonomic names were processed. The default can be set with the package option AMR_keep_synonyms, i.e. options(AMR_keep_synonyms = TRUE) or options(AMR_keep_synonyms = FALSE). |
| ... | Other arguments passed on to as.mo(), such as 'minimum_matching_score', 'ignore_pattern', and 'remove_from_input'. |
| add_morphology | a logical to indicate whether the morphology (from mo_morphology()) should be added to the Gram stain result, e.g. "Gram-negative rods" instead of "Gram-negative". The default is FALSE. |

| | |
|---|---|
| ab | Any (vector of) text that can be coerced to a valid antibiotic drug code with as.ab(). |
| open | Browse the URL using browseURL(). |
| property | One of the column names of the microorganisms data set: "mo", "fullname", "status", "domain", "kingdom", "phylum", "class", "order", "family", "genus", "species", "subspecies", "rank", "ref", "oxygen_tolerance", "morphology", "source", "lpsn", "lpsn_parent", "lpsn_renamed_to", "mycobank", "mycobank_parent", "mycobank_renamed_to", "gbif", "gbif_parent", "gbif_renamed_to", "prevalence", or "snomed", or must be "shortname". |

## Details

All functions will, at default, **not** keep old taxonomic properties, as synonyms are automatically replaced with the current taxonomy. Take for example *Enterobacter aerogenes*, which was initially named in 1960 but renamed to *Klebsiella aerogenes* in 2017:

- mo_genus("Enterobacter aerogenes") will return "Klebsiella" (with a note about the renaming)
- mo_genus("Enterobacter aerogenes", keep_synonyms = TRUE) will return "Enterobacter" (with a once-per-session warning that the name is outdated)
- mo_ref("Enterobacter aerogenes") will return "Tindall et al., 2017" (with a note about the renaming)
- mo_ref("Enterobacter aerogenes", keep_synonyms = TRUE) will return "Hormaeche et al., 1960" (with a once-per-session warning that the name is outdated)

mo_ref() returns the abbreviated authority of the nomenclatural act that created the queried name combination. When keep_synonyms = FALSE (default), this is the authority of the currently accepted name. When keep_synonyms = TRUE, this is the authority under which the queried (possibly outdated) name was published. Emendations (changes to the species description without a name change) are not reflected; only the combination or original description authority is returned.

The short name (mo_shortname()) returns the first character of the genus and the full species, such as "E. coli", for species and subspecies. Exceptions are abbreviations of staphylococci (such as *"CoNS"*, Coagulase-Negative Staphylococci) and beta-haemolytic streptococci (such as *"GBS"*, Group B Streptococci). Please bear in mind that e.g. *E. coli* could mean *Escherichia coli* (domain of Bacteria) as well as *Entamoeba coli* (domain of Protozoa). Returning to the full name will be done using as.mo() internally, giving priority to bacteria and human pathogens, i.e. "E. coli" will always be considered *Escherichia coli*. As a result, mo_fullname(mo_shortname("Entamoeba coli")) returns "Escherichia coli".

Following the formal introduction of the new kingdom rank into prokaryotic nomenclature in 2024 (doi:10.1099/ijsem.0.006242), mo_kingdom() and mo_domain() return different results for bacteria and archaea: mo_kingdom() returns the new formal kingdom (e.g. "Pseudomonadati", "Bacillati"), while mo_domain() returns the new domain (e.g. "Bacteria", "Archaea"). For non-prokaryotic organisms, both functions return identical results.

Determination of human pathogenicity (mo_pathogenicity()) is strongly based on Bartlett *et al.* (2022, doi:10.1099/mic.0.001269). This function returns a factor with the levels *Pathogenic*, *Potentially pathogenic*, *Non-pathogenic*, and *Unknown*.

Determination of the Gram stain (mo_gramstain()) is based on the taxonomic domain and phylum. Originally, Cavalier-Smith defined the so-called subkingdoms Negibacteria and Posibacteria (2002, doi:10.1099/002077135217), and only considered these phyla as Posibacteria: Actinobacteria, Chloroflexi, Firmicutes, and Tenericutes. These phyla were later renamed to Actinomycetota, Chloroflexota, Bacillota, and Mycoplasmatota (2021, doi:10.1099/ijsem.0.005056). Bacteria in these phyla are considered Gram-positive in this AMR package, except for members of the class Negativicutes (within phylum Bacillota) which are Gram-negative. All other bacteria are considered Gram-negative. Species outside the kingdom of Bacteria will return a value NA. Functions mo_is_gram_negative() and mo_is_gram_positive() always return TRUE or FALSE (or NA when the input is NA or the MO code is UNKNOWN), thus always return FALSE for species outside the taxonomic kingdom of Bacteria.

Determination of yeasts (mo_is_yeast()) is based on the taxonomic domain and class. *Budding yeasts* are yeasts that reproduce asexually through a process called budding, where a new cell develops from a small protrusion on the parent cell. Taxonomically, these are members of the phylum Ascomycota, class Saccharomycetes (also called Hemiascomycetes) or Pichiomycetes. *True yeasts* quite specifically refers to yeasts in the underlying order Saccharomycetales (such as *Saccharomyces cerevisiae*). Thus, for all microorganisms that are member of the taxonomic class Saccharomycetes or Pichiomycetes, the function will return TRUE. It returns FALSE otherwise (or NA when the input is NA or the MO code is UNKNOWN).

Determination of intrinsic resistance (mo_is_intrinsic_resistant()) is based on the intrinsic_resistant data set, which is based on 'EUCAST Expected Resistant Phenotypes' v1.2 (2023). The mo_is_intrinsic_resistant() function can be vectorised over both argument x (input for microorganisms) and ab (input for antimicrobials).

Determination of both bacterial oxygen tolerance (mo_oxygen_tolerance()) and morphology (mo_morphology()) are based on BacDive, see *Source*. The function mo_is_anaerobic() only returns TRUE if the oxygen tolerance is "anaerobe", indicating an obligate anaerobic species or genus. It always returns FALSE for species outside the taxonomic domain of Bacteria.

The function mo_url() will return the direct URL to the online database entry, which also shows the scientific reference of the concerned species. This MycoBank URL is used for fungi wherever available , this LPSN URL for bacteria wherever available, and this GBIF link otherwise.

SNOMED codes (mo_snomed()) was last updated on 16th of July, 2024. See *Source* and the microorganisms data set for more info.

Old taxonomic names (so-called 'synonyms') can be retrieved with mo_synonyms() (which will have the scientific reference as name), the current taxonomic name can be retrieved with mo_current(). Both functions return full names.

All output will be translated where possible.

## Value

- An integer in case of mo_year()
- An ordered factor in case of mo_pathogenicity()
- A list in case of mo_taxonomy(), mo_synonyms(), mo_snomed(), and mo_info()
- A logical in case of mo_is_anaerobic(), mo_is_gram_negative(), mo_is_gram_positive(), mo_is_intrinsic_resistant(), and mo_is_yeast()
- A named character in case of mo_synonyms() and mo_url()
- A character in all other cases

### Matching Score for Microorganisms

This function uses `as.mo()` internally, which uses an advanced algorithm to translate arbitrary user input to valid taxonomy using a so-called matching score. You can read about this public algorithm on the MO matching score page.

### Source


- Berends MS *et al.* (2022). **AMR: An R Package for Working with Antimicrobial Resistance Data**. *Journal of Statistical Software*, 104(3), 1-31; doi:10.18637/jss.v104.i03
- Freese, HM *et al.* (2026). **TYGS and LPSN in 2025: a Global Core Biodata Resource for genome-based classification and nomenclature of prokaryotes within DSMZ Digital Diversity.** Nucleic Acids Research, 54, D884–D891; doi:10.1093/nar/gkaf1110. Accessed from `https://lpsn.dsmz.de` on 5th of October, 2026.
- Vincent, R *et al* (2013). **MycoBank gearing up for new horizons.** IMA Fungus, 4(2), 371-9; doi:10.5598/imafungus.2013.04.02.16. Accessed from `https://www.mycobank.org` on 7th of January, 2026.
- Banki, O. *et al.* (2026). Catalogue of Life (2026-04-18 XR). Catalogue of Life Foundation, Amsterdam, Netherlands. doi:10.48580/dgxjw. Accessed from `https://www.gbif.org` on 30th of April, 2026.
- Reimer, LC *et al.* (2022). ***BacDive* in 2022: the knowledge base for standardized bacterial and archaeal data.** Nucleic Acids Res., 50(D1):D741-D74; doi:10.1093/nar/gkab961. Accessed from `https://bacdive.dsmz.de` on 5th of October, 2026.
- Public Health Information Network Vocabulary Access and Distribution System (PHIN VADS). US Edition of SNOMED CT from 1 September 2020. Value Set Name 'Microorganism', OID 2.16.840.1.114222.4.11.1009 (v12). URL: `https://www.cdc.gov/phin/php/phinvads/`
- Bartlett A *et al.* (2022). **A comprehensive list of bacterial pathogens infecting humans** *Microbiology* 168:001269; doi:10.1099/mic.0.001269


### Download Our Reference Data

All reference data sets in the AMR package - including information on microorganisms, antimicrobials, and clinical breakpoints - are freely available for download in multiple formats: R, MS Excel, Apache Feather, Apache Parquet, SPSS, and Stata.

For maximum compatibility, we also provide machine-readable, tab-separated plain text files suitable for use in any software, including laboratory information systems.

Visit our website for direct download links, or explore the actual files in our GitHub repository.

### See Also

Data set microorganisms

### Examples

```
# taxonomic tree ---------------------------------------------------------

mo_domain("Klebsiella pneumoniae")
mo_kingdom("Klebsiella pneumoniae")
```

```
mo_phylum("Klebsiella pneumoniae")
mo_class("Klebsiella pneumoniae")
mo_order("Klebsiella pneumoniae")
mo_family("Klebsiella pneumoniae")
mo_genus("Klebsiella pneumoniae")
mo_species("Klebsiella pneumoniae")
mo_subspecies("Klebsiella pneumoniae")

# all in one go
mo_taxonomy("Klebsiella pneumoniae")

# full names and short names ----------------------------------------------

mo_name("Klebsiella pneumoniae")
mo_fullname("Klebsiella pneumoniae")
mo_shortname("Klebsiella pneumoniae")


# other properties --------------------------------------------------------

mo_morphology("Klebsiella pneumoniae")
mo_gramstain("Klebsiella pneumoniae")
mo_gramstain("Klebsiella pneumoniae", add_morphology = TRUE)
mo_pathogenicity("Klebsiella pneumoniae")
mo_snomed("Klebsiella pneumoniae")
mo_type("Klebsiella pneumoniae")
mo_rank("Klebsiella pneumoniae")
mo_url("Klebsiella pneumoniae")
mo_is_yeast(c("Candida", "Trichophyton", "Klebsiella"))

mo_group_members("Streptococcus group A")
mo_group_members(c(
  "Streptococcus group A",
  "Streptococcus group C",
  "Streptococcus group G",
  "Streptococcus group L"
))


# scientific reference ----------------------------------------------------

mo_ref("Klebsiella aerogenes")
mo_authors("Klebsiella aerogenes")
mo_year("Klebsiella aerogenes")
mo_synonyms("Klebsiella aerogenes")
mo_lpsn("Klebsiella aerogenes")
mo_gbif("Klebsiella aerogenes")
mo_mycobank("Candida albicans")
mo_mycobank("Candida krusei")
mo_mycobank("Candida krusei", keep_synonyms = TRUE)


# abbreviations known in the field ----------------------------------------
```

```
mo_genus("MRSA")
mo_species("MRSA")
mo_shortname("VISA")
mo_gramstain("VISA")

mo_genus("EHEC")
mo_species("EIEC")
mo_name("UPEC")


# known subspecies ------------------------------------------------------

mo_fullname("K. pneu rh")
mo_shortname("K. pneu rh")


# Becker classification, see ?as.mo -------------------------------------

mo_fullname("Staph epidermidis")
mo_fullname("Staph epidermidis", Becker = TRUE)

mo_shortname("Staph epidermidis")
mo_shortname("Staph epidermidis", Becker = TRUE)


# Lancefield classification, see ?as.mo ---------------------------------

mo_fullname("Strep agalactiae")
mo_fullname("Strep agalactiae", Lancefield = TRUE)

mo_shortname("Strep agalactiae")
mo_shortname("Strep agalactiae", Lancefield = TRUE)


# language support  -----------------------------------------------------

mo_gramstain("Klebsiella pneumoniae", language = "de") # German
mo_gramstain("Klebsiella pneumoniae", language = "nl") # Dutch
mo_gramstain("Klebsiella pneumoniae", language = "es") # Spanish
mo_gramstain("Klebsiella pneumoniae", language = "el") # Greek
mo_gramstain("Klebsiella pneumoniae", language = "uk") # Ukrainian

# mo_type is equal to mo_domain, but mo_domain will remain untranslated
mo_domain("Klebsiella pneumoniae")
mo_type("Klebsiella pneumoniae")
mo_domain("Klebsiella pneumoniae", language = "zh") # Chinese, no effect
mo_type("Klebsiella pneumoniae", language = "zh") # Chinese, translated

mo_fullname("S. pyogenes", Lancefield = TRUE, language = "de")
mo_fullname("S. pyogenes", Lancefield = TRUE, language = "uk")
```

```
# other ----------------------------------------------------------------------

# gram stains and intrinsic resistance can be used as a filter in dplyr verbs
if (require("dplyr")) {
  example_isolates %>%
    filter(mo_is_gram_positive()) %>%
    count(mo_genus(), sort = TRUE)
}
if (require("dplyr")) {
  example_isolates %>%
    filter(mo_is_intrinsic_resistant(ab = "vanco")) %>%
    count(mo_genus(), sort = TRUE)
}

# get a list with the complete taxonomy (from kingdom to subspecies)
mo_taxonomy("Klebsiella pneumoniae")

# get a list with the taxonomy, the authors, Gram-stain,
# SNOMED codes, and URL to the online database
mo_info("Klebsiella pneumoniae")
```

---

`mo_source` *User-Defined Reference Data Set for Microorganisms*

---

## Description

These functions can be used to predefine your own reference to be used in `as.mo()` and consequently all `mo_*` functions (such as `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, since you don't have to bother about it again after setting it up once.

## Usage

```
set_mo_source(
  path,
  destination = getOption("AMR_mo_source", "~/mo_source.rds")
)

get_mo_source(destination = getOption("AMR_mo_source", "~/mo_source.rds"))
```

## Arguments

| | |
|---|---|
| `path` | Location of your reference file, this can be any text file (comma-, tab- or pipe-separated) or an Excel file (see *Details*). Can also be `""`, `NULL` or `FALSE` to delete the reference file. |
| `destination` | Destination of the compressed data file - the default is the user's home directory. |

## 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 will 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$mo` or `microorganisms$fullname` 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 will ask to export it to `"~/mo_source.rds"`. The CRAN policy disallows packages to write to the file system, although '*exceptions may be allowed in interactive sessions if the package obtains confirmation from the user*'. For this reason, this function only works in interactive sessions so that the user can **specifically confirm and allow** that this file will be created. The destination of this file can be set with the `destination` argument and defaults to the user's home directory. It can also be set with the package option `AMR_mo_source`, e.g. `options(AMR_mo_source = "my/location/file.rds")`.

The created compressed data file `"mo_source.rds"` will be used at default for MO determination (function `as.mo()` and consequently all `mo_*` functions like `mo_genus()` and `mo_gramstain()`). The location and timestamp of the original file will be saved as an attribute to the compressed data file.

The function `get_mo_source()` will return the data set by reading `"mo_source.rds"` with `readRDS()`. If the original file has changed (by checking the location and timestamp of the original file), it will call `set_mo_source()` to update the data file automatically if used in an interactive session.

Reading an Excel file (`.xlsx`) with only one row has a size of 8-9 kB. The compressed file created with `set_mo_source()` will then have a size of 0.1 kB and can be read by `get_mo_source()` in only a couple of microseconds (millionths of a second).

## How to Setup

Imagine this data on a sheet of an Excel file. The first column contains the organisation specific codes, the second column contains valid taxonomic names:

```
  |         A          |            B          |
--|--------------------|-----------------------|
1 | Organisation XYZ   | mo                    |
2 | lab_mo_ecoli       | Escherichia coli      |
3 | lab_mo_kpneumoniae | Klebsiella pneumoniae |
4 |                    |                       |
```

We save it as `"/Users/me/Documents/ourcodes.xlsx"`. Now we have to set it as a source:

```
set_mo_source("/Users/me/Documents/ourcodes.xlsx")
#> NOTE: Created mo_source file '/Users/me/mo_source.rds' (0.3 kB) from
#>       '/Users/me/Documents/ourcodes.xlsx' (9 kB), columns
#>       "Organisation XYZ" and "mo"
```

It has now created a file `"~/mo_source.rds"` with the contents of our Excel file. Only the first column with foreign values and the 'mo' column will be kept when creating the RDS file.

And now we can use it in our functions:

```
as.mo("lab_mo_ecoli")
#> Class <mo>
#> [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"))
#> NOTE: Translation to one microorganism was guessed with uncertainty.
#>       Use mo_uncertainties() to review it.
#> Class <mo>
#> [1] B_ESCHR_COLI B_ESCHR_COLI B_ESCHR_COLI
```

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

```
  |         A          |            B          |
--|--------------------|-----------------------|
1 | Organisation XYZ   | mo                    |
2 | lab_mo_ecoli       | Escherichia coli      |
3 | lab_mo_kpneumoniae | Klebsiella pneumoniae |
4 | lab_Staph_aureus   | Staphylococcus aureus |
5 |                    |                       |
```

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

```
as.mo("lab_mo_ecoli")
#> NOTE: Updated mo_source file '/Users/me/mo_source.rds' (0.3 kB) from
#>       '/Users/me/Documents/ourcodes.xlsx' (9 kB), columns
#>        "Organisation XYZ" and "mo"
#> Class <mo>
#> [1] B_ESCHR_COLI

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

To delete the reference data file, just use `""`, `NULL` or `FALSE` as input for `set_mo_source()`:

```
set_mo_source(NULL)
#> Removed mo_source file '/Users/me/mo_source.rds'
```

If the original file (in the previous case an Excel file) is moved or deleted, the `mo_source.rds` file will be removed upon the next use of `as.mo()` or any `mo_*` function.

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

```
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, can be unquoted since it supports quasiquotation. |
| 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. | 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 | 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. | 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

## Examples

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


if (require("dplyr")) {
  # calculate the resistance per group first
  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;
    filter(n() >= 30) %>% # filter on only 30 results per group
    summarise_if(is.sir, resistance) # then get resistance of all drugs

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

  pca_result
  summary(pca_result)
  # old base R plotting method:
  biplot(pca_result)
}

# new ggplot2 plotting method using this package:
if (require("dplyr") && require("ggplot2")) {
  ggplot_pca(pca_result)
}
if (require("dplyr") && require("ggplot2")) {
  ggplot_pca(pca_result) +
    scale_colour_viridis_d() +
    labs(title = "Title here")
}
```

plot *Plotting Helpers for AMR Data Analysis*

## Description

Functions to plot classes `sir`, `mic` and `disk`, with support for base R and `ggplot2`.

Especially the `scale_*_mic()` functions are relevant wrappers to plot MIC values for `ggplot2`. They allow custom MIC ranges and to plot intermediate log2 levels for missing MIC values.

## Usage

```
scale_x_mic(keep_operators = "edges", mic_range = NULL, ...)

scale_y_mic(keep_operators = "edges", mic_range = NULL, ...)

scale_colour_mic(keep_operators = "edges", mic_range = NULL, ...)

scale_fill_mic(keep_operators = "edges", mic_range = NULL, ...)

scale_x_sir(
  colours_SIR = c(S = "#3CAEA3", SDD = "#8FD6C4", I = "#F6D55C", R = "#ED553B"),
  language = get_AMR_locale(),
 eucast_I = plot_name_of_I(getOption("AMR_guideline", "EUCAST")) != "Intermediate",
  ...
)

scale_colour_sir(
  colours_SIR = c(S = "#3CAEA3", SDD = "#8FD6C4", I = "#F6D55C", R = "#ED553B"),
  language = get_AMR_locale(),
 eucast_I = plot_name_of_I(getOption("AMR_guideline", "EUCAST")) != "Intermediate",
  ...
)

scale_fill_sir(
  colours_SIR = c(S = "#3CAEA3", SDD = "#8FD6C4", I = "#F6D55C", R = "#ED553B"),
  language = get_AMR_locale(),
 eucast_I = plot_name_of_I(getOption("AMR_guideline", "EUCAST")) != "Intermediate",
  ...
)

## S3 method for class 'mic'
plot(
  x,
  mo = NULL,
  ab = NULL,
  guideline = getOption("AMR_guideline", "EUCAST"),
```

```
  main = deparse(substitute(x)),
  ylab = translate_AMR("Frequency", language = language),
 xlab = translate_AMR("Minimum Inhibitory Concentration (mg/L)", language = language),
  colours_SIR = c(S = "#3CAEA3", SDD = "#8FD6C4", I = "#F6D55C", R = "#ED553B"),
  language = get_AMR_locale(),
  expand = TRUE,
  include_PKPD = getOption("AMR_include_PKPD", TRUE),
  breakpoint_type = getOption("AMR_breakpoint_type", "human"),
  ...
)

## S3 method for class 'mic'
autoplot(
  object,
  mo = NULL,
  ab = NULL,
  guideline = getOption("AMR_guideline", "EUCAST"),
  title = deparse(substitute(object)),
  ylab = translate_AMR("Frequency", language = language),
 xlab = translate_AMR("Minimum Inhibitory Concentration (mg/L)", language = language),
  colours_SIR = c(S = "#3CAEA3", SDD = "#8FD6C4", I = "#F6D55C", R = "#ED553B"),
  language = get_AMR_locale(),
  expand = TRUE,
  include_PKPD = getOption("AMR_include_PKPD", TRUE),
  breakpoint_type = getOption("AMR_breakpoint_type", "human"),
  ...
)

## S3 method for class 'disk'
plot(
  x,
  main = deparse(substitute(x)),
  ylab = translate_AMR("Frequency", language = language),
  xlab = translate_AMR("Disk diffusion diameter (mm)", language = language),
  mo = NULL,
  ab = NULL,
  guideline = getOption("AMR_guideline", "EUCAST"),
  colours_SIR = c(S = "#3CAEA3", SDD = "#8FD6C4", I = "#F6D55C", R = "#ED553B"),
  language = get_AMR_locale(),
  expand = TRUE,
  include_PKPD = getOption("AMR_include_PKPD", TRUE),
  breakpoint_type = getOption("AMR_breakpoint_type", "human"),
  ...
)

## S3 method for class 'disk'
autoplot(
  object,
```

```
  mo = NULL,
  ab = NULL,
  title = deparse(substitute(object)),
  ylab = translate_AMR("Frequency", language = language),
  xlab = translate_AMR("Disk diffusion diameter (mm)", language = language),
  guideline = getOption("AMR_guideline", "EUCAST"),
  colours_SIR = c(S = "#3CAEA3", SDD = "#8FD6C4", I = "#F6D55C", R = "#ED553B"),
  language = get_AMR_locale(),
  expand = TRUE,
  include_PKPD = getOption("AMR_include_PKPD", TRUE),
  breakpoint_type = getOption("AMR_breakpoint_type", "human"),
  ...
)

## S3 method for class 'sir'
plot(
  x,
  ylab = translate_AMR("Percentage", language = language),
  xlab = translate_AMR("Antimicrobial Interpretation", language = language),
  main = deparse(substitute(x)),
  language = get_AMR_locale(),
  ...
)

## S3 method for class 'sir'
autoplot(
  object,
  title = deparse(substitute(object)),
  xlab = translate_AMR("Antimicrobial Interpretation", language = language),
  ylab = translate_AMR("Frequency", language = language),
  colours_SIR = c(S = "#3CAEA3", SDD = "#8FD6C4", I = "#F6D55C", R = "#ED553B"),
  language = get_AMR_locale(),
  ...
)

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

scale_y_percent(
  breaks = function(x) seq(0, max(x, na.rm = TRUE), 0.1),
  limits = c(0, NA)
)

scale_sir_colours(
  ...,
  aesthetics,
  colours_SIR = c(S = "#3CAEA3", SDD = "#8FD6C4", I = "#F6D55C", R = "#ED553B")
)
```

```
theme_sir()
```

## Arguments

| | |
|---|---|
| keep_operators | A character specifying how to handle operators (such as > and <=) in the input. Accepts one of three values: "all" (or TRUE) to keep all operators, "none" (or FALSE) to remove all operators, or "edges" to keep operators only at both ends of the range. |
| mic_range | A manual range to rescale the MIC values (using rescale_mic()), e.g., mic_range = c(0.001, 32). Use NA to prevent rescaling on one side, e.g., mic_range = c(NA, 32). **Note:** This rescales values but does not filter them - use the ggplot2 limits argument separately to exclude values from the plot. |
| ... | Arguments passed on to methods. |
| colours_SIR | Colours to use for filling in the bars, must be a vector of one, three or four values (in the order S, SDD, I and R), or a named vector with any of the names S, SDD, I, R, SI and IR. Missing names S, I and R get the default colour, and with three values or without a name SDD, SDD gets the colour of S. The default colours are colour-blind friendly. |
| language | Language to be used to translate 'Susceptible', 'Increased exposure'/'Intermediate' and 'Resistant' - the default is system language (see get_AMR_locale()) and can be overwritten by setting the package option AMR_locale, e.g. options(AMR_locale = "de"), see translate. Use language = NULL to prevent translation. |
| eucast_I | A logical to indicate whether the 'I' must be interpreted as "Susceptible, under increased exposure". Will be TRUE if the default AMR interpretation guideline is set to EUCAST 2019 or later (which is the default). With FALSE, it will be interpreted as "Intermediate". |
| x, object | Values created with as.mic(), as.disk() or as.sir() (or their random_* variants, such as random_mic()). |
| 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 drug code with as.ab(). |
| guideline | Interpretation guideline to use - the default is the latest included EUCAST guideline, see *Details*. |
| main, title | Title of the plot. |
| xlab, ylab | Axis title. |
| expand | A logical to indicate whether the range on the x axis should be expanded between the lowest and highest value. For MIC values, intermediate values will be factors of 2 starting from the highest MIC value. For disk diameters, the whole diameter range will be filled. |
| include_PKPD | A logical to indicate that PK/PD clinical breakpoints must be applied as a last resort - the default is TRUE. Can also be set with the package option AMR_include_PKPD. |
| breakpoint_type | The type of breakpoints to use, either "ECOFF", "animal", or "human". ECOFF stands for Epidemiological Cut-Off values. The default is "human", which can |

| | |
|---|---|
| | also be set with the package option AMR_breakpoint_type. If host is set to values of veterinary species, this will automatically be set to "animal". |
| facet | Variable to split plots by, either "interpretation" (default) or "antibiotic" or a grouping variable. |
| nrow | (when using facet) number of rows. |
| breaks | A numeric vector of positions. |
| limits | A numeric vector of length two providing limits of the scale, use NA to refer to the existing minimum or maximum. |
| aesthetics | Aesthetics to apply the colours to - the default is "fill" but can also be (a combination of) "alpha", "colour", "fill", "linetype", "shape" or "size". |

## Details

### The scale_*_mic() Functions:

The functions scale_x_mic(), scale_y_mic(), scale_colour_mic(), and scale_fill_mic() allow to plot the mic class (MIC values) on a continuous, logarithmic scale.

There is normally no need to add these scale functions to your plot, as they are applied automatically when plotting values of class mic.

When manually added though, they allow to rescale the MIC range with an 'inside' or 'outside' range if required, and provide the option to retain the operators in MIC values (such as >=). Missing intermediate log2 levels will always be plotted too.

### The scale_*_sir() Functions:

The functions scale_x_sir(), scale_colour_sir(), and scale_fill_sir() allow to plot the sir class in the right order (S < SDD < I < R < NI < WT < NWT < NS).

There is normally no need to add these scale functions to your plot, as they are applied automatically when plotting values of class sir.

At default, they translate the S/I/R values to an interpretative text ("Susceptible", "Resistant", etc.) in any of the 28 supported languages (use language = NULL to keep S/I/R). Also, except for scale_x_sir(), they set colour-blind friendly colours to the colour and fill aesthetics.

### Additional ggplot2 Functions:

This package contains more functions that extend the ggplot2 package, to help in visualising AMR data results. All these functions are internally used by ggplot_sir() too.

- facet_sir() 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_y_continuous().
- scale_sir_colours() allows to set colours to any aesthetic, even for shape or linetype.
- theme_sir() is a ggplot2 theme with minimal distraction.
- labels_sir_count() prints data labels on the bars with percentage and number of isolates, using ggplot2::geom_text().

The interpretation of "I" will be named "Susceptible, incr. exp." (susceptible, increased exposure) for all EUCAST guidelines since 2019, and will be named "Intermediate" in all other cases.

For interpreting MIC values as well as disk diffusion diameters, the default guideline is EUCAST 2026, unless the package option AMR_guideline is set. See as.sir() for more information.

## Value

The `autoplot()` functions return a `ggplot` model that is extendible with any `ggplot2` function.

## Examples

```
some_mic_values <- random_mic(size = 100)
some_disk_values <- random_disk(size = 100, mo = "Escherichia coli", ab = "cipro")
some_sir_values <- random_sir(50, prob_SIR = c(0.55, 0.05, 0.30))


# Plotting using ggplot2's autoplot() for MIC, disk, and SIR -----------
if (require("ggplot2")) {
  autoplot(some_mic_values)
}
if (require("ggplot2")) {
  # when providing the microorganism and antibiotic, colours will show interpretations:
  autoplot(some_mic_values, mo = "Escherichia coli", ab = "cipro")
}
if (require("ggplot2")) {
  autoplot(some_mic_values, mo = "Staph aureus", ab = "Ceftaroline", guideline = "CLSI")
}

if (require("ggplot2")) {
  # support for all languages in ?translate, various guidelines, and many options
  autoplot(some_disk_values,
    mo = "Escherichia coli", ab = "cipro",
    guideline = "CLSI", language = "no",
    title = "Disk diffusion from the North"
  )
}


# Plotting using scale_x_mic() ----------------------------------------
if (require("ggplot2")) {
  mic_plot <- ggplot(
    data.frame(
      mics = as.mic(c(0.25, "<=4", 4, 8, 32, ">=32")),
      counts = c(1, 1, 2, 2, 3, 3)
    ),
    aes(mics, counts)
  ) +
    geom_col()
  mic_plot +
    labs(title = "scale_x_mic() automatically applied")
}
if (require("ggplot2")) {
  mic_plot +
    scale_x_mic(keep_operators = "none") +
    labs(title = "with scale_x_mic() keeping no operators")
}
if (require("ggplot2")) {
  mic_plot +
```

```
    scale_x_mic(mic_range = c(1, 16)) +
    labs(title = "with scale_x_mic() using a manual 'within' range")
}
if (require("ggplot2")) {
  mic_plot +
    scale_x_mic(mic_range = c(0.032, 256)) +
    labs(title = "with scale_x_mic() using a manual 'outside' range")
}


# Plotting using scale_y_mic() ----------------------------------------
some_groups <- sample(LETTERS[1:5], 20, replace = TRUE)

if (require("ggplot2")) {
  ggplot(
    data.frame(
      mic = some_mic_values,
      group = some_groups
    ),
    aes(group, mic)
  ) +
    geom_boxplot() +
    geom_violin(linetype = 2, colour = "grey30", fill = NA) +
    labs(title = "scale_y_mic() automatically applied")
}
if (require("ggplot2")) {
  ggplot(
    data.frame(
      mic = some_mic_values,
      group = some_groups
    ),
    aes(group, mic)
  ) +
    geom_boxplot() +
    geom_violin(linetype = 2, colour = "grey30", fill = NA) +
    scale_y_mic(mic_range = c(NA, 0.25))
}


# Plotting using scale_x_sir() ----------------------------------------
if (require("ggplot2")) {
  ggplot(
    data.frame(
      x = c("I", "R", "S"),
      y = c(45, 323, 573)
    ),
    aes(x, y)
  ) +
    geom_col() +
    scale_x_sir()
}
```

```
# Plotting using scale_y_mic() and scale_colour_sir() ------------------
if (require("ggplot2")) {
  mic_sir_plot <- ggplot(
    data.frame(
      mic = some_mic_values,
      group = some_groups,
      sir = as.sir(some_mic_values,
        mo = "E. coli",
        ab = "cipro"
      )
    ),
    aes(x = group, y = mic, colour = sir)
  ) +
    theme_minimal() +
    geom_boxplot(fill = NA, colour = "grey30") +
    geom_jitter(width = 0.25) +
    labs(title = "scale_y_mic()/scale_colour_sir() automatically applied")

  mic_sir_plot
}
if (require("ggplot2")) {
  mic_sir_plot +
    scale_y_mic(mic_range = c(0.005, 32), name = "Our MICs!") +
    scale_colour_sir(
      language = "pt", # Portuguese
      name = "Support in many languages"
    )
}


# Plotting using base R's plot() ---------------------------------------

plot(some_mic_values)
# when providing the microorganism and antibiotic, colours will show interpretations:
plot(some_mic_values, mo = "S. aureus", ab = "ampicillin")

plot(some_disk_values)
plot(some_disk_values, mo = "Escherichia coli", ab = "cipro")
plot(some_disk_values, mo = "Escherichia coli", ab = "cipro", language = "nl")

plot(some_sir_values)
```

---

`proportion` *Calculate Antimicrobial Resistance*

---

## Description

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()` from the `dplyr` package and also support grouped variables, 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,
  guideline = getOption("AMR_guideline", "EUCAST")
)

susceptibility(
  ...,
  minimum = 30,
  as_percent = FALSE,
  only_all_tested = FALSE,
  guideline = getOption("AMR_guideline", "EUCAST")
)

sir_confidence_interval(
  ...,
  ab_result = "R",
  minimum = 30,
  as_percent = FALSE,
  only_all_tested = FALSE,
  confidence_level = 0.95,
  side = "both",
  collapse = 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_AMR_locale(),
  minimum = 30,
  as_percent = FALSE,
```

```
  combine_SI,
  confidence_level = 0.95,
  guideline = getOption("AMR_guideline", "EUCAST")
)

sir_df(
  data,
  translate_ab = "name",
  language = get_AMR_locale(),
  minimum = 30,
  as_percent = FALSE,
  combine_SI,
  confidence_level = 0.95,
  guideline = getOption("AMR_guideline", "EUCAST")
)
```

## Arguments

| | |
|---|---|
| `...` | One or more vectors (or columns) with antibiotic interpretations. They will be transformed internally with `as.sir()` 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 antimicrobials, see section *Combination Therapy* below. Use `sir_combine()` to create a new variable for a combination therapy. |
| `guideline` | A guideline name, `"EUCAST"` or `"CLSI"` (or any version thereof, e.g. `"CLSI 2025"`), to determine whether the 'I' category is considered susceptible (EUCAST) or resistant (CLSI). The default is the package option `AMR_guideline`, or `"EUCAST"` if it is not set. For functions that return results per category, i.e., `count_df()`, `proportion_df()`, `sir_df()`, `ggplot_sir()`, `geom_sir()`, and `labels_sir_count()`, this can also be `NULL` or `FALSE` to keep S, I, and R separate. See section *Interpretation of SIR* below. |
| `ab_result` | Antibiotic results to test against, must be one or more values of "S", "SDD", "I", or "R". |
| `confidence_level` | The confidence level for the returned confidence interval. For the calculation, the number of S or SI isolates, and R isolates are compared with the total number of available isolates with R, S, or I by using `binom.test()`, i.e., the Clopper-Pearson method. |

| | |
|---|---|
| side | The side of the confidence interval to return. The default is "both" for a length 2 vector, but can also be (abbreviated as) "min"/"left"/"lower"/"less" or "max"/"right"/"higher"/"greater". |
| collapse | A logical to indicate whether the output values should be 'collapsed', i.e. be merged together into one value, or a character value to use for collapsing. |
| data | A data.frame containing columns with class sir (see as.sir()). |
| translate_ab | A column name of the antimicrobials data set to translate the antibiotic abbreviations to, using ab_property(). |
| language | Language of the returned text - the default is the current system language (see get_AMR_locale()) and can also be set with the package option AMR_locale. Use language = NULL or language = "" to prevent translation. |
| combine_SI | *(deprecated in favour of* guideline*)* A logical. If set, it takes precedence over guideline: TRUE merges S, SDD, and I into "SI" vs. "R", and FALSE keeps S, I, and R separate. |

## Details

For a more automated and comprehensive analysis, consider using antibiogram() or wisca(), which streamline many aspects of susceptibility reporting and, importantly, also support WISCA. The functions described here offer a more hands-on, manual approach for greater customisation.

**Remember that you should filter your data 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 with one of the four available algorithms.

The function resistance() is equal to the function proportion_R(). The function susceptibility() is equal to the function proportion_SI(). Since AMR v3.0, proportion_SI() and proportion_I() include dose-dependent susceptibility ('SDD').

Use sir_confidence_interval() to calculate the confidence interval, which relies on binom.test(), i.e., the Clopper-Pearson method. This function returns a vector of length 2 at default for antimicrobial *resistance*. Change the side argument to "left"/"min" or "right"/"max" to return a single value, and change the ab_result argument to e.g. c("S", "I") to test for antimicrobial *susceptibility*, see Examples.

These functions are not meant to count isolates, but to calculate the proportion of resistance/susceptibility. Use the count_*() functions 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 dependent on the* minimum *argument).*

The function proportion_df() takes any variable from data that has an sir class (created with as.sir()) and calculates the proportions S, I, and R. It also supports grouped variables. The function sir_df() works exactly like proportion_df(), but adds the number of isolates.

## Value

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

### Interpretation of SIR

In 2019, the European Committee on Antimicrobial Susceptibility Testing (EUCAST) has decided to change the definitions of susceptibility testing categories S, I, and R (https://www.eucast.org/bacteria/clinical-breakpoints-and-interpretation/definition-of-s-i-and-r/).

In aggregated analyses, such as proportions, counts, antibiograms, and plots, the 'I' category must be considered either susceptible or resistant. In epidemiological AMR data analysis, this follows the guideline: EUCAST considers 'I' susceptible, while CLSI considers 'I' resistant. The dose-dependent susceptibility of CLSI ('SDD') is always considered susceptible, as it equals what EUCAST defines as 'I'. All functions with a `guideline` argument for this purpose use these categories:

| `guideline` | Susceptible | Resistant |
|---|---|---|
| `"EUCAST"` | S, SDD, I, WT | R, NWT, NS |
| `"CLSI"` | S, SDD, WT | I, R, NWT, NS |

The values 'WT' (wild type) and 'NWT' (non-wild type) result from epidemiological cut-off values (ECOFFs) instead of clinical breakpoints. Since they are no clinical categories, 'WT' is always considered susceptible and 'NWT' always resistant, regardless of the guideline. The value 'NS' (non-susceptible) is always considered resistant, and 'NI' (non-interpretable) is always considered a missing result.

The `guideline` argument can be:

- **Left unset**, to use the package option `AMR_guideline`, or `"EUCAST"` if this option is not set. In the latter case, a message is shown once per session (but not in R Markdown or Quarto documents); set the option to silence it.
- `"EUCAST"`, or any EUCAST version such as `"EUCAST 2025"`, to consider 'I' susceptible, e.g., to return `"SI"` vs. `"R"` in `proportion_df()`.
- `"CLSI"`, or any CLSI version such as `"CLSI 2025"`, to consider 'I' resistant, e.g., to return `"S"` vs. `"IR"` in `proportion_df()`.
- `NULL` **or** `FALSE`, to keep S, I, and R separate. This is only possible in functions that return results per category: `count_df()`, `proportion_df()`, `sir_df()`, `ggplot_sir()`, `geom_sir()`, `labels_sir_count()`, and `bug_drug_combinations()`. Functions that return a single proportion or count, such as `susceptibility()`, `count_resistant()`, `antibiogram()`, `wisca()`, and `format()` on the result of `bug_drug_combinations()`, require 'I' to be considered either susceptible or resistant, and return an error.

Use `susceptibility()` and `resistance()` (or `count_susceptible()` and `count_resistant()`) to follow the guideline, or e.g. `proportion_SI()` and `proportion_R()` to be explicit regardless of the guideline.

The `combine_SI` argument of the functions that return results per category, `antibiogram()`, and `wisca()` is deprecated in favour of `guideline`. If set, it is followed literally and takes precedence over `guideline`.

### Combination Therapy

Results are ordered from susceptible to resistant: S (and WT), SDD, I, R (and NWT, NS). With `method = "most_susceptible"` (in `sir_combine()`), the most susceptible result of the antimicro-

bials is returned, and with `method = "most_resistant"`, the most resistant result. The value `"NI"` (non-interpretable) is considered a missing result.

A missing result can only be ignored if it cannot change the conclusion: a susceptible result for `"most_susceptible"`, and a non-susceptible result for `"most_resistant"`. Under EUCAST, the 'I' category is considered susceptible (*susceptible, increased exposure*), and under CLSI, it is not. With `only_all_tested = TRUE`, any missing result leads to `NA`.

This is an example for two antimicrobials, Drug A and Drug B, under EUCAST (the order of the antimicrobials does not matter):

| | | Result of method = "most_susceptible" | | Result of method = "most_resistant" | |
|---|---|---|---|---|---|
| | | only_all_tested = | | only_all_tested = | |
| Drug A | Drug B | FALSE | TRUE | FALSE | TRUE |
| S | S | S | S | S | S |
| S | I | S | S | I | I |
| S | R | S | S | R | R |
| I | R | I | I | R | R |
| R | R | R | R | R | R |
| S | <NA> | S | <NA> | <NA> | <NA> |
| I | <NA> | I | <NA> | <NA> | <NA> |
| R | <NA> | <NA> | <NA> | R | <NA> |
| <NA> | <NA> | <NA> | <NA> | <NA> | <NA> |

Under CLSI, the combination of I with a missing result is <NA> for `"most_susceptible"` and I for `"most_resistant"`.

The `count_*()` and `proportion_*()` functions apply these rules when more than one variable is used. With `only_all_tested = FALSE` (default), the functions for susceptibility, such as `susceptibility()` and `count_susceptible()`, count an isolate as susceptible if any of the antimicrobials is susceptible (`"most_susceptible"`), and the functions for resistance, such as `resistance()` and `count_resistant()`, count an isolate as resistant if any of the antimicrobials is resistant (`"most_resistant"`). An isolate is considered tested if it has such a result, or if it was tested for all antimicrobials. Therefore, for combinations with `only_all_tested = FALSE`:

```
   count_S()    +   count_I()    +   count_R()    >= count_all()
proportion_S() + proportion_I() + proportion_R() >= 1
```

With `only_all_tested = TRUE`, all these functions count the result of the combination therapy (`"most_susceptible"`) of isolates tested for all antimicrobials, so that:

```
   count_S()    +   count_I()    +   count_R()    = count_all()
proportion_S() + proportion_I() + proportion_R() = 1
```

In `antibiogram()` and `wisca()`, the result of the combination therapy (`"most_susceptible"`) is used, the 'I' category is considered susceptible or not following the `guideline` argument ('SDD' is always considered susceptible), and the result is then reduced to S or R.

## References

**M39 Analysis and Presentation of Cumulative Antimicrobial Susceptibility Test Data, 5th Edition**, 2022, *Clinical and Laboratory Standards Institute (CLSI)*. https://clsi.org/standards/products/microbiology/documents/m39/.

## See Also

`count()` to count resistant and susceptible isolates.

## Examples

```
# example_isolates is a data set available in the AMR package.
# run ?example_isolates for more info.
example_isolates


# base R ------------------------------------------------------------
# determines %R
resistance(example_isolates$AMX)
sir_confidence_interval(example_isolates$AMX)
sir_confidence_interval(example_isolates$AMX,
  confidence_level = 0.975
)
sir_confidence_interval(example_isolates$AMX,
  confidence_level = 0.975,
  collapse = ", "
)

# determines %S+I:
susceptibility(example_isolates$AMX)
sir_confidence_interval(example_isolates$AMX,
  ab_result = c("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)

# dplyr -------------------------------------------------------------

if (require("dplyr")) {
  example_isolates %>%
    group_by(ward) %>%
    summarise(
      r = resistance(CIP),
      n = n_sir(CIP)
    ) # n_sir works like n_distinct in dplyr, see ?n_sir
}
if (require("dplyr")) {
```

```
  example_isolates %>%
    group_by(ward) %>%
    summarise(
      cipro_R = resistance(CIP),
      ci_min = sir_confidence_interval(CIP, side = "min"),
      ci_max = sir_confidence_interval(CIP, side = "max"),
    )
}
if (require("dplyr")) {
  # scoped dplyr verbs with antimicrobial selectors
  # (you could also use across() of course)
  example_isolates %>%
    group_by(ward) %>%
    summarise_at(
      c(aminoglycosides(), carbapenems()),
      resistance
    )
}
if (require("dplyr")) {
  example_isolates %>%
    group_by(ward) %>%
    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_sir(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(ward) %>%
  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 sir columns
example_isolates %>%
  select(AMX, CIP) %>%
  proportion_df(translate = FALSE)

# It also supports grouping variables
# (use sir_df to also include the count)
example_isolates %>%
  select(ward, AMX, CIP) %>%
  group_by(ward) %>%
  sir_df(translate = FALSE)
}
```

---

random *Random MIC Values/Disk Zones/SIR Generation*

---

## Description

These functions can be used for generating random MIC values and disk diffusion diameters, for AMR data analysis practice. By providing a microorganism and antimicrobial drug, the generated results will reflect reality as much as possible.

## Usage

```
random_mic(
  size = NULL,
  mo = NULL,
  ab = NULL,
  skew = "right",
  severity = 1,
  guideline = getOption("AMR_guideline", "EUCAST"),
  ...
)
```

```
random_disk(
  size = NULL,
  mo = NULL,
  ab = NULL,
  skew = "left",
  severity = 1,
  guideline = getOption("AMR_guideline", "EUCAST"),
  ...
)

random_sir(size = NULL, prob_SIR = c(0.33, 0.33, 0.33), ...)
```

## Arguments

| | |
|---|---|
| size | Desired size of the returned vector. If used in a data.frame call or dplyr verb, will get the current (group) size if left blank. |
| mo | Any character that can be coerced to a valid microorganism code with as.mo(). Can be the same length as size. |
| ab | Any character that can be coerced to a valid antimicrobial drug code with as.ab(). |
| skew | Direction of skew for MIC or disk values, either "right" or "left". A left-skewed distribution has the majority of the data on the right. |
| severity | Skew severity; higher values will increase the skewedness. Default is 2; use 0 to prevent skewedness. |
| guideline | A guideline name to base the generated values on, either "EUCAST" (default) or "CLSI" for the latest implemented version, or a guideline including a year (e.g., "CLSI 2019"). The default can be changed with the package option AMR_guideline, see as.sir(). |
| ... | Ignored, only in place to allow future extensions. |
| prob_SIR | A vector of length 3: the probabilities for "S" (1st value), "I" (2nd value) and "R" (3rd value). |

## Details

Internally, MIC and disk zone values are sampled based on the human clinical breakpoints of guideline in the clinical_breakpoints data set. To create specific generated values per bug or drug, set the mo and/or ab argument. The MICs are sampled on a log2 scale and disks linearly, using weighted probabilities. The weights are based on the skew and severity arguments:

- skew = "right" places more emphasis on lower MIC or higher disk values.
- skew = "left" places more emphasis on higher MIC or lower disk values.
- severity controls the exponential bias applied.

## Value

class mic for random_mic() (see as.mic()) and class disk for random_disk() (see as.disk())

## Examples

```
random_mic(25)
random_disk(25)
random_sir(25)

# add more skewedness, make more realistic by setting a bug and/or drug:
disks <- random_disk(100, severity = 2, mo = "Escherichia coli", ab = "CIP")
plot(disks)
# `plot()` and `ggplot2::autoplot()` allow for coloured bars if `mo` and `ab` are set
plot(disks, mo = "Escherichia coli", ab = "CIP", guideline = "CLSI")


random_mic(25, "Klebsiella pneumoniae") # range 0.0625-64
random_mic(25, "Klebsiella pneumoniae", "meropenem") # range 0.0625-16
random_mic(25, "Streptococcus pneumoniae", "meropenem") # range 0.0625-4

random_disk(25, "Klebsiella pneumoniae") # range 8-50
random_disk(25, "Klebsiella pneumoniae", "ampicillin") # range 11-17
random_disk(25, "Streptococcus pneumoniae", "ampicillin") # range 12-27
```

---

# resistance_predict *Predict Antimicrobial Resistance*

---

## Description

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

**NOTE:** These functions are deprecated and will be removed in a future version. Use the AMR package combined with the tidymodels framework instead, for which we have written a basic and short introduction on our website.

## 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(),
  ...
)
```

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

## S3 method for class 'resistance_predict'
autoplot(
  object,
  main = paste("Resistance Prediction of", x_name),
  ribbon = TRUE,
  ...
)
```

## Arguments

| | |
|---|---|
| x | A data.frame containing isolates. Can be left blank for automatic determination, see *Examples*. |
| 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 - the default is the first column of with a date class. |
| year_min | Lowest year to use in the prediction model, dafaults to the lowest year in col_date. |
| year_max | Highest year to use in the prediction model - the default is 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. |
| ... | Arguments passed on to functions. |
| main | Title of the plot. |
| ribbon | A logical to indicate whether a ribbon should be shown (default) or error bars. |
| object | Model data to be plotted. |

## Details

Valid options for the statistical model (argument 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, see *Examples*.

### Interpretation of SIR

In 2019, the European Committee on Antimicrobial Susceptibility Testing (EUCAST) has decided to change the definitions of susceptibility testing categories S, I, and R (https://www.eucast.org/bacteria/clinical-breakpoints-and-interpretation/definition-of-s-i-and-r/).

In aggregated analyses, such as proportions, counts, antibiograms, and plots, the 'I' category must be considered either susceptible or resistant. In epidemiological AMR data analysis, this follows the guideline: EUCAST considers 'I' susceptible, while CLSI considers 'I' resistant. The dose-dependent susceptibility of CLSI ('SDD') is always considered susceptible, as it equals what EUCAST defines as 'I'. All functions with a `guideline` argument for this purpose use these categories:

| `guideline` | Susceptible | Resistant |
|---|---|---|
| `"EUCAST"` | S, SDD, I, WT | R, NWT, NS |
| `"CLSI"` | S, SDD, WT | I, R, NWT, NS |

The values 'WT' (wild type) and 'NWT' (non-wild type) result from epidemiological cut-off values (ECOFFs) instead of clinical breakpoints. Since they are no clinical categories, 'WT' is always considered susceptible and 'NWT' always resistant, regardless of the guideline. The value 'NS' (non-susceptible) is always considered resistant, and 'NI' (non-interpretable) is always considered a missing result.

The `guideline` argument can be:

- **Left unset**, to use the package option `AMR_guideline`, or `"EUCAST"` if this option is not set. In the latter case, a message is shown once per session (but not in R Markdown or Quarto documents); set the option to silence it.
- `"EUCAST"`, or any EUCAST version such as `"EUCAST 2025"`, to consider 'I' susceptible, e.g., to return `"SI"` vs. `"R"` in `proportion_df()`.
- `"CLSI"`, or any CLSI version such as `"CLSI 2025"`, to consider 'I' resistant, e.g., to return `"S"` vs. `"IR"` in `proportion_df()`.
- `NULL` **or** `FALSE`, to keep S, I, and R separate. This is only possible in functions that return results per category: `count_df()`, `proportion_df()`, `sir_df()`, `ggplot_sir()`, `geom_sir()`, `labels_sir_count()`, and `bug_drug_combinations()`. Functions that return a single proportion or count, such as `susceptibility()`, `count_resistant()`, `antibiogram()`, `wisca()`, and `format()` on the result of `bug_drug_combinations()`, require 'I' to be considered either susceptible or resistant, and return an error.

Use `susceptibility()` and `resistance()` (or `count_susceptible()` and `count_resistant()`) to follow the guideline, or e.g. `proportion_SI()` and `proportion_R()` to be explicit regardless of the guideline.

The `combine_SI` argument of the functions that return results per category, `antibiogram()`, and `wisca()` is deprecated in favour of `guideline`. If set, it is followed literally and takes precedence over `guideline`.

### See Also

The `proportion()` functions to calculate resistance

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

## Examples

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

if (require("ggplot2")) {
  ggplot_sir_predict(x)
}

# using dplyr:
if (require("dplyr")) {
  x <- example_isolates %>%
    filter_first_isolate() %>%
    filter(mo_genus(mo) == "Staphylococcus") %>%
    resistance_predict("PEN", model = "binomial")
  print(plot(x))

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

# create nice plots with ggplot2 yourself
if (require("dplyr") && 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
    )
  head(data)
  autoplot(data)
}
```

---

`sir_combine` *Combine SIR Results of Multiple Antimicrobials*

---

## Description

Combine the SIR results of two or more antimicrobials into one SIR result per isolate. This can answer two different questions, set with the `method` argument:

- "most_susceptible" (default): *Is at least one of these antimicrobials expected to work?* This is the result of a combination therapy, e.g. piperacillin/tazobactam with gentamicin, which covers the isolate if any of its antimicrobials is active.
- "most_resistant": *Is resistance demonstrated to any of these antimicrobials?* This is the result for a group of antimicrobials, e.g. resistance to third-generation cephalosporins if an isolate is resistant to ceftazidime or ceftriaxone, as used in surveillance reports.

See *Combination Therapy* below for the exact rules. The same rules are used by antibiogram() and wisca() for combination antibiograms, and by the count_*() and proportion_*() functions when more than one variable is used.

## Usage

```
sir_combine(
  ...,
  method = c("most_susceptible", "most_resistant"),
  only_all_tested = FALSE,
  guideline = getOption("AMR_guideline", "EUCAST")
)
```

## Arguments

| | |
|---|---|
| ... | Two or more vectors (or columns) with antimicrobial interpretations, or a data frame with such columns. They will be transformed internally with as.sir() if needed. All vectors must be of the same length. |
| method | Either "most_susceptible" (default) to return the most susceptible result, as for combination therapy, or "most_resistant" to return the most resistant result, as for resistance within a group of antimicrobials. See *Combination Therapy* below. |
| only_all_tested | A logical to indicate that isolates must be tested for all antimicrobials. If TRUE, the result is NA for isolates with a missing result for any of the antimicrobials. If FALSE (default), a missing result only leads to NA if a result for the missing antimicrobial could have changed the result, see *Combination Therapy* below. |
| guideline | A guideline name, "EUCAST" or "CLSI" (or any version thereof, e.g. "CLSI 2025"), to determine whether the 'I' category is considered susceptible (EUCAST) or resistant (CLSI). This only affects isolates with a missing result for any of the antimicrobials when only_all_tested = FALSE, see *Combination Therapy* below. The default is the package option AMR_guideline, or "EUCAST" if it is not set. See section *Interpretation of SIR* below. |

## Value

A sir vector with the same length as the input.

## Combination Therapy

Results are ordered from susceptible to resistant: S (and WT), SDD, I, R (and NWT, NS). With method = "most_susceptible" (in sir_combine()), the most susceptible result of the antimicro-

bials is returned, and with `method = "most_resistant"`, the most resistant result. The value `"NI"` (non-interpretable) is considered a missing result.

A missing result can only be ignored if it cannot change the conclusion: a susceptible result for `"most_susceptible"`, and a non-susceptible result for `"most_resistant"`. Under EUCAST, the 'I' category is considered susceptible (*susceptible, increased exposure*), and under CLSI, it is not. With `only_all_tested = TRUE`, any missing result leads to `NA`.

This is an example for two antimicrobials, Drug A and Drug B, under EUCAST (the order of the antimicrobials does not matter):

| | | Result of method = "most_susceptible" | | Result of method = "most_resistant" | |
|---|---|---|---|---|---|
| | | only_all_tested = | | only_all_tested = | |
| Drug A | Drug B | FALSE | TRUE | FALSE | TRUE |
| S | S | S | S | S | S |
| S | I | S | S | I | I |
| S | R | S | S | R | R |
| I | R | I | I | R | R |
| R | R | R | R | R | R |
| S | <NA> | S | <NA> | <NA> | <NA> |
| I | <NA> | I | <NA> | <NA> | <NA> |
| R | <NA> | <NA> | <NA> | R | <NA> |
| <NA> | <NA> | <NA> | <NA> | <NA> | <NA> |

Under CLSI, the combination of I with a missing result is <NA> for `"most_susceptible"` and I for `"most_resistant"`.

The `count_*()` and `proportion_*()` functions apply these rules when more than one variable is used. With `only_all_tested = FALSE` (default), the functions for susceptibility, such as `susceptibility()` and `count_susceptible()`, count an isolate as susceptible if any of the antimicrobials is susceptible (`"most_susceptible"`), and the functions for resistance, such as `resistance()` and `count_resistant()`, count an isolate as resistant if any of the antimicrobials is resistant (`"most_resistant"`). An isolate is considered tested if it has such a result, or if it was tested for all antimicrobials. Therefore, for combinations with `only_all_tested = FALSE`:

```
   count_S()    +   count_I()    +   count_R()    >= count_all()
proportion_S() + proportion_I() + proportion_R() >= 1
```

With `only_all_tested = TRUE`, all these functions count the result of the combination therapy (`"most_susceptible"`) of isolates tested for all antimicrobials, so that:

```
   count_S()    +   count_I()    +   count_R()    = count_all()
proportion_S() + proportion_I() + proportion_R() = 1
```

In `antibiogram()` and `wisca()`, the result of the combination therapy (`"most_susceptible"`) is used, the 'I' category is considered susceptible or not following the `guideline` argument ('SDD' is always considered susceptible), and the result is then reduced to S or R.

### Interpretation of SIR

In 2019, the European Committee on Antimicrobial Susceptibility Testing (EUCAST) has decided to change the definitions of susceptibility testing categories S, I, and R (https://www.eucast.org/bacteria/clinical-breakpoints-and-interpretation/definition-of-s-i-and-r/).

In aggregated analyses, such as proportions, counts, antibiograms, and plots, the 'I' category must be considered either susceptible or resistant. In epidemiological AMR data analysis, this follows the guideline: EUCAST considers 'I' susceptible, while CLSI considers 'I' resistant. The dose-dependent susceptibility of CLSI ('SDD') is always considered susceptible, as it equals what EUCAST defines as 'I'. All functions with a `guideline` argument for this purpose use these categories:

| guideline | Susceptible | Resistant |
|---|---|---|
| "EUCAST" | S, SDD, I, WT | R, NWT, NS |
| "CLSI" | S, SDD, WT | I, R, NWT, NS |

The values 'WT' (wild type) and 'NWT' (non-wild type) result from epidemiological cut-off values (ECOFFs) instead of clinical breakpoints. Since they are no clinical categories, 'WT' is always considered susceptible and 'NWT' always resistant, regardless of the guideline. The value 'NS' (non-susceptible) is always considered resistant, and 'NI' (non-interpretable) is always considered a missing result.

The `guideline` argument can be:

- **Left unset**, to use the package option `AMR_guideline`, or `"EUCAST"` if this option is not set. In the latter case, a message is shown once per session (but not in R Markdown or Quarto documents); set the option to silence it.
- `"EUCAST"`, or any EUCAST version such as `"EUCAST 2025"`, to consider 'I' susceptible, e.g., to return `"SI"` vs. `"R"` in `proportion_df()`.
- `"CLSI"`, or any CLSI version such as `"CLSI 2025"`, to consider 'I' resistant, e.g., to return `"S"` vs. `"IR"` in `proportion_df()`.
- `NULL` **or** `FALSE`, to keep S, I, and R separate. This is only possible in functions that return results per category: `count_df()`, `proportion_df()`, `sir_df()`, `ggplot_sir()`, `geom_sir()`, `labels_sir_count()`, and `bug_drug_combinations()`. Functions that return a single proportion or count, such as `susceptibility()`, `count_resistant()`, `antibiogram()`, `wisca()`, and `format()` on the result of `bug_drug_combinations()`, require 'I' to be considered either susceptible or resistant, and return an error.

Use `susceptibility()` and `resistance()` (or `count_susceptible()` and `count_resistant()`) to follow the guideline, or e.g. `proportion_SI()` and `proportion_R()` to be explicit regardless of the guideline.

The `combine_SI` argument of the functions that return results per category, `antibiogram()`, and `wisca()` is deprecated in favour of `guideline`. If set, it is followed literally and takes precedence over `guideline`.

### See Also

`count()` and `proportion()` for counting and calculating susceptibility of (combinations of) antimicrobials, and `antibiogram()` for combination antibiograms.

## Examples

```
drug_a <- as.sir(c("S", "R", "R", "I", "I", NA, "R", "SDD"))
drug_b <- as.sir(c("R", "S", "R", "R", NA, NA, NA, "I"))

# combination therapy: is at least one of them expected to work?
sir_combine(drug_a, drug_b)
sir_combine(drug_a, drug_b, guideline = "CLSI")
sir_combine(drug_a, drug_b, only_all_tested = TRUE)

# resistance within a group: is resistance demonstrated to any of them?
sir_combine(drug_a, drug_b, method = "most_resistant")

# the count and proportion functions use the same rules
susceptibility(drug_a, drug_b, minimum = 0)
susceptibility(sir_combine(drug_a, drug_b), minimum = 0)
resistance(drug_a, drug_b, minimum = 0)
resistance(sir_combine(drug_a, drug_b, method = "most_resistant"), minimum = 0)


if (require("dplyr")) {
  # create a new variable for piperacillin/tazobactam + gentamicin
  example_isolates %>%
    filter(!is.na(TZP)) %>%
    mutate(TZP_GEN = sir_combine(TZP, GEN)) %>%
    select(mo, TZP, GEN, TZP_GEN)
}
if (require("dplyr")) {
  # resistance to third-generation cephalosporins
  example_isolates %>%
    mutate(cephalosporins_3rd = sir_combine(CAZ, CRO, method = "most_resistant")) %>%
    count(cephalosporins_3rd)
}
if (require("dplyr")) {
  # combine any number of antimicrobials
  example_isolates %>%
    mutate(any_aminoglycoside = sir_combine(GEN, TOB, AMK, KAN)) %>%
    count(any_aminoglycoside)
}
if (require("dplyr")) {
  # use the new variable in further analyses
  # THOUGH CONSIDER wisca() FOR RELIABLE ESTIMATES
  example_isolates %>%
    mutate(TZP_GEN = sir_combine(TZP, GEN)) %>%
    group_by(ward) %>%
    summarise(
      TZP_mono = susceptibility(TZP),
      GEN_mono = susceptibility(GEN),
      TZP_GEN = susceptibility(TZP_GEN),
      # or directly, without combining first:
      TZP_GEN2 = susceptibility(TZP, GEN)
    )
}
```

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 ('left-skewed'): the left tail is longer; the mass of the distribution is concentrated on the right of a histogram. When positive ('right-skewed'): the right tail is longer; the mass of the distribution is concentrated on the left of a histogram. A normal distribution has a skewness of 0.

## Usage

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

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

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

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

## Arguments

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

## See Also

kurtosis()

## Examples

```
skewness(runif(1000))
```

top_n_microorganisms *Filter Top* n *Microorganisms*

## Description

Filters a data set to include only the top *n* microorganisms based on a specified property, such as taxonomic family or genus. For example, it can filter a data set to the top 3 species, to any species in the top 5 genera, or to the top 3 species in each of the top 5 genera.

## Usage

```
top_n_microorganisms(
  x,
  n,
  property = "species",
  n_for_each = NULL,
  property_for_each = "species",
  col_mo = NULL,
  ...
)
```

## Arguments

| | |
|---|---|
| x | A data frame containing microbial data. |
| n | A positive whole number specifying the maximum number of unique values of property to include in the output. |
| property | A character string indicating the microorganism property to use for filtering. Must be one of the column names of the microorganisms data set: "mo", "fullname", "status", "domain", "kingdom", "phylum", "class", "order", "family", "genus", "species", "subspecies", "rank", "ref", "oxygen_tolerance", "morphology", "source", "lpsn", "lpsn_parent", "lpsn_renamed_to", "mycobank", "mycobank_parent", "mycobank_renamed_to", "gbif", "gbif_parent", "gbif_renamed_to", "prevalence", or "snomed". If NULL, the raw values from col_mo will be used without transformation. When using "species" (default) or "subspecies", the genus is prepended to ensure each name is unambiguous. |
| n_for_each | An optional positive whole number specifying the maximum number of distinct microorganism groups at the level of property_for_each to retain within each of the top *n* groups. Only used when property_for_each is also set. |
| property_for_each | The microorganism property to use for sub-grouping within each top *n* group. Must be one of the column names of the microorganisms data set and at a strictly lower taxonomic rank than property (allowed order: domain > kingdom > phylum > class > order > family > genus > species > subspecies). Defaults to "species". Only relevant when n_for_each is set. |

| | |
|---|---|
| col_mo | A character string indicating the column in x that contains microorganism names or codes. Defaults to the first column of class mo. Values will be coerced using as.mo(). |
| ... | Additional arguments passed on to mo_property() when property is not NULL. |

## Details

This function is useful for preprocessing data before creating antibiograms or other analyses that require focused subsets of microbial data.

## See Also

mo_property(), as.mo(), antibiogram()

## Examples

```
# filter to the top 3 species:
top_n_microorganisms(example_isolates, n = 3)

# filter to any species in the top 5 genera:
top_n_microorganisms(example_isolates, n = 5, property = "genus")

# filter to the top 3 species in each of the top 5 genera:
top_n_microorganisms(example_isolates,
  n = 5, property = "genus", n_for_each = 3
)

# filter to the top 2 genera in each of the top 3 families:
top_n_microorganisms(example_isolates,
  n = 3, property = "family", n_for_each = 2, property_for_each = "genus"
)
```

---

# translate *Translate Strings from the AMR Package*

---

## Description

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

## Usage

```
get_AMR_locale()

set_AMR_locale(language)

reset_AMR_locale()

translate_AMR(x, language = get_AMR_locale())
```

## Arguments

| | |
|---|---|
| language | Language to choose. Use one of these supported language names or ISO 639-1 codes: English (en), Arabic (ar), Bengali (bn), Chinese (zh), Czech (cs), Danish (da), Dutch (nl), Finnish (fi), French (fr), German (de), Greek (el), Hindi (hi), Indonesian (id), Italian (it), Japanese (ja), Korean (ko), Norwegian (no), Polish (pl), Portuguese (pt), Romanian (ro), Russian (ru), Spanish (es), Swahili (sw), Swedish (sv), Turkish (tr), Ukrainian (uk), Urdu (ur), or Vietnamese (vi). |
| x | Text to translate. |

## Details

The currently 28 supported languages are English (en), Arabic (ar), Bengali (bn), Chinese (zh), Czech (cs), Danish (da), Dutch (nl), Finnish (fi), French (fr), German (de), Greek (el), Hindi (hi), Indonesian (id), Italian (it), Japanese (ja), Korean (ko), Norwegian (no), Polish (pl), Portuguese (pt), Romanian (ro), Russian (ru), Spanish (es), Swahili (sw), Swedish (sv), Turkish (tr), Ukrainian (uk), Urdu (ur), and Vietnamese (vi). All these languages have translations available for all antimicrobial drugs and colloquial microorganism names.

To permanently silence the once-per-session language note on a non-English operating system, you can set the package option `AMR_locale` in your `.Rprofile` file like this:

```
# Open .Rprofile file
utils::file.edit("~/.Rprofile")

# Then add e.g. Italian support to that file using:
options(AMR_locale = "Italian")
```

And then save the file.

Please read about adding or updating a language in our Wiki.

### Changing the Default Language:

The system language will be used at default, if that language is supported. The language to be used is determined in this order:

1. The package option `AMR_locale`, set by using e.g. `set_AMR_locale("German")` or by running e.g. `options(AMR_locale = "German")`.
   Note that setting an R option only works in the same session. Save the command `options(AMR_locale = "(your language)")` to your `.Rprofile` file to apply it for every session. Run `utils::file.edit("~/.Rprofile")` to edit your `.Rprofile` file.
2. The system variable `LANG`, as returned by `Sys.getenv("LANG")`, e.g. set by adding `LANG="de_DE.utf8"` to your `.Renviron` file in your home directory.
3. The system variable `LANGUAGE`, but only if `LANG` is not set. Note that `LANG` is set on most Linux and macOS systems, in which case `LANGUAGE` has no effect.
4. The system locale, as returned by `Sys.getlocale("LC_COLLATE")`, if neither `LANG` nor `LANGUAGE` is set.

Thus, if the package option `AMR_locale` is set, the system variables `LANG` and `LANGUAGE` will be ignored.

## Examples

```
# Current settings (based on system language)
ab_name("Ciprofloxacin")
mo_name("Coagulase-negative Staphylococcus (CoNS)")

# setting another language
set_AMR_locale("Dutch")
ab_name("Ciprofloxacin")
mo_name("Coagulase-negative Staphylococcus (CoNS)")

# setting yet another language
set_AMR_locale("German")
ab_name("Ciprofloxacin")
mo_name("Coagulase-negative Staphylococcus (CoNS)")

# set_AMR_locale() understands endonyms, English exonyms, and ISO 639-1:
set_AMR_locale("Deutsch")
set_AMR_locale("German")
set_AMR_locale("de")
ab_name("amox/clav")

# reset to system default
reset_AMR_locale()
ab_name("amox/clav")
```

---

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

---

## Description

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

## WHOCC

This package contains **all ~550 antibiotic, antimycotic and antiviral drugs** and their Anatomical Therapeutic Chemical (ATC) codes, ATC groups and Defined Daily Dose (DDD) from the World Health Organization Collaborating Centre for Drug Statistics Methodology (WHOCC, https://atcddd.fhi.no) and the Pharmaceuticals Community Register of the European Commission (https://ec.europa.eu/health/documents/community-register/html/index_en.htm).

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

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

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

## Examples

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

ab_tradenames("flucloxacillin")
```

---

WHONET *Data Set with 500 Isolates - WHONET Example*

---

## Description

This example data set has the exact same structure as an export file from WHONET. Such files can be used with this package, as this example data set shows. The antimicrobial results are from our example_isolates data set. All patient names were created using online surname generators and are only in place for practice purposes.

## Usage

```
WHONET
```

## Format

A tibble 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`
  Fictitious last name of patient
- `First name`
  Fictitious initial of patient
- `Sex`
  Fictitious gender of patient
- `Age`
  Fictitious 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 antimicrobials. You can lookup the abbreviations in the antimicrobials data set, or use e.g. `ab_name("AMP")` to get the official name immediately. Before analysis, you should transform this to a valid antimicrobial class, using `as.sir()`.

### Download Our Reference Data

All reference data sets in the AMR package - including information on microorganisms, antimicrobials, and clinical breakpoints - are freely available for download in multiple formats: R, MS Excel, Apache Feather, Apache Parquet, SPSS, and Stata.

For maximum compatibility, we also provide machine-readable, tab-separated plain text files suitable for use in any software, including laboratory information systems.

Visit our website for direct download links, or explore the actual files in our GitHub repository.

**Examples**

```
WHONET
```

# Index


∗ **datasets**
antimicrobials, 31
clinical_breakpoints, 90
dosage, 113
esbl_isolates, 114
example_isolates, 115
example_isolates_unclean, 116
intrinsic_resistant, 144
microorganisms, 160
microorganisms.codes, 164
microorganisms.groups, 166
WHONET, 217
'ab' vector, *39*
%like% *(*like*)*, 151
%like_case% *(*like*)*, 151
%unlike% *(*like*)*, 151
%unlike_case% *(*like*)*, 151
3MRGN *(*mdro*)*, 153
4MRGN *(*mdro*)*, 153

ab, *50*, *53*
ab *(*as.ab*)*, 49
ab_∗, *5*, *31*, *50*
ab_atc *(*ab_property*)*, 6
ab_atc(), *8*, *80*
ab_atc_group1 *(*ab_property*)*, 6
ab_atc_group2 *(*ab_property*)*, 6
ab_cid *(*ab_property*)*, 6
ab_cid(), *8*
ab_ddd *(*ab_property*)*, 6
ab_ddd(), *8*
ab_ddd_units *(*ab_property*)*, 6
ab_from_text, 4
ab_from_text(), *51*
ab_group *(*ab_property*)*, 6
ab_group(), *5*
ab_info *(*ab_property*)*, 6
ab_info(), *8*
ab_loinc *(*ab_property*)*, 6
ab_loinc(), *33*
ab_name *(*ab_property*)*, 6
ab_name(), *5*, *115*, *135*, *144*, *214*
ab_name(AMP), *218*
ab_property, 6
ab_property(), *4*, *33*, *94*, *134*, *196*
ab_reset_session *(*as.ab*)*, 49
ab_reset_session(), *50*
ab_synonyms *(*ab_property*)*, 6
ab_synonyms(), *8*
ab_tradenames *(*ab_property*)*, 6
ab_tradenames(), *8*
ab_url *(*ab_property*)*, 6
ab_url(), *8*
add_custom_antimicrobial_synonyms *(*add_custom_antimicrobials*)*, 10
add_custom_antimicrobial_synonyms(), *10*, *11*
add_custom_antimicrobials, 10
add_custom_antimicrobials(), *10*, *11*, *14*, *50*
add_custom_microorganisms, 13
add_custom_microorganisms(), *12–14*, *62*
administrable_iv *(*antimicrobial_selectors*)*, 35
administrable_iv(), *39*
administrable_per_os *(*antimicrobial_selectors*)*, 35
administrable_per_os(), *39*
age, 15
age(), *17*
age_groups, 16
age_groups(), *16*, *217*
all microorganism functions, *60*
all(), *108*
all_antimicrobials *(*key_antimicrobials*)*, 147
all_antimicrobials(), *120*, *149*
aminoglycosides *(*antimicrobial_selectors*)*, 35

aminoglycosides(), *21*, *39*, *40*, *108*
aminopenicillins *(*antimicrobial_selectors*)*, 35
aminopenicillins(), *40*, *108*
AMR interpretation guideline, *189*
AMR_antibiogram_formatting_type, *24*
AMR_breakpoint_type, *71*, *92*, *190*
AMR_capped_mic_handling, *70*
amr_class *(*antimicrobial_selectors*)*, 35
amr_class(), *39*
AMR_cleaning_regex, *60*
amr_course, 18
AMR_custom_ab, *10*, *11*
AMR_custom_mo, *13*
amr_distance_from_row *(*mean_amr_distance*)*, 158
amr_distance_from_row(), *159*
AMR_guideline, *22*, *28*, *69*, *74*, *75*, *88*, *89*, *94*, *95*, *134*, *136*, *141*, *157*, *190*, *195*, *197*, *202*, *206*, *208*, *210*
AMR_ignore_pattern, *60*
AMR_include_PKPD, *71*, *189*
AMR_include_screening, *71*
AMR_interpretive_rules, *140*, *142*
AMR_keep_synonyms, *60*, *175*
AMR_locale, *7*, *85*, *88*, *94*, *134*, *189*, *196*, *215*
AMR_mo_source, *182*
AMR_only_fungi, *60*
amr_selector *(*antimicrobial_selectors*)*, 35
amr_selector(), *39*
AMR_substitute_missing_r_breakpoint, *71*
anti_join_microorganisms *(*join*)*, 146
antibiogram, 19
antibiogram(), *23–25*, *27*, *28*, *75*, *89*, *96*, *97*, *136*, *158*, *196–198*, *206*, *208–210*, *214*
antibiograms, *214*
antibiotics *(*antimicrobials*)*, 31
antifungals *(*antimicrobial_selectors*)*, 35
antifungals(), *40*, *108*
antimicrobial selector, *21*, *144*
antimicrobial selectors, *21*, *69*, *159*
antimicrobial_selectors, 35
antimicrobials, *4*, *6–8*, *10*, *21*, *31*, 31, *32*, *33*, *38*, *39*, *49–51*, *88*, *94*, *115*, *134*, *138*, *139*, *144*, *196*, *218*
antimicrobials$ab, *144*
antimicrobials_equal *(*key_antimicrobials*)*, 147
antimicrobials_equal(), *149*
antimycobacterials *(*antimicrobial_selectors*)*, 35
antimycobacterials(), *40*, *108*
antivirals, *33*, *52*, *53*, *83–86*
antivirals *(*antimicrobials*)*, 31
any(), *108*
as.ab, 49
as.ab(), *4–7*, *10*, *11*, *31*, *50*, *69*, *71*, *80*, *91*, *138*, *142*, *176*, *189*, *202*
as.av, 52
as.av(), *52*, *83–85*
as.character(), *152*
as.disk, 54
as.disk(), *73*, *76*, *189*, *202*
as.double(), *57*, *74*
as.integer(), *57*
as.mic, 55
as.mic(), *72*, *76*, *114*, *159*, *189*, *202*
as.mo, 59
as.mo(), *22*, *39*, *61*, *62*, *64*, *69*, *71*, *72*, *76*, *87*, *91*, *115*, *116*, *118*, *140*, *146*, *148*, *154*, *160*, *164–167*, *169*, *175*, *176*, *178*, *181–183*, *189*, *202*, *214*, *217*
as.numeric(), *57*
as.POSIXlt(), *15*, *16*
as.sir, 67
as.sir(), *21*, *54*, *56*, *58*, *67*, *70*, *72–74*, *90*, *92*, *94*, *95*, *115*, *116*, *119*, *134*, *135*, *142*, *159*, *189*, *190*, *195*, *196*, *202*, *208*, *218*
ATC *(*ab_property*)*, 6
atc_online_ddd *(*atc_online_property*)*, 80
atc_online_ddd_units *(*atc_online_property*)*, 80
atc_online_groups *(*atc_online_property*)*, 80
atc_online_property, 80
attribute, *24*, *25*, *182*
autoplot.antibiogram *(*antibiogram*)*, 19
autoplot.disk *(*plot*)*, 186
autoplot.mic *(*plot*)*, 186
autoplot.resistance_predict *(*resistance_predict*)*, 203

autoplot.sir *(plot)*, 186
av *(as.av)*, 52
av_*, *52*, *84*
av_atc *(av_property)*, 84
av_atc(), *86*
av_cid *(av_property)*, 84
av_cid(), *86*
av_ddd *(av_property)*, 84
av_ddd(), *86*
av_ddd_units *(av_property)*, 84
av_from_text, 83
av_from_text(), *53*
av_group *(av_property)*, 84
av_group(), *84*
av_info *(av_property)*, 84
av_info(), *86*
av_loinc *(av_property)*, 84
av_loinc(), *33*
av_name *(av_property)*, 84
av_name(), *84*
av_property, 84
av_property(), *33*, *83*
av_synonyms *(av_property)*, 84
av_synonyms(), *86*
av_tradenames *(av_property)*, 84
av_tradenames(), *86*
av_url *(av_property)*, 84
av_url(), *85*
availability, 82
available microbial taxonomy, *61*

barplot(), *25*
betalactams *(antimicrobial_selectors)*, 35
betalactams(), *40*, *109*
betalactams_with_inhibitor *(antimicrobial_selectors)*, 35
betalactams_with_inhibitor(), *41*, *109*
binom.test(), *195*, *196*
biplot(), *129*
BRMO *(mdro)*, 153
brmo *(mdro)*, 153
browseURL(), *176*
bug_drug_combinations, 87
bug_drug_combinations(), *28*, *75*, *88*, *89*, *96*, *136*, *158*, *197*, *206*, *210*

c(), *107*, *108*
c.custom_mdro_guideline *(custom_mdro_guideline)*, 106
carbapenems *(antimicrobial_selectors)*, 35
carbapenems(), *21*, *41*, *109*
case_when(), *22*, *99*, *107*
cephalosporins *(antimicrobial_selectors)*, 35
cephalosporins(), *41*, *109*
cephalosporins_1st *(antimicrobial_selectors)*, 35
cephalosporins_1st(), *42*, *110*
cephalosporins_2nd *(antimicrobial_selectors)*, 35
cephalosporins_2nd(), *42*, *110*
cephalosporins_3rd *(antimicrobial_selectors)*, 35
cephalosporins_3rd(), *42*, *110*
cephalosporins_4th *(antimicrobial_selectors)*, 35
cephalosporins_4th(), *42*, *110*
cephalosporins_5th *(antimicrobial_selectors)*, 35
cephalosporins_5th(), *42*, *110*
character, *4*, *5*, *7*, *8*, *10*, *15*, *39*, *49*, *50*, *52*, *53*, *56*, *60*, *62*, *63*, *69–71*, *80*, *83–86*, *88*, *107*, *118*, *140*, *141*, *145*, *146*, *148*, *149*, *152*, *175*, *177*, *189*, *196*, *202*
chisq.test(), *122–124*
clear_custom_antimicrobials *(add_custom_antimicrobials)*, 10
clear_custom_antimicrobials(), *11*, *50*
clear_custom_microorganisms *(add_custom_microorganisms)*, 13
clear_custom_microorganisms(), *14*
clinical_breakpoints, *67*, *69–72*, 90, *202*
clsi_rules *(interpretive_rules)*, 139
count, 93
count(), *199*, *210*
count_*(), *27*, *96*, *196*, *198*, *208*, *209*
count_all *(count)*, 93
count_all(), *95*, *196*
count_df *(count)*, 93
count_df(), *28*, *75*, *89*, *94*, *95*, *134–136*, *157*, *195*, *197*, *206*, *210*
count_I *(count)*, 93
count_IR *(count)*, 93

count_R (count), 93
count_resistant (count), 93
count_resistant(), *27, 28*, *75*, *89*, *93*, *96, 97*, *136*, *158*, *197, 198*, *206*, *209, 210*
count_S (count), 93
count_SI (count), 93
count_susceptible (count), 93
count_susceptible(), *27, 28*, *75*, *89*, *93*, *96, 97*, *136*, *158*, *196–198*, *206*, *209, 210*
custom_interpretive_rules, 98
custom_interpretive_rules(), *140–142*
custom_mdro_guideline, 106
custom_mdro_guideline(), *156*, *158*

data.frame, *7, 8*, *10*, *13*, *21*, *51*, *53*, *57*, *60–62*, *66*, *69–71*, *73, 74*, *82*, *88*, *94*, *118, 119*, *134*, *138, 139*, *143*, *146–149*, *151*, *154–156*, *159*, *182*, *184, 185*, *196*, *202*, *204, 205*, *212*
Date, *218*
deprecated, *203*
detecting multidrug-resistant organisms here, *74*
disk, *54*, *69*, *73*, *159*
disk (as.disk), 54
dosage, 113, *139*
double, *8*, *16*, *86*, *196*
droplevels(), *56, 57*
droplevels.mic (as.mic), 55

esbl_isolates, 114
EUCAST (interpretive_rules), 139
eucast_dosage (interpretive_rules), 139
eucast_dosage(), *113*, *139*
eucast_exceptional_phenotypes (mdro), 153
eucast_rules (interpretive_rules), 139
eucast_rules(), *39*
everything(), *39*
example_isolates, 115, *217*
example_isolates_unclean, 116
exp(), *57*
expression, *39*

facet_sir (plot), 186
facet_sir(), *135*, *190*
factor, *16, 17*, *56–58*, *67*, *74*, *107*, *157*, *176*
filter(), *39*, *127*
filter_first_isolate (first_isolate), 117
filter_first_isolate(), *119*
first_isolate, 117
first_isolate(), *24*, *119, 120*, *127*, *147, 149, 150*, *196*
fisher.test(), *123*
fivenum(), *57*
fluoroquinolones (antimicrobial_selectors), 35
fluoroquinolones(), *42*, *110*
format(), *28*, *75*, *87–89*, *96*, *136*, *158*, *197*, *206*, *210*
format.bug_drug_combinations (bug_drug_combinations), 87
formula, *99*, *107*
full_join_microorganisms (join), 146
future.apply, *22*, *72*
future::plan(), *22*, *72*

g.test, 122
g.test(), *122*
geom_sir (ggplot_sir), 132
geom_sir(), *28*, *75*, *89*, *94, 95*, *134–136*, *157*, *195*, *197*, *206*, *210*
get_AMR_locale (translate), 214
get_AMR_locale(), *7*, *22*, *49*, *61*, *85*, *88*, *94*, *134*, *175*, *189*, *196*
get_episode, 125
get_episode(), *125*, *127*
get_mo_source (mo_source), 181
get_mo_source(), *60*, *182*
ggplot, *191*
ggplot2, *132*
ggplot2 theme, *135*, *190*
ggplot2::autoplot(), *25*
ggplot2::facet_wrap(), *135*, *190*
ggplot2::geom_text(), *135*, *190*
ggplot2::scale_y_continuous(), *135*, *190*
ggplot_pca, 129
ggplot_pca(), *131*, *185*
ggplot_sir, 132
ggplot_sir(), *28*, *75*, *89*, *94, 95*, *134–136*, *157*, *190*, *195*, *197*, *206*, *210*
ggplot_sir_predict (resistance_predict), 203
glm(), *206*
glycopeptides (antimicrobial_selectors), 35

glycopeptides(), *43*, *110*
grepl(), *151*, *152*
group generic functions, *57*
guess_ab_col, 138
guess_ab_col(), *21*, *142*, *155*

human pathogenicity, *60*, *61*

ifelse(), *22*
inner_join *(*join*)*, 146
inner_join_microorganisms *(*join*)*, 146
integer, *8*, *16*, *54*, *86*, *95*, *127*, *177*
interaction(), *146*
interactive sessions, *69*
interpretive_rules, 139
interpretive_rules(), *74*, *99*
intrinsic_resistant, *23*, *26*, *34*, *71*, *93*, 144, *164*, *177*
ionophores *(*antimicrobial_selectors*)*, 35
ionophores(), *43*, *110*
is.ab *(*as.ab*)*, 49
is.av *(*as.av*)*, 52
is.disk *(*as.disk*)*, 54
is.mic *(*as.mic*)*, 55
is.mic(), *57*
is.mo *(*as.mo*)*, 59
is.sir *(*as.sir*)*, 67
is.sir(), *74*
is_new_episode *(*get_episode*)*, 125
is_new_episode(), *117*, *119*, *125*, *127*
is_sir_eligible *(*as.sir*)*, 67
is_sir_eligible(), *74*
isoxazolylpenicillins *(*antimicrobial_selectors*)*, 35
isoxazolylpenicillins(), *43*, *111*
italicise_taxonomy, 145
italicise_taxonomy(), *23*
italicize_taxonomy *(*italicise_taxonomy*)*, 145

join, 146

key_antimicrobials, 147
key_antimicrobials(), *118*, *119*, *121*, *149*
knit_print.antibiogram *(*antibiogram*)*, 19
knitr, *23*
knitr::kable(), *25*, *88*
kurtosis, 150
kurtosis(), *212*

labels_sir_count *(*ggplot_sir*)*, 132
labels_sir_count(), *28*, *75*, *89*, *94*, *96*, *134–136*, *158*, *190*, *195*, *197*, *206*, *210*
left_join_microorganisms *(*join*)*, 146
like, 151
like(), *152*
lincosamides *(*antimicrobial_selectors*)*, 35
lincosamides(), *43*, *111*
lipoglycopeptides *(*antimicrobial_selectors*)*, 35
lipoglycopeptides(), *43*, *111*
list, *4*, *5*, *8*, *57*, *74*, *82–84*, *86*, *106*, *112*, *124*, *177*
lm(), *206*
log2(), *57*, *159*
logical, *4*, *7*, *15–17*, *22*, *23*, *38*, *49*, *52*, *54*, *56*, *57*, *60*, *61*, *69–72*, *74*, *83*, *85*, *88*, *91*, *94*, *107*, *118*, *119*, *121*, *127*, *131*, *134*, *138*, *140*, *141*, *148*, *151*, *152*, *154*, *155*, *159*, *175*, *177*, *189*, *195*, *196*, *205*, *208*, *212*

macrolides *(*antimicrobial_selectors*)*, 35
macrolides(), *43*, *111*
mad(), *57*
matching score, *61*
matching score algorithm, *62*
matrix, *122*, *151*, *212*
MDR *(*mdro*)*, 153
mdr_cmi2012 *(*mdro*)*, 153
mdr_cmi2012(), *157*
mdr_tb *(*mdro*)*, 153
mdr_tb(), *157*
mdro, 153
mdro(), *74*, *106*, *107*, *155–157*
mean_amr_distance, 158
mean_amr_distance(), *159*
median(), *57*
merge(), *146*
mic, *55*, *58*, *69*, *72*, *114*, *159*, *190*
mic *(*as.mic*)*, 55
mic_p50 *(*as.mic*)*, 55
mic_p50(), *57*
mic_p90 *(*as.mic*)*, 55
mic_p90(), *57*
microorganisms, *13*, *22*, *34*, *64–66*, *100*, *115*, *116*, *142*, *144–146*, 160, *165–168*,

176–178, 213
microorganisms.codes, 164, 164
microorganisms.groups, 92, 163, 164, 166
microorganisms$fullname, 167, 182
microorganisms$mo, 144, 182
mo, 22, 39, 59–61, 63, 72, 87, 118, 140, 146, 148, 154, 214
mo (as.mo), 59
MO matching score, 60
MO matching score page, 178
mo_*, 61, 62, 64, 66, 160, 164, 167, 181, 183
mo_authors (mo_property), 169
mo_class (mo_property), 169
mo_cleaning_regex (as.mo), 59
mo_cleaning_regex(), 60, 61
mo_current (mo_property), 169
mo_current(), 177
mo_domain (mo_property), 169
mo_domain(), 176
mo_failures (as.mo), 59
mo_failures(), 62
mo_family (mo_property), 169
mo_fullname (mo_property), 169
mo_gbif (mo_property), 169
mo_genus (mo_property), 169
mo_genus(), 66, 181, 182
mo_gramstain (mo_property), 169
mo_gramstain(), 66, 177, 181, 182, 214
mo_group_members (mo_property), 169
mo_info (mo_property), 169
mo_info(), 177
mo_is_anaerobic (mo_property), 169
mo_is_anaerobic(), 177
mo_is_gram_negative (mo_property), 169
mo_is_gram_negative(), 177
mo_is_gram_positive (mo_property), 169
mo_is_gram_positive(), 177
mo_is_intrinsic_resistant (mo_property), 169
mo_is_intrinsic_resistant(), 144, 177
mo_is_yeast (mo_property), 169
mo_is_yeast(), 177
mo_kingdom (mo_property), 169
mo_kingdom(), 176
mo_lpsn (mo_property), 169
mo_matching_score, 167
mo_matching_score(), 64, 162, 167
mo_morphology (mo_property), 169
mo_morphology(), 175, 177
mo_mycobank (mo_property), 169
mo_name (mo_property), 169
mo_name(), 144, 166, 214
mo_order (mo_property), 169
mo_oxygen_tolerance (mo_property), 169
mo_oxygen_tolerance(), 177
mo_pathogenicity (mo_property), 169
mo_pathogenicity(), 176, 177
mo_phylum (mo_property), 169
mo_property, 169
mo_property(), 162, 164, 214
mo_rank (mo_property), 169
mo_ref (mo_property), 169
mo_ref(), 176
mo_renamed (as.mo), 59
mo_renamed(), 62
mo_reset_session (as.mo), 59
mo_reset_session(), 62
mo_shortname (mo_property), 169
mo_shortname(), 88, 176
mo_snomed (mo_property), 169
mo_snomed(), 162, 177
mo_source, 181
mo_species (mo_property), 169
mo_status (mo_property), 169
mo_subspecies (mo_property), 169
mo_synonyms (mo_property), 169
mo_synonyms(), 177
mo_taxonomy (mo_property), 169
mo_taxonomy(), 177
mo_type (mo_property), 169
mo_type(), 214, 218
mo_uncertainties (as.mo), 59
mo_uncertainties(), 61, 62
mo_url (mo_property), 169
mo_url(), 177
mo_year (mo_property), 169
mo_year(), 177
monobactams (antimicrobial_selectors), 35
monobactams(), 43, 111
mrgn (mdro), 153
mrgn(), 157
mutate(), 127

n_sir (count), 93
n_sir(), 95
NA_ab_ (as.ab), 49

NA_character_, *50*, *57*, *63*, *74*
NA_disk_ *(*as.disk*)*, 54
NA_mic_ *(*as.mic*)*, 55
NA_mo_ *(*as.mo*)*, 59
NA_sir_ *(*as.sir*)*, 67
name, *177*
nitrofurans *(*antimicrobial_selectors*)*, 35
nitrofurans(), *43*, *111*
not_intrinsic_resistant *(*antimicrobial_selectors*)*, 35
not_intrinsic_resistant(), *39*, *144*
numeric, *5*, *17*, *56–58*, *84*, *134*, *184*, *185*, *190*

ordered factor, *177*
oxazolidinones *(*antimicrobial_selectors*)*, 35
oxazolidinones(), *43*, *111*

pca, 184, *185*
pca(), *130*, *185*
PDR *(*mdro*)*, 153
penicillins *(*antimicrobial_selectors*)*, 35
penicillins(), *43*, *111*
peptides *(*antimicrobial_selectors*)*, 35
peptides(), *44*, *111*
phenicols *(*antimicrobial_selectors*)*, 35
phenicols(), *44*, *111*
phosphonics *(*antimicrobial_selectors*)*, 35
phosphonics(), *44*, *111*
plot, 186
plot(), *25*
plot.antibiogram *(*antibiogram*)*, 19
plot.resistance_predict *(*resistance_predict*)*, 203
polymyxins *(*antimicrobial_selectors*)*, 35
polymyxins(), *44*, *111*
portion *(*proportion*)*, 193
prcomp, *185*
prcomp(), *130*, *185*
princomp, *130*
princomp(), *130*
proportion, 193
proportion(), *206*, *210*
proportion_*, *97*
proportion_*(), *27*, *96*, *198*, *208*, *209*
proportion_df *(*proportion*)*, 193
proportion_df(), *28*, *75*, *89*, *94*, *95*, *134*, *136*, *157*, *195–197*, *206*, *210*
proportion_I *(*proportion*)*, 193
proportion_I(), *196*
proportion_IR *(*proportion*)*, 193
proportion_R *(*proportion*)*, 193
proportion_R(), *28*, *75*, *89*, *96*, *136*, *158*, *196*, *197*, *206*, *210*
proportion_S *(*proportion*)*, 193
proportion_SI *(*proportion*)*, 193
proportion_SI(), *28*, *75*, *89*, *96*, *136*, *158*, *196*, *197*, *206*, *210*

quantile(), *57*
quinolones *(*antimicrobial_selectors*)*, 35
quinolones(), *44*, *111*

random, 201
random_disk *(*random*)*, 201
random_disk(), *202*
random_mic *(*random*)*, 201
random_mic(), *189*, *202*
random_sir *(*random*)*, 201
readRDS(), *100*, *108*, *182*
regular expression, *60*, *61*, *69*, *72*
rescale_mic *(*as.mic*)*, 55
rescale_mic(), *56*, *57*, *189*
reset_AMR_locale *(*translate*)*, 214
resistance *(*proportion*)*, 193
resistance(), *27*, *28*, *75*, *82*, *89*, *94*, *96*, *97*, *136*, *158*, *194*, *196–198*, *206*, *209*, *210*
resistance_predict, 203, *205*
retrieve_wisca_parameters *(*antibiogram*)*, 19
retrieve_wisca_parameters(), *23*
rifamycins *(*antimicrobial_selectors*)*, 35
rifamycins(), *44*, *112*
right_join_microorganisms *(*join*)*, 146

saveRDS(), *10*, *13*, *100*, *108*
scale, *184*
scale_*_mic(), *57*
scale_color_mic *(*plot*)*, 186
scale_color_sir *(*plot*)*, 186
scale_colour_mic *(*plot*)*, 186
scale_colour_mic(), *190*
scale_colour_sir *(*plot*)*, 186
scale_colour_sir(), *190*

scale_fill_mic *(plot)*, 186
scale_fill_mic(), *190*
scale_fill_sir *(plot)*, 186
scale_fill_sir(), *190*
scale_sir_colors *(plot)*, 186
scale_sir_colours *(plot)*, 186
scale_sir_colours(), *135*, *190*
scale_x_mic *(plot)*, 186
scale_x_mic(), *190*
scale_x_sir *(plot)*, 186
scale_x_sir(), *190*
scale_y_mic *(plot)*, 186
scale_y_mic(), *190*
scale_y_percent *(plot)*, 186
scale_y_percent(), *135*, *190*
sd(), *57*
select(), *39*
semi_join_microorganisms *(join)*, 146
set_ab_names *(ab_property)*, 6
set_ab_names(), *7*, *8*, *115*
set_AMR_locale *(translate)*, 214
set_AMR_locale(), *71*
set_mo_source *(mo_source)*, 181
set_mo_source(), *60*, *164*, *182*, *183*
sir, *38*, *67*, *94*, *95*, *115*, *134*, *135*, *138*, *141*, *148*, *155*, *159*, *190*, *196*, *208*
sir *(as.sir)*, 67
sir_combine, 207
sir_combine(), *25*, *26*, *94*, *96*, *195*, *197*, *208*
sir_confidence_interval *(proportion)*, 193
sir_confidence_interval(), *196*
sir_df *(proportion)*, 193
sir_df(), *28*, *75*, *89*, *94*, *95*, *134–136*, *157*, *195–197*, *206*, *210*
sir_interpretation_history *(as.sir)*, 67
sir_interpretation_history(), *70*, *73*
sir_predict *(resistance_predict)*, 203
skewness, 212
skewness(), *151*
spiropyrimidinetriones *(antimicrobial_selectors)*, 35
spiropyrimidinetriones(), *44*, *112*
streptogramins *(antimicrobial_selectors)*, 35
streptogramins(), *44*, *112*
sulfonamides *(antimicrobial_selectors)*, 35
sulfonamides(), *44*, *112*
summarise(), *39*, *127*
summary(), *205*
susceptibility *(proportion)*, 193
susceptibility(), *27*, *28*, *75*, *82*, *89*, *94*, *96*, *136*, *158*, *194*, *196–198*, *206*, *209*, *210*
Sys.getlocale(LC_COLLATE), *215*

taxonomic domain, *60*, *61*
tetracyclines *(antimicrobial_selectors)*, 35
tetracyclines(), *45*, *112*
the clinical breakpoints table, *166*
theme_sir *(plot)*, 186
theme_sir(), *135*, *190*
tibble, *32*, *33*, *90*, *113–116*, *144*, *161*, *165*, *166*, *217*
tibbles, *25*
tidymodels integration, *115*
tidyselect language, *69*, *159*
Tidyverse selection helpers, *39*
top_n_microorganisms, 213
top_n_microorganisms(), *24*
translate, *189*, 214
translate_AMR *(translate)*, 214
trimethoprims *(antimicrobial_selectors)*, 35
trimethoprims(), *45*, *112*

ureidopenicillins *(antimicrobial_selectors)*, 35
ureidopenicillins(), *45*, *112*
usethis::use_course(), *18*, *19*
utils::browseURL(), *7*, *85*

var(), *57*
variable grouping, *127*
vector, *50*, *53*, *62*, *63*

WHOCC, 216
WHONET, 217
will be translated, *8*, *85*, *177*
wisca *(antibiogram)*, 19
wisca(), *23–25*, *27*, *28*, *75*, *89*, *96*, *97*, *136*, *144*, *158*, *196–198*, *206*, *208–210*
wisca_plot *(antibiogram)*, 19

XDR *(mdro)*, 153