Package: AMR (via r-universe)

October 18, 2024

Version 2.1.1.9103

Date 2024-10-18

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, curl, data.table, dplyr, ggplot2, knitr, progress, readxl, rmarkdown, rvest, skimr, tibble, tidyselect, tinytest, vctrs, xml2

VignetteBuilder knitr,rmarkdown

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

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

License GPL-2 | file LICENSE

Encoding UTF-8

LazyData true

RoxygenNote 7.3.2

Roxygen list(markdown = TRUE)

Repository https://msberends.r-universe.dev

RemoteUrl https://github.com/msberends/AMR

RemoteRef HEAD

RemoteSha f424c39474f9a247bc5e27057837fba4cd772347

Contents

`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 antibiotics 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 antibiotics 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(), ...)

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()
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 antibiotics data set: vector_or(colnames(antibiotics), sort = FALSE).
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/reg_hum_atc.htm

Reference Data Publicly Available

All data sets in this AMR package (about microorganisms, antibiotics, SIR interpretation, EUCAST rules, etc.) are publicly and freely available for download in the following formats: R, MS Excel, Apache Feather, Apache Parquet, SPSS, and Stata. We also provide tab-separated plain text files that are machine-readable and suitable for input in any software program, such as laboratory information systems. Please visit our website for the download links. The actual files are of course available on our GitHub repository.

See Also

antibiotics

Examples

```
# all properties:
ab_name("AMX")
ab_atc("AMX")
ab_cid("AMX")
ab_synonyms("AMX")
ab_tradenames("AMX")
ab_group("AMX")
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)

clear_custom_antimicrobials()
```

Arguments

x	a data.frame resembling the antibiotics data set, at least containing columns "ab" and "name"

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 antibiotics data set (containing at the very least columns "ab" and "name") 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.

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 `clear_custom_antimicrobials()` to clear the previously added antimicrobials.

See Also

`add_custom_microorganisms()` to add custom microorganisms.

Examples

```
# returns NA and throws a warning (which is suppressed here):
suppressWarnings(
  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
    # 'antibiotics' 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 antibiotic selectors work
x <- data.frame(
  random_column = "some value",
  coflu = as.sir("S"),
  ampicillin = as.sir("R")
)
x
x[, betalactams()]
```

`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.

Usage

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

Arguments

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

Details

To split ages, the input for the `split_at` 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))

# 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"
    )
}
```

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

Description

Create detailed antibiograms with options for traditional, combination, syndromic, and Bayesian WISCA methods. Based on the approaches of Klinker *et al.*, Barbieri *et al.*, and the Bayesian WISCA model (Weighted-Incidence Syndromic Combination Antibiogram) by Bielicki *et al.*, this function provides flexible output formats including plots and tables, ideal for integration with R Markdown and Quarto reports.

Usage

```
antibiogram(
  x,
  antibiotics = where(is.sir),
  mo_transform = "shortname",
  ab_transform = "name",
  syndromic_group = NULL,
  add_total_n = FALSE,
  only_all_tested = FALSE,
  digits = 0,
  formatting_type = getOption("AMR_antibiogram_formatting_type", 10),
  col_mo = NULL,
  language = get_AMR_locale(),
  minimum = 30,
  combine_SI = TRUE,
  sep = " + ",
  info = interactive()
)

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

## S3 method for class 'antibiogram'
autoplot(object, ...)

## 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 antibiotic results (class 'sir', see as.sir())
antibiotics	vector of any antibiotic name or code (will be evaluated with as.ab(), column name of x, or (any combinations of) antibiotic selectors such as aminoglycosides() or carbapenems(). For combination antibiograms, this can also be set to values separated with "+", such as "TZP+TOB" or "cipro + genta", given that columns resembling such antibiotics exist in x. See *Examples*.
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", "kingdom", "phylum", "class", "order", "family", "genus", "species", "subspecies", "rank", "ref", "oxygen_tolerance", "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.
ab_transform	a character to transform antibiotic input - must be one of the column names of the antibiotics 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*.
add_total_n	a logical to indicate whether total available numbers per pathogen should be added to the table (default is TRUE). This will add the lowest and highest number of available isolate per antibiotic (e.g, if for *E. coli* 200 isolates are available for ciprofloxacin and 150 for amoxicillin, the returned number will be "150-200").
only_all_tested	(for combination antibiograms): a logical to indicate that isolates must be tested for all antibiotics, see *Details*
digits	number of digits to use for rounding the susceptibility percentage
formatting_type	numeric value (1–12) 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())
minimum	the minimum allowed number of available (tested) isolates. Any isolate count lower than minimum will return NA with a warning. The default number of 30 isolates is advised by the Clinical and Laboratory Standards Institute (CLSI) as best practice, see *Source*.
combine_SI	a logical to indicate whether all susceptibility should be determined by results of either S, SDD, or I, instead of only S (default is TRUE)
sep	a separating character for antibiotic columns in combination antibiograms
info	a logical to indicate info should be printed - the default is TRUE only in interactive mode
...	when used in R Markdown or Quarto: arguments passed on to knitr::kable() (otherwise, has no use)
object	an antibiogram() object
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

This function returns 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 in your data set with one of the four available algorithms.

Formatting Type:

The formatting of the 'cells' of the table can be set with the argument `formatting_type`. In these examples, `5` is the susceptibility percentage, `15` the numerator, and `300` the denominator:

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)

The default is `10`, which can be set globally with the package option `AMR_antibiogram_formatting_type`, e.g. `options(AMR_antibiogram_formatting_type = 5)`.

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

Antibiogram Types:

There are four antibiogram types, as summarised by Klinker *et al.* (2021, doi:10.1177/20499361211011373), and they are all supported by `antibiogram()`. Use WISCA whenever possible, since it provides precise coverage estimates by accounting for pathogen incidence and antimicrobial susceptibility. See the section *Why Use WISCA?* on this page.

The four antibiogram types:

1. **Traditional Antibiogram**
   Case example: Susceptibility of *Pseudomonas aeruginosa* to piperacillin/tazobactam (TZP)
   Code example:

```
antibiogram(your_data,
            antibiotics = "TZP")
```

2. **Combination Antibiogram**
   Case example: Additional susceptibility of *Pseudomonas aeruginosa* to TZP + tobramycin versus TZP alone
   Code example:

```
antibiogram(your_data,
            antibiotics = c("TZP", "TZP+TOB", "TZP+GEN"))
```

3. **Syndromic Antibiogram**
   Case example: Susceptibility of *Pseudomonas aeruginosa* to TZP among respiratory specimens (obtained among ICU patients only)
   Code example:

```
antibiogram(your_data,
            antibiotics = penicillins(),
            syndromic_group = "ward")
```

4. **Weighted-Incidence Syndromic Combination Antibiogram (WISCA)**
   WISCA enhances empirical antibiotic selection by weighting the incidence of pathogens in specific clinical syndromes and combining them with their susceptibility data. It provides an estimation of regimen coverage by aggregating pathogen incidences and susceptibilities across potential causative organisms. See also the section *Why Use WISCA?* on this page.
   Case example: Susceptibility of *Pseudomonas aeruginosa* to TZP among respiratory specimens (obtained among ICU patients only) for male patients age >=65 years with heart failure
   Code example:

```
library(dplyr)
your_data %>%
  filter(ward == "ICU" & specimen_type == "Respiratory") %>%
  antibiogram(antibiotics = c("TZP", "TZP+TOB", "TZP+GEN"),
              syndromic_group = ifelse(.$age >= 65 &
                                         .$gender == "Male" &
                                         .$condition == "Heart Disease",
                                       "Study Group", "Control Group"))
```

   WISCA uses a sophisticated Bayesian decision model to combine both local and pooled antimicrobial resistance data. This approach not only evaluates local patterns but can also draw on multi-centre datasets to improve regimen accuracy, even in low-incidence infections like paediatric bloodstream infections (BSIs).

Inclusion in Combination Antibiogram and Syndromic Antibiogram:

Note that for types 2 and 3 (Combination Antibiogram and Syndromic Antibiogram), it is important to realise that susceptibility can be calculated in two ways, which can be set with the `only_all_tested` argument (default is `FALSE`). See this example for two antibiotics, Drug A and Drug B, about how `antibiogram()` works to calculate the %SI:

		only_all_tested = FALSE		only_all_tested = TRUE	
Drug A	Drug B	include as numerator	include as denominator	include as numerator	include as denominator
S or I	S or I	X	X	X	X
R	S or I	X	X	X	X
<NA>	S or I	X	X	-	-
S or I	R	X	X	X	X
R	R	-	X	-	X
<NA>	R	-	-	-	-
S or I	<NA>	X	X	-	-
R	<NA>	-	-	-	-
<NA>	<NA>	-	-	-	-

Plotting:

All types of antibiograms as listed above can be plotted (using `ggplot2::autoplot()` or base R's `plot()` and `barplot()`).

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()`.

Why Use WISCA?

WISCA is a powerful tool for guiding empirical antibiotic therapy because it provides precise coverage estimates by accounting for pathogen incidence and antimicrobial susceptibility. This is particularly important in empirical treatment, where the causative pathogen is often unknown at the outset. Traditional antibiograms do not reflect the weighted likelihood of specific pathogens based on clinical syndromes, which can lead to suboptimal treatment choices.

The Bayesian WISCA, as described by Bielicki *et al.* (2016), improves on earlier methods by handling uncertainties common in smaller datasets, such as low-incidence infections. This method offers a significant advantage by:

1. Pooling Data from Multiple Sources:
   WISCA uses pooled data from multiple hospitals or surveillance sources to overcome limitations of small sample sizes at individual institutions, allowing for more confident selection of narrow-spectrum antibiotics or combinations.
2. Bayesian Framework:
   The Bayesian decision tree model accounts for both local data and prior knowledge (such as inherent resistance patterns) to estimate regimen coverage. It allows for a more precise estimation of coverage, even in cases where susceptibility data is missing or incomplete.
3. Incorporating Pathogen and Regimen Uncertainty:
   WISCA allows clinicians to see the likelihood that an empirical regimen will be effective against all relevant pathogens, taking into account uncertainties related to both pathogen prevalence and antimicrobial resistance. This leads to better-informed, data-driven clinical decisions.
4. Scenarios for Optimising Treatment:
   For hospitals or settings with low-incidence infections, WISCA helps determine whether local data is sufficient or if pooling with external data is necessary. It also identifies statistically significant differences or similarities between antibiotic regimens, enabling clinicians to choose optimal therapies with greater confidence.

WISCA is essential in optimising empirical treatment by shifting away from broad-spectrum antibiotics, which are often overused in empirical settings. By offering precise estimates based on syndromic patterns and pooled data, WISCA supports antimicrobial stewardship by guiding more targeted therapy, reducing unnecessary broad-spectrum use, and combating the rise of antimicrobial resistance.

Source

- 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); doi:10.1093/jac/dkv397

- 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


# Traditional antibiogram -------------------------------------------------

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

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

antibiogram(example_isolates,
  antibiotics = carbapenems(),
  ab_transform = "name",
  mo_transform = "name"
)


# Combined antibiogram ----------------------------------------------------

# combined antibiotics yield higher empiric coverage
antibiogram(example_isolates,
  antibiotics = c("TZP", "TZP+TOB", "TZP+GEN"),
  mo_transform = "gramstain"
)

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

```
)


# Syndromic antibiogram ------------------------------------------------

# the data set could contain a filter for e.g. respiratory specimens
antibiogram(example_isolates,
  antibiotics = c(aminoglycosides(), carbapenems()),
  syndromic_group = "ward"
)

# now define a data set with only E. coli
ex1 <- example_isolates[which(mo_genus() == "Escherichia"), ]

# with a custom language, though this will be determined automatically
# (i.e., this table will be in Spanish on Spanish systems)
antibiogram(ex1,
  antibiotics = aminoglycosides(),
  ab_transform = "name",
  syndromic_group = ifelse(ex1$ward == "ICU",
    "UCI", "No UCI"
  ),
  language = "es"
)


# Weighted-incidence syndromic combination antibiogram (WISCA) ---------

# the data set could contain a filter for e.g. respiratory specimens/ICU
antibiogram(example_isolates,
  antibiotics = c("AMC", "AMC+CIP", "TZP", "TZP+TOB"),
  mo_transform = "gramstain",
  minimum = 10, # this should be >=30, but now just as example
  syndromic_group = ifelse(example_isolates$age >= 65 &
    example_isolates$gender == "M",
  "WISCA Group 1", "WISCA Group 2"
  )
)


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

ureido <- antibiogram(example_isolates,
  antibiotics = ureidopenicillins(),
  ab_transform = "name"
)

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

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

plot(ab1)
plot(ab2)
```

`antibiotics` *Data Sets with 605 Antimicrobial Drugs*

Description

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

Usage

```
antibiotics

antivirals
```

Format

For the antibiotics data set: a tibble with 485 observations and 14 variables::

- `ab`
  Antibiotic 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`
  A short and concise group name, based on WHONET and WHOCC definitions
- `atc`
  ATC codes (Anatomical Therapeutic Chemical) as defined by the WHOCC, like `J01CR02`
- `atc_group1`
  Official pharmacological subgroup (3rd level ATC code) as defined by the WHOCC, like `"Macrolides, lincosamides and streptogramins"`
- `atc_group2`
  Official chemical subgroup (4th level ATC code) as defined by the WHOCC, like `"Macrolides"`
- `abbr`
  List of abbreviations as used in many countries, also for antibiotic susceptibility testing (AST)
- `synonyms`
  Synonyms (often trade names) of a drug, as found in PubChem based on their compound ID
- `oral_ddd`
  Defined Daily Dose (DDD), oral treatment, currently available for 179 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`
  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
- `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 `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`.

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

Direct download:

Like all data sets in this package, these data sets are publicly available for download in the following formats: R, MS Excel, Apache Feather, Apache Parquet, SPSS, and Stata. Please visit our website for the download links. The actual files are of course available on 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/reg_hum_atc.htm).

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

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

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/.

Source

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

- Logical Observation Identifiers Names and Codes (LOINC), Version 2.76 (18 September, 2023). Accessed from https://loinc.org on October 19th, 2023.
- European Commission Public Health PHARMACEUTICALS - COMMUNITY REGISTER: https://ec.europa.eu/health/documents/community-register/html/reg_hum_atc.htm

See Also

microorganisms, intrinsic_resistant

Examples

```
antibiotics
antivirals
```

---

antibiotic_class_selectors

Antibiotic Selectors

Description

These functions allow for filtering rows and selecting columns based on antibiotic test results that are of a specific antibiotic class or group (according to the antibiotics data set), without the need to define the columns or antibiotic abbreviations.

In short, 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: "cefazolin", "kefzol", "CZO" and "J01DB04" will all be picked up by `cephalosporins()`.

Usage

```
ab_class(ab_class, only_sir_columns = FALSE, only_treatable = TRUE, ...)

ab_selector(filter, only_sir_columns = FALSE, only_treatable = TRUE, ...)

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

aminopenicillins(only_sir_columns = FALSE, ...)

antifungals(only_sir_columns = FALSE, ...)

antimycobacterials(only_sir_columns = FALSE, ...)

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

carbapenems(only_sir_columns = FALSE, only_treatable = TRUE, ...)

cephalosporins(only_sir_columns = FALSE, ...)
```

```
cephalosporins_1st(only_sir_columns = FALSE, ...)

cephalosporins_2nd(only_sir_columns = FALSE, ...)

cephalosporins_3rd(only_sir_columns = FALSE, ...)

cephalosporins_4th(only_sir_columns = FALSE, ...)

cephalosporins_5th(only_sir_columns = FALSE, ...)

fluoroquinolones(only_sir_columns = FALSE, ...)

glycopeptides(only_sir_columns = FALSE, ...)

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

lipoglycopeptides(only_sir_columns = FALSE, ...)

macrolides(only_sir_columns = FALSE, ...)

nitrofurans(only_sir_columns = FALSE, ...)

oxazolidinones(only_sir_columns = FALSE, ...)

penicillins(only_sir_columns = FALSE, ...)

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

quinolones(only_sir_columns = FALSE, ...)

rifamycins(only_sir_columns = FALSE, ...)

streptogramins(only_sir_columns = FALSE, ...)

tetracyclines(only_sir_columns = FALSE, ...)

trimethoprims(only_sir_columns = FALSE, ...)

ureidopenicillins(only_sir_columns = FALSE, ...)

administrable_per_os(only_sir_columns = FALSE, ...)

administrable_iv(only_sir_columns = FALSE, ...)

not_intrinsic_resistant(
  only_sir_columns = FALSE,
  col_mo = NULL,
  version_expertrules = 3.3,
```

```
  ...
)
```

Arguments

ab_class	an antimicrobial class or a part of it, such as "carba" and "carbapenems". The columns group, atc_group1 and atc_group2 of the antibiotics data set will be searched (case-insensitive) for this value.
only_sir_columns	a logical to indicate whether only columns of class sir must be selected (default is FALSE), see as.sir()
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)
...	ignored, only in place to allow future extensions
filter	an expression to be evaluated in the antibiotics 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_expertrules	the version number to use for the EUCAST Expert Rules and Intrinsic Resistance guideline. Can be "3.3", "3.2", or "3.1".

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

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

The ab_selector() function can be used to internally filter the antibiotics 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 administrable_per_os() and administrable_iv() functions also rely on the antibiotics data set - antibiotic columns will be matched where a DDD (defined daily dose) for resp. oral and IV treatment is available in the antibiotics data set.

The not_intrinsic_resistant() function can be used to only select antibiotic columns 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 Expert Rules' and 'EUCAST Intrinsic Resistance and Unusual Phenotypes' v3.3 (2021) to determine intrinsic resistance, using the `eucast_rules()` function internally. Because of this determination, this function is quite slow in terms of performance.

Value

(internally) a character vector of column names, with additional class `"ab_selector"`

Full list of supported (antibiotic) 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), 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) 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), simvastatin/fenofibrate (SMF), 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), avibactam (AVB), azidocillin (AZD), azlocillin (AZL), aztreonam (ATM), aztreonam/avibactam (AZA), aztreonam/nacubactam (ANC), bacampicillin (BAM), benzathine benzylpenicillin (BNB), benzathine phenoxymethylpenicillin (BNP), benzylpenicillin (PEN), biapenem (BIA), carbenicillin (CRB), carindacillin (CRN), 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/clavulanic acid (CPC), cefepime/nacubactam (FNC), cefepime/tazobactam (FPT), 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/clavulanic acid (CTC), cefotaxime/sulbactam (CTS), cefotetan (CTT), cefotiam (CTF), cefotiam hexetil (CHE), cefovecin (FOV), cefoxitin (FOX), cefoxitin screening (FOX1), 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), 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), nacubactam (NAC), nafcillin (NAF), oxacillin (OXA), 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), sulbactam (SUL), sulbenicillin (SBC), sultamicillin (SLT6), talampicillin (TAL), tazobactam (TAZ), tebipenem (TBP), temocillin (TEM), ticarcillin (TIC), 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), 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/clavulanic acid (CPC), cefepime/tazobactam (FPT), 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/clavulanic acid (CTC), cefotaxime/sulbactam (CTS), cefotetan (CTT), cefotiam (CTF), cefotiam hexetil (CHE), cefovecin (FOV), cefoxitin (FOX), cefoxitin screening (FOX1), 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), cefte-

zole (CTL), ceftibuten (CTB), 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 (FOX1), 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/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), ceftiofur (TIO), ceftizoxime (CZX), ceftizoxime alapivoxil (CZP), ceftriaxone (CRO), ceftriaxone/beta-lactamase inhibitor (CEB), and latamoxef (LTM)
- `cephalosporins_4th()` can select:
cefepime (FEP), cefepime/clavulanic acid (CPC), cefepime/tazobactam (FPT), 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), 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), levonadifloxacin (LND), lomefloxacin (LOM), marbofloxacin (MAR), metioxate (MXT), miloxacin (MIL), moxifloxacin (MFX), nadifloxacin (NAD), nifuroquine (NIF), norfloxacin (NOR), ofloxacin (OFX), orbifloxacin (ORB), pazufloxacin (PAZ), pefloxacin (PEF), 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), dalbavancin (DAL), norvancomycin (NVA), oritavancin (ORI), ramoplanin (RAM), teicoplanin (TEC), teicoplanin-macromethod (TCM), telavancin (TLV), vancomycin (VAN), and vancomycin-macromethod (VAM)
- `lincosamides()` can select:
acetylmidecamycin (ACM), acetylspiramycin (ASP), clindamycin (CLI), clindamycin inducible screening (CLI1), gamithromycin (GAM), kitasamycin (KIT), lincomycin (LIN), meleumycin (MEL), nafithromycin (ZWK), pirlimycin (PRL), primycin (PRM), solithromycin (SOL), tildipirosin (TIP), tilmicosin (TIL), tulathromycin (TUL), tylosin (TYL), and tylvalosin (TYL1)

- `lipoglycopeptides()` can select:
dalbavancin (DAL), oritavancin (ORI), and telavancin (TLV)
- `macrolides()` can select:
acetylmidecamycin (ACM), acetylspiramycin (ASP), azithromycin (AZM), clarithromycin (CLR), 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)
- `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), avibactam (AVB), azidocillin (AZD), azlocillin (AZL), aztreonam (ATM), aztreonam/avibactam (AZA), aztreonam/nacubactam (ANC), bacampicillin (BAM), benzathine benzylpenicillin (BNB), benzathine phenoxymethylpenicillin (BNP), benzylpenicillin (PEN), carbenicillin (CRB), carindacillin (CRN), cefepime/nacubactam (FNC), 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), nacubactam (NAC), nafcillin (NAF), oxacillin (OXA), 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), sulbactam (SUL), sulbenicillin (SBC), sultamicillin (SLT6), talampicillin (TAL), tazobactam (TAZ), temocillin (TEM), ticarcillin (TIC), and ticarcillin/clavulanic acid (TCC)
- `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), levonadifloxacin (LND), lomefloxacin (LOM), marbofloxacin (MAR), metioxate (MXT), miloxacin (MIL), moxifloxacin (MFX), nadifloxacin (NAD), nalidixic acid (NAL), nemonoxacin (NEM), nifuroquine (NIF), nitroxoline (NTR), norfloxacin (NOR), ofloxacin (OFX), orbifloxacin (ORB), oxolinic acid (OXO), pazufloxacin (PAZ), pefloxacin (PEF), 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)
- `streptogramins()` can select:
  pristinamycin (PRI) and quinupristin/dalfopristin (QDA)
- `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), and tigecycline (TGC)
- `trimethoprims()` can select:
  brodimoprim (BDP), 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)

Reference Data Publicly Available

All data sets in this AMR package (about microorganisms, antibiotics, SIR interpretation, EUCAST rules, etc.) are publicly and freely available for download in the following formats: R, MS Excel, Apache Feather, Apache Parquet, SPSS, and Stata. We also provide tab-separated plain text files that are machine-readable and suitable for input in any software program, such as laboratory information systems. Please visit our website for the download links. The actual files are of course available on our GitHub repository.

Examples

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


# Examples sections below are split into 'dplyr', 'base R', and 'data.table':


# dplyr -----------------------------------------------------------------

if (require("dplyr")) {
  example_isolates %>% select(carbapenems())
}
```

```
if (require("dplyr")) {
  # select columns 'mo', 'AMK', 'GEN', 'KAN' and 'TOB'
  example_isolates %>% select(mo, aminoglycosides())
}

if (require("dplyr")) {
  # select only antibiotic columns with DDDs for oral treatment
  example_isolates %>% select(administrable_per_os())
}

if (require("dplyr")) {
  # get AMR for all aminoglycosides e.g., per ward:
  example_isolates %>%
    group_by(ward) %>%
    summarise(across(aminoglycosides(),
                     resistance))
}
if (require("dplyr")) {
  # You can combine selectors with '&' to be more specific:
  example_isolates %>%
    select(penicillins() & administrable_per_os())
}
if (require("dplyr")) {
  # 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)
}
if (require("dplyr")) {
  # get susceptibility for antibiotics whose name contains "trim":
  example_isolates %>%
    filter(first_isolate()) %>%
    group_by(ward) %>%
    summarise(across(ab_selector(name %like% "trim"), susceptibility))
}
if (require("dplyr")) {
  # this will select columns 'IPM' (imipenem) and 'MEM' (meropenem):
  example_isolates %>%
    select(carbapenems())
}
if (require("dplyr")) {
  # this will select columns 'mo', 'AMK', 'GEN', 'KAN' and 'TOB':
  example_isolates %>%
    select(mo, aminoglycosides())
}
if (require("dplyr")) {
  # any() and all() work in dplyr's filter() too:
  example_isolates %>%
    filter(
      any(aminoglycosides() == "R"),
      all(cephalosporins_2nd() == "R")
```

```
    )
}
if (require("dplyr")) {
  # also works with c():
  example_isolates %>%
    filter(any(c(carbapenems(), aminoglycosides()) == "R"))
}
if (require("dplyr")) {
  # not setting any/all will automatically apply all():
  example_isolates %>%
    filter(aminoglycosides() == "R")
}
if (require("dplyr")) {
  # this will select columns 'mo' and all antimycobacterial drugs ('RIF'):
  example_isolates %>%
    select(mo, ab_class("mycobact"))
}
if (require("dplyr")) {
  # get bug/drug combinations for only glycopeptides in Gram-positives:
  example_isolates %>%
    filter(mo_is_gram_positive()) %>%
    select(mo, glycopeptides()) %>%
    bug_drug_combinations() %>%
    format()
}
if (require("dplyr")) {
  data.frame(
    some_column = "some_value",
    J01CA01 = "S"
  ) %>% # ATC code of ampicillin
    select(penicillins()) # only the 'J01CA01' column will be selected
}
if (require("dplyr")) {
  # 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)
}


# 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 antibiotic columns 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 antibiotic 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()]

# ab_selector() applies a filter in the `antibiotics` data set and is thus
# very flexible. For instance, to select antibiotic columns with an oral DDD
# of at least 1 gram:
example_isolates[, ab_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 AB 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 AB 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 antibiotic drug code of one or more antibiotics. The data set antibiotics will be searched for abbreviations, official names and synonyms (brand names).

Usage

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

is.ab(x)
```

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

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().

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/reg_hum_atc.htm).

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

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

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/.

Reference Data Publicly Available

All data sets in this AMR package (about microorganisms, antibiotics, SIR interpretation, EUCAST rules, etc.) are publicly and freely available for download in the following formats: R, MS Excel, Apache Feather, Apache Parquet, SPSS, and Stata. We also provide tab-separated plain text files that are machine-readable and suitable for input in any software program, such as laboratory information systems. Please visit our website for the download links. The actual files are of course available on our GitHub repository.

See Also

- antibiotics 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 123")
as.ab("ERYT")
as.ab("ERY")
as.ab("eritromicine") # spelled wrong, yet works
as.ab("Erythrocin") # trade name
as.ab("Romycin") # trade name

# spelling from different languages and dyslexia are no problem
ab_atc("ceftriaxon")
ab_atc("cephtriaxone") # small spelling error
ab_atc("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/reg_hum_atc.htm).

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

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

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/.

Reference Data Publicly Available

All data sets in this AMR package (about microorganisms, antibiotics, SIR interpretation, EUCAST rules, etc.) are publicly and freely available for download in the following formats: R, MS Excel, Apache Feather, Apache Parquet, SPSS, and Stata. We also provide tab-separated plain text files that are machine-readable and suitable for input in any software program, such as laboratory information systems. Please visit our website for the download links. The actual files are of course available on 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

Format

An object of class `disk` (inherits from `integer`) of length 1.

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. Values higher than 50 but lower than 100 will be maximised to 50. All others input values outside the 0-50 range will return `NA`.

`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")

is.mic(x)

NA_mic_

rescale_mic(x, mic_range, keep_operators = "edges", as.mic = TRUE)
```

```
## 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.
mic_range	a manual range to limit the MIC values, e.g., mic_range = c(0.001, 32). Use NA to set no limit 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 FALSE
...	arguments passed on to methods

Details

To interpret MIC values as SIR values, use as.sir() on MIC values. It supports guidelines from EUCAST (2011-2024) and CLSI (2011-2024).

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 functions of the `stats` package 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).

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_`.

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 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))
```

```
# 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 kingdoms Animalia, Archaea, Bacteria, Chromista, and Protozoa, and most microbial species from the kingdom Fungi (see *Source*). The input can be almost anything: a full name (like `"Staphylococcus aureus"`), an abbreviated name (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()
```

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. 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. 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 kingdom and human pathogenicity.
keep_synonyms	a logical to indicate if old, previously valid taxonomic names must be preserved and not be corrected to currently accepted names. The default is FALSE, which will return a note if old 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 kingdom 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 old 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
\----> kingdom: 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 kingdom, 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 old, previously accepted taxonomic names.

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. caseolyticus*, *S. chromogenes*, *S. cohnii*, *S. cohnii cohnii*, *S. cohnii urealyticum*, *S. cohnii urealyticus*, *S. condimenti*, *S. croceilyticus*, *S. debuckii*, *S. devriesei*, *S. durrellii*, *S. edaphicus*, *S. epidermidis*, *S. equorum*, *S. equorum equorum*, *S. equorum linens*, *S. felis*, *S. fleurettii*, *S. gallinarum*, *S. haemolyticus*, *S. hominis*, *S. hominis hominis*, *S. hominis novobiosepticus*, *S. jettensis*, *S. kloosii*, *S. lentus*, *S. lloydii*, *S. lugdunensis*, *S. marylandisciuri*, *S. massiliensis*, *S. microti*, *S. muscae*, *S. nepalensis*, *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. veratri*, *S. vitulinus*, *S. vitulus*, *S. warneri*, and *S. xylosus*

- Coagulase-positive: *S. agnetis*, *S. argenteus*, *S. coagulans*, *S. cornubiensis*, *S. delphini*, *S. hyicus*, *S. hyicus chromogenes*, *S. hyicus hyicus*, *S. intermedius*, *S. lutrae*, *S. pseudintermedius*, *S. roterodami*, *S. schleiferi coagulans*, *S. schweitzeri*, *S. simiae*, and *S. singaporensis*

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*, *S. salivarius salivarius*, and *S. salivarius thermophilus*
- 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

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

- Parte, AC *et al.* (2020). **List of Prokaryotic names with Standing in Nomenclature (LPSN) moves to the DSMZ.** International Journal of Systematic and Evolutionary Microbiology, 70, 5607-5612; doi:10.1099/ijsem.0.004332. Accessed from `https://lpsn.dsmz.de` on June 24th, 2024.
- 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 June 24th, 2024.
- GBIF Secretariat (2023). GBIF Backbone Taxonomy. Checklist dataset doi:10.15468/39omei. Accessed from `https://www.gbif.org` on June 24th, 2024.
- 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 July 16th, 2024.
- 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://phinvads.cdc.gov`
- 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 kingdom 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*, *Apophysomyces*, *Arthroderma*, *Aspergillus*, *Aureobasidium*, *Basidiobolus*, *Beauveria*, *Bipolaris*, *Blastobotrys*, *Blastocystis*, *Blastomyces*, *Candida*, *Capillaria*, *Chaetomium*, *Chilomastix*, *Chrysonilia*, *Chrysosporium*, *Cladophialophora*, *Cladosporium*, *Clavispora*, *Coccidioides*, *Cokeromyces*, *Conidiobolus*, *Coniochaeta*, *Contracaecum*, *Cordylobia*, *Cryptococcus*, *Cryptosporidium*, *Cunninghamella*, *Curvularia*, *Cyberlindnera*, *Debaryozyma*, *Demodex*, *Dermatobia*, *Dientamoeba*, *Diphyllobothrium*, *Dirofilaria*, *Echinostoma*, *Entamoeba*, *Enterobius*, *Epidermophyton*, *Exidia*, *Exophiala*, *Exserohilum*, *Fasciola*, *Fonsecaea*, *Fusarium*, *Geotrichum*, *Giardia*, *Graphium*, *Haloarcula*, *Halobacterium*, *Halococcus*, *Hansenula*, *Hendersonula*, *Heterophyes*, *Histomonas*, *Histoplasma*, *Hortaea*, *Hymenolepis*, *Hypomyces*, *Hysterothylacium*, *Kloeckera*, *Kluyveromyces*, *Kodamaea*, *Lacazia*, *Leishmania*, *Lichtheimia*, *Lodderomyces*, *Lomentospora*, *Madurella*, *Malassezia*, *Malbranchea*, *Metagonimus*, *Meyerozyma*, *Microsporidium*, *Microsporum*, *Millerozyma*, *Mortierella*, *Mucor*, *Mycocentrospora*, *Nannizzia*, *Necator*, *Nectria*, *Ochroconis*, *Oesophagostomum*, *Oidiodendron*, *Opisthorchis*, *Paecilomyces*, *Paracoccidioides*, *Pediculus*, *Penicillium*, *Phaeoacremonium*, *Phaeomoniella*, *Phialophora*, *Phlebotomus*, *Phoma*, *Pichia*, *Piedraia*, *Pithomyces*, *Pityrosporum*, *Pneumocystis*, *Pseudallescheria*, *Pseudoscopulariopsis*, *Pseudoterranova*, *Pulex*, *Purpureocillium*, *Quambalaria*, *Rhinocladiella*, *Rhizomucor*, *Rhizopus*, *Rhodotorula*, *Saccharomyces*, *Saksenaea*, *Saprochaete*, *Sarcoptes*, *Scedosporium*, *Schistosoma*, *Schizosaccharomyces*, *Scolecobasidium*, *Scopulariopsis*, *Scytalidium*, *Spirometra*, *Sporobolomyces*, *Sporopachydermia*, *Sporothrix*, *Sporotrichum*, *Stachybotrys*, *Strongyloides*, *Syncephalastrum*, *Syngamus*, *Taenia*, *Talaromyces*, *Teleomorph*, *Toxocara*, *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.

Reference Data Publicly Available

All data sets in this AMR package (about microorganisms, antibiotics, SIR interpretation, EUCAST rules, etc.) are publicly and freely available for download in the following formats: R, MS Excel, Apache Feather, Apache Parquet, SPSS, and Stata. We also provide tab-separated plain text files that are machine-readable and suitable for input in any software program, such as laboratory information systems. Please visit our website for the download links. The actual files are of course available on 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` *Translate MIC and Disk Diffusion to 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`.

These breakpoints are currently implemented:

- For **clinical microbiology**: EUCAST 2011-2024 and CLSI 2011-2024;
- For **veterinary microbiology**: EUCAST 2021-2024 and CLSI 2019-2024;
- For **ECOFFs** (Epidemiological Cut-off Values): EUCAST 2020-2024 and CLSI 2022-2024.

All breakpoints used for interpretation are available in our clinical_breakpoints data set.

Usage

```
as.sir(x, ...)

NA_sir_

is.sir(x)

is_sir_eligible(x, threshold = 0.05)

## Default S3 method:
as.sir(
  x,
  S = "^(S|U)+$",
  I = "^(I)+$",
  R = "^(R)+$",
  NI = "^(N|NI|V)+$",
  SDD = "^(SDD|D|H)+$",
  ...
)

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

```
  mo = NULL,
  ab = deparse(substitute(x)),
  guideline = getOption("AMR_guideline", "EUCAST"),
  uti = NULL,
  conserve_capped_values = FALSE,
  add_intrinsic_resistance = FALSE,
  reference_data = AMR::clinical_breakpoints,
  include_screening = getOption("AMR_include_screening", FALSE),
  include_PKPD = getOption("AMR_include_PKPD", TRUE),
  breakpoint_type = getOption("AMR_breakpoint_type", "human"),
  host = NULL,
  verbose = FALSE,
  ...
)

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

## S3 method for class 'data.frame'
as.sir(
  x,
  ...,
  col_mo = NULL,
  guideline = getOption("AMR_guideline", "EUCAST"),
  uti = NULL,
  conserve_capped_values = FALSE,
  add_intrinsic_resistance = FALSE,
  reference_data = AMR::clinical_breakpoints,
  include_screening = getOption("AMR_include_screening", FALSE),
  include_PKPD = getOption("AMR_include_PKPD", TRUE),
  breakpoint_type = getOption("AMR_breakpoint_type", "human"),
  host = NULL,
  verbose = FALSE
)
```

```
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: names of columns to apply as.sir() on (supports tidy selection such as column1:column4). Otherwise: arguments passed on to methods.
threshold	maximum fraction of invalid antimicrobial interpretations of x, see *Examples*
S, I, R, NI, SDD	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.
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	defaults to EUCAST 2024 (the latest implemented EUCAST guideline in the clinical_breakpoints data set), but can be set with the package option AMR_guideline. Currently supports EUCAST (2011-2024) and CLSI (2011-2024), see *Details*.
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. See *Examples*.
conserve_capped_values	a logical to indicate that MIC values starting with ">" (but not ">=") must always return "R" , and that MIC values starting with "<" (but not "<=") must always return "S"
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 contain a data set that is equal in structure to the clinical_breakpoints data set (same column names and column types). Please note that the guideline argument will be ignored when reference_data is manually set.
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 20 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).

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

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().

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.
   - Using dplyr, SIR interpretation can be done very easily with either:

```
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
```

- Operators like "<=" will be stripped before interpretation. When using `conserve_capped_values = TRUE`, an MIC value of e.g. ">2" will always return "R", even if the breakpoint according to the chosen guideline is ">=4". This is to prevent that capped values from raw laboratory data would not be treated conservatively. The default behaviour (`conserve_capped_values = FALSE`) considers ">2" to be lower than ">=4" and might in this case return "S" or "I".

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.
   - Using `dplyr`, SIR interpretation can be done very easily with either:

```
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
```

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 (or tibble if the `tibble` package is installed) with all results of the last `as.sir()` call.

Supported Guidelines:

For interpreting MIC values as well as disk diffusion diameters, currently implemented guidelines are for **clinical microbiology**: EUCAST 2011-2024 and CLSI 2011-2024, and for **veterinary microbiology**: EUCAST 2021-2024 and CLSI 2019-2024.

Thus, the `guideline` argument must be set to e.g., `"EUCAST 2024"` or `"CLSI 2024"`. By simply using `"EUCAST"` (the default) or `"CLSI"` as input, the latest included version of that guideline will automatically be selected. You can set your own data set using the `reference_data` argument. The `guideline` argument will then be ignored.

You can 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)
```

For veterinary guidelines, these might be the best options:

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

When applying veterinary breakpoints (by setting `host` or by setting `breakpoint_type = "animal"`), the CLSI VET09 guideline will be applied to cope with missing animal species-specific breakpoints.

After Interpretation:

After using `as.sir()`, you can use the `eucast_rules()` defined by EUCAST 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 interpreting multidrug-resistant organisms here.

Machine-Readable Clinical Breakpoints:

The repository of this package contains a machine-readable version of all guidelines. This is a CSV file consisting of 34 063 rows and 14 columns. This file is machine-readable, since it contains one row for every unique combination of the test method (MIC or disk diffusion), the antimicrobial drug and the microorganism. **This allows for easy implementation of these rules in laboratory information systems (LIS)**. Note that it only contains interpretation guidelines for humans - interpretation guidelines from CLSI for animals were removed.

Other:

The function `is.sir()` detects if the input contains class `sir`. If the input is a data.frame, it iterates over all columns 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% invalid antimicrobial interpretations (not S and/or I and/or R and/or NI and/or SDD), 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 as shown below (`https://www.eucast.org/newsiandr`):

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

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

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

Reference Data Publicly Available

All data sets in this AMR package (about microorganisms, antibiotics, SIR interpretation, EUCAST rules, etc.) are publicly and freely available for download in the following formats: R, MS Excel, Apache Feather, Apache Parquet, SPSS, and Stata. We also provide tab-separated plain text files that are machine-readable and suitable for input in any software program, such as laboratory information systems. Please visit our website for the download links. The actual files are of course available on our GitHub repository.

Source

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

- **CLSI M39: Analysis and Presentation of Cumulative Antimicrobial Susceptibility Test Data**, 2011-2024, *Clinical and Laboratory Standards Institute* (CLSI). https://clsi.org/standards/products/microbiology/documents/m39/.
- **CLSI M100: Performance Standard for Antimicrobial Susceptibility Testing**, 2011-2024, *Clinical and Laboratory Standards Institute* (CLSI). https://clsi.org/standards/products/microbiology/documents/m100/.
- **CLSI VET01: Performance Standards for Antimicrobial Disk and Dilution Susceptibility Tests for Bacteria Isolated From Animals**, 2019-2024, *Clinical and Laboratory Standards Institute* (CLSI). https://clsi.org/standards/products/veterinary-medicine/documents/vet01/.
- **CLSI VET09: Understanding Susceptibility Test Data as a Component of Antimicrobial Stewardship in Veterinary Settings**, 2019-2024, *Clinical and Laboratory Standards Institute* (CLSI). https://clsi.org/standards/products/veterinary-medicine/documents/vet09/.
- **EUCAST Breakpoint tables for interpretation of MICs and zone diameters**, 2011-2024, *European Committee on Antimicrobial Susceptibility Testing* (EUCAST). https://www.eucast.org/clinical_breakpoints.
- **WHONET** as a source for machine-reading the clinical breakpoints (read more here), 1989-2024, *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) # see all SIR results at a glance

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

# 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))
)


## 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))

  # 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 = "CLSI")
  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 --------------------------------------------------

as.sir(df_wide)
```

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

# 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"
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:
  # 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 antibiotic), 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 antibiotics, 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: `vector_or(colnames(antivirals), sort = FALSE)`.

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

Reference Data Publicly Available

All data sets in this AMR package (about microorganisms, antibiotics, SIR interpretation, EUCAST rules, etc.) are publicly and freely available for download in the following formats: R, MS Excel, Apache Feather, Apache Parquet, SPSS, and Stata. We also provide tab-separated plain text files that are machine-readable and suitable for input in any software program, such as laboratory information systems. Please visit our website for the download links. The actual files are of course available on 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, ...)

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

Arguments

x	a data set with antibiotic 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()
...	arguments passed on to FUN
translate_ab	a character of length 1 containing column names of the antibiotics data set
language	language of the returned text - 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	a logical to indicate whether values S, SDD, and I should be summed, so resistance will be based on only R - the default is TRUE

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. Use combine_SI = TRUE (default) to test R vs. S+I and combine_SI = FALSE to test R+I vs. S. 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", and "total".

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 dataset contain breakpoints for humans, 7 different animal groups, and ECOFFs.

These breakpoints are currently implemented:

- For **clinical microbiology**: EUCAST 2011-2024 and CLSI 2011-2024;
- For **veterinary microbiology**: EUCAST 2021-2024 and CLSI 2019-2024;
- For **ECOFFs** (Epidemiological Cut-off Values): EUCAST 2020-2024 and CLSI 2022-2024.

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

Usage

```
clinical_breakpoints
```

Format

A tibble with 34 063 observations and 14 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` from 1 (subspecies/infraspecies) to 5 (unknown microorganism)
- `ab`
  Antibiotic 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"
- `breakpoint_R`
  Highest MIC value or lowest number of millimetres that leads to "R"
- `uti`
  A logical value (`TRUE`/`FALSE`) to indicate whether the rule applies to a urinary tract infection (UTI)
- `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 24 breakpoints.

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.

Imported from WHONET:

Clinical 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.

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:

Like all data sets in this package, this data set is publicly available for download in the following formats: R, MS Excel, Apache Feather, Apache Parquet, SPSS, and Stata. Please visit our website for the download links. The actual files are of course available on our GitHub repository. They allow for machine reading EUCAST and CLSI guidelines, which is almost impossible with the MS Excel and PDF files distributed by EUCAST and CLSI, though initiatives have started to overcome these burdens.

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 using the breakpoints for own use.

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)

count_susceptible(..., only_all_tested = FALSE)

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 = TRUE
)
```

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 antibiotics, see section *Combination Therapy* below

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

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

language
: language of the returned text - 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
: a logical to indicate whether all values of S, SDD, and I must be merged into one, so the output only consists of S+SDD+I vs. R (susceptible vs. resistant) - the default is `TRUE`

Details

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

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

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

The function `count_df()` takes any variable from `data` that has an `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 as shown below (https://www.eucast.org/newsiandr):

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

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

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

Combination Therapy

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

		only_all_tested = FALSE		only_all_tested = TRUE	
Drug A	Drug B	include as numerator	include as denominator	include as numerator	include as denominator
S or I	S or I	X	X	X	X
R	S or I	X	X	X	X
<NA>	S or I	X	X	-	-
S or I	R	X	X	X	X
R	R	-	X	-	X
<NA>	R	-	-	-	-
S or I	<NA>	X	X	-	-
R	<NA>	-	-	-	-
<NA>	<NA>	-	-	-	-

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

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

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

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

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

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_eucast_rules` *Define Custom EUCAST Rules*

Description

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

Usage

```
custom_eucast_rules(...)
```

Arguments

`...`	rules in formula notation, see below for instructions, and in *Examples*

Details

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

Value

A list containing the custom rules

How it works

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_eucast_rules(TZP == "S" ~ aminopenicillins == "S",
                         TZP == "R" ~ aminopenicillins == "R")
```

These are two custom EUCAST 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 EUCAST 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

eucast_rules(df, rules = "custom", custom_rules = x, info = FALSE)
#>                       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", "kingdom", "phylum", "class", "order", "family", "genus", "species", "subspecies", "rank", "ref", "oxygen_tolerance", "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_eucast_rules(TZP == "S" & genus == "Klebsiella" ~ aminopenicillins == "S",
                         TZP == "R" & genus == "Klebsiella" ~ aminopenicillins == "R")

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

Usage of multiple antibiotics and antibiotic group names:

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

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

```
x <- custom_eucast_rules(TZP == "R" ~ c(aminopenicillins, ureidopenicillins) == "R")
x
#> A set of custom EUCAST rules:
#>
#>   1. If TZP is "R" then set to "R":
#>    amoxicillin (AMX), ampicillin (AMP), azlocillin (AZL), mezlocillin (MEZ), piperacillin (PIP), p
```

These 30 antibiotic groups are allowed in the rules (case-insensitive) and can be used in any combination:

- aminoglycosides
  (amikacin, amikacin/fosfomycin, apramycin, arbekacin, astromicin, bekanamycin, dibekacin, framycetin, gentamicin, gentamicin-high, habekacin, hygromycin, isepamicin, kanamycin, kanamycin-high, kanamycin/cephalexin, micronomicin, neomycin, netilmicin, pentisomicin, plazomicin, propikacin, ribostamycin, sisomicin, streptoduocin, streptomycin, streptomycin-high, tobramycin, and tobramycin-high)
- aminopenicillins
  (amoxicillin 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, simvastatin/fenofibrate, sodium aminosalicylate, streptomycin/isoniazid, terizidone, thioacetazone, thioacetazone/isoniazid, tiocarlide, and viomycin)
- betalactams
  (amoxicillin, amoxicillin/clavulanic acid, amoxicillin/sulbactam, ampicillin, ampicillin/sulbactam,

apalcillin, aspoxicillin, avibactam, azidocillin, azlocillin, aztreonam, aztreonam/avibactam, aztreonam/nacubactam, bacampicillin, benzathine benzylpenicillin, benzathine phenoxymethylpenicillin, benzylpenicillin, biapenem, carbenicillin, carindacillin, cefacetrile, cefaclor, cefadroxil, cefalexin, cefaloridine, cefalotin, cefamandole, cefapirin, cefatrizine, cefazedone, cefazolin, cefcapene, cefcapene pivoxil, cefdinir, cefditoren, cefditoren pivoxil, cefepime, cefepime/clavulanic acid, cefepime/nacubactam, cefepime/tazobactam, cefetamet, cefetamet pivoxil, cefetecol, cefetrizole, cefiderocol, cefixime, cefmenoxime, cefmetazole, cefodizime, cefonicid, cefoperazone, cefoperazone/sulbactam, ceforanide, cefoselis, cefotaxime, cefotaxime/clavulanic acid, cefotaxime/sulbactam, cefotetan, cefotiam, cefotiam hexetil, cefovecin, cefoxitin, cefoxitin screening, 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, 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, nacubactam, nafcillin, oxacillin, panipenem, penamecillin, penicillin/novobiocin, penicillin/sulbactam, pheneticillin, phenoxymethylpenicillin, piperacillin, piperacillin/sulbactam, piperacillin/tazobactam, piridicillin, pivampicillin, pivmecillinam, procaine benzylpenicillin, propicillin, razupenem, ritipenem, ritipenem acoxil, sarmoxicillin, sulbactam, sulbenicillin, sultamicillin, talampicillin, tazobactam, tebipenem, temocillin, ticarcillin, and ticarcillin/clavulanic acid)

- carbapenems
  (biapenem, doripenem, ertapenem, imipenem, imipenem/EDTA, imipenem/relebactam, meropenem, meropenem/nacubactam, meropenem/vaborbactam, panipenem, razupenem, ritipenem, ritipenem acoxil, and tebipenem)
- cephalosporins
  (cefacetrile, cefaclor, cefadroxil, cefalexin, cefaloridine, cefalotin, cefamandole, cefapirin, cefatrizine, cefazedone, cefazolin, cefcapene, cefcapene pivoxil, cefdinir, cefditoren, cefditoren pivoxil, cefepime, cefepime/clavulanic acid, cefepime/tazobactam, cefetamet, cefetamet pivoxil, cefetecol, cefetrizole, cefiderocol, cefixime, cefmenoxime, cefmetazole, cefodizime, cefonicid, cefoperazone, cefoperazone/sulbactam, ceforanide, cefoselis, cefotaxime, cefotaxime/clavulanic acid, cefotaxime/sulbactam, cefotetan, cefotiam, cefotiam hexetil, cefovecin, cefoxitin, cefoxitin screening, 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, ceftiofur, ceftizoxime, ceftizoxime alapivoxil, ceftobiprole, ceftobiprole medocaril, ceftolozane/tazobactam, ceftriaxone, ceftriaxone/beta-lactamase inhibitor, cefuroxime, cefuroxime axetil, cephradine, latamoxef, 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, 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/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, ceftiofur, ceftizoxime, ceftizoxime alapivoxil, ceftriaxone, ceftriaxone/beta-lactamase inhibitor, and latamoxef)
- cephalosporins_4th
  (cefepime, cefepime/clavulanic acid, cefepime/tazobactam, 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/clavulanic acid, cefepime/tazobactam, cefetamet, cefetamet pivoxil, cefetecol, cefetrizole, cefiderocol, cefixime, cefmenoxime, cefmetazole, cefodizime, cefonicid, cefoperazone, cefoperazone/sulbactam, ceforanide, cefoselis, cefotaxime, cefotaxime/clavulanic acid, cefotaxime/sulbactam, cefotetan, cefotiam, cefotiam hexetil, cefovecin, cefoxitin, cefoxitin screening, 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, ceftiofur, ceftizoxime, ceftizoxime alapivoxil, ceftobiprole, ceftobiprole medocaril, ceftolozane/tazobactam, ceftriaxone, ceftriaxone/beta-lactamase inhibitor, cefuroxime, cefuroxime axetil, cephradine, latamoxef, and loracarbef)
- fluoroquinolones
  (besifloxacin, ciprofloxacin, clinafloxacin, danofloxacin, delafloxacin, difloxacin, enoxacin, enrofloxacin, finafloxacin, fleroxacin, garenoxacin, gatifloxacin, gemifloxacin, grepafloxacin, lascufloxacin, levofloxacin, levonadifloxacin, lomefloxacin, marbofloxacin, metioxate, miloxacin, moxifloxacin, nadifloxacin, nifuroquine, norfloxacin, ofloxacin, orbifloxacin, pazufloxacin, pefloxacin, pradofloxacin, premafloxacin, prulifloxacin, rufloxacin, sarafloxacin, sitafloxacin, sparfloxacin, temafloxacin, tilbroquinol, tioxacin, tosufloxacin, and trovafloxacin)
- glycopeptides
  (avoparcin, dalbavancin, norvancomycin, oritavancin, ramoplanin, teicoplanin, teicoplanin-macromethod, telavancin, vancomycin, and vancomycin-macromethod)
- glycopeptides_except_lipo
  (avoparcin, norvancomycin, ramoplanin, teicoplanin, teicoplanin-macromethod, vancomycin, and vancomycin-macromethod)
- lincosamides
  (acetylmidecamycin, acetylspiramycin, clindamycin, clindamycin inducible screening, gamithromycin, kitasamycin, lincomycin, meleumycin, nafithromycin, pirlimycin, primycin, solithromycin, tildipirosin, tilmicosin, tulathromycin, tylosin, and tylvalosin)
- lipoglycopeptides
  (dalbavancin, oritavancin, and telavancin)
- macrolides
  (acetylmidecamycin, acetylspiramycin, azithromycin, clarithromycin, 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)

- 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, avibactam, azidocillin, azlocillin, aztreonam, aztreonam/avibactam, aztreonam/nacubactam, bacampicillin, benzathine benzylpenicillin, benzathine phenoxymethylpenicillin, benzylpenicillin, carbenicillin, carindacillin, cefepime/nacubactam, ciclacillin, clometocillin, cloxacillin, dicloxacillin, epicillin, flucloxacillin, hetacillin, lenampicillin, mecillinam, metampicillin, meticillin, mezlocillin, mezlocillin/sulbactam, nacubactam, nafcillin, oxacillin, penamecillin, penicillin/novobiocin, penicillin/sulbactam, pheneticillin, phenoxymethylpenicillin, piperacillin, piperacillin/sulbactam, piperacillin/tazobactam, piridicillin, pivampicillin, pivmecillinam, procaine benzylpenicillin, propicillin, sarmoxicillin, sulbactam, sulbenicillin, sultamicillin, talampicillin, tazobactam, temocillin, ticarcillin, and ticarcillin/clavulanic acid)
- 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, levonadifloxacin, lomefloxacin, marbofloxacin, metioxate, miloxacin, moxifloxacin, nadifloxacin, nalidixic acid, nemonoxacin, nifuroquine, nitroxoline, norfloxacin, ofloxacin, orbifloxacin, oxolinic acid, pazufloxacin, pefloxacin, 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)
- streptogramins
(pristinamycin and quinupristin/dalfopristin)
- tetracyclines
(cetocycline, chlortetracycline, clomocycline, demeclocycline, doxycycline, eravacycline, lymecycline, metacycline, minocycline, omadacycline, oxytetracycline, penimepicycline, rolitetracycline, sarecycline, tetracycline, and tigecycline)
- tetracyclines_except_tgc
(cetocycline, chlortetracycline, clomocycline, demeclocycline, doxycycline, eravacycline, lymecycline, metacycline, minocycline, omadacycline, oxytetracycline, penimepicycline, rolitetracycline, sarecycline, and tetracycline)
- trimethoprims
(brodimoprim, 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)

Examples

```
x <- custom_eucast_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,
  verbose = TRUE
)

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

---

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 503 observations and 9 variables:

- ab
  Antibiotic 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", or "oral"
- `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 13, 12, or 11

Details

Like all data sets in this package, this data set is publicly available for download in the following formats: R, MS Excel, Apache Feather, Apache Parquet, SPSS, and Stata. Please visit our website for the download links. The actual files are of course available on our GitHub repository.

Examples

```
dosage
```

`eucast_rules` *Apply EUCAST Rules*

Description

Apply rules for clinical breakpoints and intrinsic resistance as defined by 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 EUCAST rules are applied, some non-EUCAST rules can applied at default, see *Details*.

Usage

```
eucast_rules(
  x,
  col_mo = NULL,
  info = interactive(),
  rules = getOption("AMR_eucastrules", default = c("breakpoints", "expert")),
  verbose = FALSE,
  version_breakpoints = 12,
  version_expertrules = 3.3,
  ampc_cephalosporin_resistance = NA,
  only_sir_columns = FALSE,
  custom_rules = NULL,
  ...
)

eucast_dosage(ab, administration = "iv", version_breakpoints = 12)
```

Arguments

x	a data set with antibiotic 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", "expert", "other", "custom", "all", and defaults to c("breakpoints", "expert"). The default value can be set to another value using the package option AMR_eucastrules: options(AMR_eucastrules = "all"). If using "custom", be sure to fill in argument custom_rules too. Custom rules can be created with custom_eucast_rules().
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 "12.0", "11.0", or "10.0".
version_expertrules	the version number to use for the EUCAST Expert Rules and Intrinsic Resistance guideline. Can be "3.3", "3.2", or "3.1".
ampc_cephalosporin_resistance	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 version 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*, *Klebsiella aerogenes*, *Morganella morganii*, *Providencia*, and *Serratia*.

`only_sir_columns`
: a logical to indicate whether only antibiotic columns must be detected that were transformed to class sir (see as.sir()) on beforehand (default is FALSE)

`custom_rules`
: custom rules to apply, created with custom_eucast_rules()

`...`
: column name of an antibiotic, see section *Antibiotics* below

`ab`
: any (vector of) text that can be coerced to a valid antibiotic drug code with as.ab()

`administration`
: route of administration, either "im", "iv", or "oral"

Details

Note: This function does not translate MIC 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 EUCAST rules is located here: https://github.com/msberends/AMR/blob/main/data-raw/eucast_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; the original EUCAST rules v3.1 and v3.2 did not yet contain this new taxonomic name. The AMR package contains the full microbial taxonomy updated until June 24th, 2024, see microorganisms.

Custom Rules:

Custom rules can be created using custom_eucast_rules(), e.g.:

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

eucast_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 `eucast_rules(..., rules = "all")`. You can also set the package option `AMR_eucastrules`, i.e. run `options(AMR_eucastrules = "all")`.

Value

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

Antibiotics

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

The following antibiotics are eligible for the functions `eucast_rules()` and `mdro()`. These are shown below in the format 'name (`antimicrobial ID`, ATC code)', sorted alphabetically:

Amikacin (`AMK`, J01GB06), amoxicillin (`AMX`, J01CA04), amoxicillin/clavulanic acid (`AMC`, J01CR02), ampicillin (`AMP`, J01CA01), ampicillin/sulbactam (`SAM`, J01CR01), apramycin (`APR`, QA07AA92), arbekacin (`ARB`, J01GB12), aspoxicillin (`APX`, J01CA19), azidocillin (`AZD`, J01CE04), azithromycin (`AZM`, J01FA10), azlocillin (`AZL`, J01CA09), aztreonam (`ATM`, J01DF01), bacampicillin (`BAM`, J01CA06), bekanamycin (`BEK`, J01GB13), benzathine benzylpenicillin (`BNB`, J01CE08), benzathine phenoxymethylpenicillin (`BNP`, J01CE10), benzylpenicillin (`PEN`, J01CE01), besifloxacin (`BES`, S01AE08), biapenem (`BIA`, J01DH05), carbenicillin (`CRB`, J01CA03), carindacillin (`CRN`, J01CA05), cefacetrile (`CAC`, J01DB10), cefaclor (`CEC`, J01DC04), cefadroxil (`CFR`, J01DB05), cefalexin (`LEX`, J01DB01), cefaloridine (`RID`, J01DB02), cefalotin (`CEP`, J01DB03), cefamandole (`MAN`, J01DC03), cefapirin (`HAP`, J01DB08), cefatrizine (`CTZ`, J01DB07), cefazedone (`CZD`, J01DB06), cefazolin (`CZO`, J01DB04), cefcapene (`CCP`, J01DD17), cefdinir (`CDR`, J01DD15), cefditoren (`DIT`, J01DD16), cefepime (`FEP`, J01DE01), cefetamet (`CAT`, J01DD10), cefiderocol (`FDC`, J01DI04), cefixime (`CFM`, J01DD08), cefmenoxime (`CMX`, J01DD05), cefmetazole (`CMZ`, J01DC09), cefodizime (`DIZ`, J01DD09), cefonicid (`CID`, J01DC06), cefoperazone (`CFP`, J01DD12), cefoperazone/sulbactam (`CSL`, J01DD62), ceforanide (`CND`, J01DC11), cefotaxime (`CTX`, J01DD01), cefotaxime/clavulanic acid (`CTC`, J01DD51), cefotetan (`CTT`, J01DC05), cefotiam (`CTF`, J01DC07), cefovecin (`FOV`, QJ01DD91), cefoxitin (`FOX`, J01DC01), cefozopran (`ZOP`, J01DE03), cefpiramide (`CPM`, J01DD11), cefpirome (`CPO`, J01DE02), cefpodoxime (`CPD`, J01DD13), cefprozil (`CPR`, J01DC10), cefquinome (`CEQ`, QG51AA07), cefroxadine (`CRD`, J01DB11), cefsulodin (`CFS`, J01DD03), ceftaroline (`CPT`, J01DI02), ceftazidime (`CAZ`, J01DD02), ceftazidime/clavulanic acid (`CCV`, J01DD52), cefteram (`CEM`, J01DD18), ceftezole (`CTL`, J01DB12), ceftibuten (`CTB`, J01DD14), ceftiofur (`TIO`, QJ01DD90), ceftizoxime (`CZX`, J01DD07), ceftobiprole medocaril (`CFM1`, J01DI01), ceftolozane/tazobactam (`CZT`, J01DI54), ceftriaxone (`CRO`, J01DD04), ceftriaxone/beta-lactamase inhibitor (`CEB`, J01DD63), cefuroxime (`CXM`, J01DC02), cephradine (`CED`, J01DB09), chloramphenicol (`CHL`, J01BA01), ciprofloxacin (`CIP`, J01MA02), clarithromycin (`CLR`, J01FA09), clindamycin (`CLI`, J01FF01), clometocillin (`CLM`, J01CE07), cloxacillin (`CLO`, J01CF02), colistin (`COL`, J01XB01), cycloserine (`CYC`, J04AB01), dalbavancin (`DAL`, J01XA04), danofloxacin (`DAN`, QJ01MA92), daptomycin (`DAP`, J01XX09), delafloxacin (`DFX`, J01MA23), dibekacin (`DKB`, J01GB09), dicloxacillin (`DIC`, J01CF01), difloxacin (`DIF`, QJ01MA94), dirithromycin (`DIR`, J01FA13),

doripenem (DOR, J01DH04), doxycycline (DOX, J01AA02), enoxacin (ENX, J01MA04), enrofloxacin (ENR, QJ01MA90), epicillin (EPC, J01CA07), ertapenem (ETP, J01DH03), erythromycin (ERY, J01FA01), fleroxacin (FLE, J01MA08), flucloxacillin (FLC, J01CF05), flurithromycin (FLR1, J01FA14), fosfomycin (FOS, J01XX01), framycetin (FRM, D09AA01), fusidic acid (FUS, J01XC01), gamithromycin (GAM, QJ01FA95), garenoxacin (GRN, J01MA19), gatifloxacin (GAT, J01MA16), gemifloxacin (GEM, J01MA15), gentamicin (GEN, J01GB03), grepafloxacin (GRX, J01MA11), hetacillin (HET, J01CA18), imipenem (IPM, J01DH51), imipenem/relebactam (IMR, J01DH56), isepamicin (ISE, J01GB11), josamycin (JOS, J01FA07), kanamycin (KAN, J01GB04), kitasamycin (KIT, QJ01FA93), lascufloxacin (LSC, J01MA25), latamoxef (LTM, J01DD06), levofloxacin (LVX, J01MA12), levonadifloxacin (LND, J01MA24), lincomycin (LIN, J01FF02), linezolid (LNZ, J01XX08), lomefloxacin (LOM, J01MA07), loracarbef (LOR, J01DC08), marbofloxacin (MAR, QJ01MA93), mecillinam (MEC, J01CA11), meropenem (MEM, J01DH02), meropenem/vaborbactam (MEV, J01DH52), metampicillin (MTM, J01CA14), meticillin (MET, J01CF03), mezlocillin (MEZ, J01CA10), micronomicin (MCR, S01AA22), midecamycin (MID, J01FA03), minocycline (MNO, J01AA08), miocamycin (MCM, J01FA11), moxifloxacin (MFX, J01MA14), nadifloxacin (NAD, D10AF05), nafcillin (NAF, J01CF06), nalidixic acid (NAL, J01MB02), neomycin (NEO, J01GB05), netilmicin (NET, J01GB07), nitrofurantoin (NIT, J01XE01), norfloxacin (NOR, J01MA06), novobiocin (NOV, QJ01XX95), ofloxacin (OFX, J01MA01), oleandomycin (OLE, J01FA05), orbifloxacin (ORB, QJ01MA95), oritavancin (ORI, J01XA05), oxacillin (OXA, J01CF04), panipenem (PAN, J01DH55), pazufloxacin (PAZ, J01MA18), pefloxacin (PEF, J01MA03), penamecillin (PNM, J01CE06), pheneticillin (PHE, J01CE05), phenoxymethylpenicillin (PHN, J01CE02), piperacillin (PIP, J01CA12), piperacillin/tazobactam (TZP, J01CR05), pirlimycin (PRL, QJ51FF90), pivampicillin (PVM, J01CA02), pivmecillinam (PME, J01CA08), plazomicin (PLZ, J01GB14), polymyxin B (PLB, J01XB02), pradofloxacin (PRA, QJ01MA97), pristinamycin (PRI, J01FG01), procaine benzylpenicillin (PRB, J01CE09), propicillin (PRP, J01CE03), prulifloxacin (PRU, J01MA17), quinupristin/dalfopristin (QDA, QJ01FG02), ribostamycin (RST, J01GB10), rifampicin (RIF, J04AB02), rokitamycin (ROK, J01FA12), roxithromycin (RXT, J01FA06), rufloxacin (RFL, J01MA10), sarafloxacin (SAR, QJ01MA98), sisomicin (SIS, J01GB08), sitafloxacin (SIT, J01MA21), solithromycin (SOL, J01FA16), sparfloxacin (SPX, J01MA09), spiramycin (SPI, J01FA02), streptoduocin (STR, J01GA02), streptomycin (STR1, J01GA01), sulbactam (SUL, J01CG01), sulbenicillin (SBC, J01CA16), sulfadiazine (SDI, J01EC02), sulfadiazine/trimethoprim (SLT1, J01EE02), sulfadimethoxine (SUD, J01ED01), sulfadimidine (SDM, J01EB03), sulfadimidine/trimethoprim (SLT2, J01EE05), sulfafurazole (SLF, J01EB05), sulfaisodimidine (SLF1, J01EB01), sulfalene (SLF2, J01ED02), sulfamazone (SZO, J01ED09), sulfamerazine (SLF3, J01ED07), sulfamerazine/trimethoprim (SLT3, J01EE07), sulfamethizole (SLF4, J01EB02), sulfamethoxazole (SMX, J01EC01), sulfamethoxypyridazine (SLF5, J01ED05), sulfametomidine (SLF6, J01ED03), sulfametoxydiazine (SLF7, J01ED04), sulfametrole/trimethoprim (SLT4, J01EE03), sulfamoxole (SLF8, J01EC03), sulfamoxole/trimethoprim (SLT5, J01EE04), sulfanilamide (SLF9, J01EB06), sulfaperin (SLF10, J01ED06), sulfaphenazole (SLF11, J01ED08), sulfapyridine (SLF12, J01EB04), sulfathiazole (SUT, J01EB07), sulfathiourea (SLF13, J01EB08), sultamicillin (SLT6, J01CR04), talampicillin (TAL, J01CA15), tazobactam (TAZ, J01CG02), tebipenem (TBP, J01DH06), tedizolid (TZD, J01XX11), teicoplanin (TEC, J01XA02), telavancin (TLV, J01XA03), telithromycin (TLT, J01FA15), temafloxacin (TMX, J01MA05), temocillin (TEM, J01CA17), tetracycline (TCY, J01AA07), ticarcillin (TIC, J01CA13), ticarcillin/clavulanic acid (TCC, J01CR03), tigecycline (TGC, J01AA12), tilbroquinol (TBQ, P01AA05), tildipirosin (TIP, QJ01FA96), tilmicosin (TIL, QJ01FA91), tobramycin (TOB, J01GB01), tosufloxacin (TFX, J01MA22), trimethoprim (TMP, J01EA01), trimethoprim/sulfamethoxazole (SXT, J01EE01), troleandomycin (TRL, J01FA08), trovafloxacin (TVA, J01MA13), tulathromycin (TUL, QJ01FA94), tylosin (TYL, QJ01FA90), tylvalosin (TYL1, QJ01FA92), vancomycin (VAN, J01XA01)

Reference Data Publicly Available

All data sets in this AMR package (about microorganisms, antibiotics, SIR interpretation, EUCAST rules, etc.) are publicly and freely available for download in the following formats: R, MS Excel, Apache Feather, Apache Parquet, SPSS, and Stata. We also provide tab-separated plain text files that are machine-readable and suitable for input in any software program, such as laboratory information systems. Please visit our website for the download links. The actual files are of course available on our GitHub repository.

Source

- EUCAST Expert Rules. Version 2.0, 2012.
  Leclercq et al. **EUCAST expert rules in antimicrobial susceptibility testing.** *Clin Microbiol Infect.* 2013;19(2):141-60; doi:10.1111/j.14690691.2011.03703.x
- EUCAST Expert Rules, Intrinsic Resistance and Exceptional Phenotypes Tables. Version 3.1, 2016. (link)
- EUCAST Intrinsic Resistance and Unusual Phenotypes. Version 3.2, 2020. (link)
- EUCAST Intrinsic Resistance and Unusual Phenotypes. Version 3.3, 2021. (link)
- EUCAST Breakpoint tables for interpretation of MICs and zone diameters. Version 9.0, 2019. (link)
- EUCAST Breakpoint tables for interpretation of MICs and zone diameters. Version 10.0, 2020. (link)
- EUCAST Breakpoint tables for interpretation of MICs and zone diameters. Version 11.0, 2021. (link)
- EUCAST Breakpoint tables for interpretation of MICs and zone diameters. Version 12.0, 2022. (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)

head(b)


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


# Dosage guidelines:

eucast_dosage(c("tobra", "genta", "cipro"), "iv")

eucast_dosage(c("tobra", "genta", "cipro"), "iv", version_breakpoints = 10)
```

`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 antibiotics with class sir (see as.sir()); these column names occur in the antibiotics data set and can be translated with set_ab_names() or ab_name()

Details

Like all data sets in this package, this data set is publicly available for download in the following formats: R, MS Excel, Apache Feather, Apache Parquet, SPSS, and Stata. Please visit our website for the download links. The actual files are of course available on 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 antibiotics that have to be transformed with as.sir()

Details

Like all data sets in this package, this data set is publicly available for download in the following formats: R, MS Excel, Apache Feather, Apache Parquet, SPSS, and Stata. Please visit our website for the download links. The actual files are of course available on 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

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 Hindler *et al.* (2007, doi:10.1086/511864), 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 *(= all isolates)*	`first_isolate(x, method = "isolate-based")`
Patient-based *(= first isolate per patient)*	`first_isolate(x, method = "patient-based")`
Episode-based *(= first isolate per episode)* - 7-Day interval from initial isolate - 30-Day interval from initial isolate	`first_isolate(x, method = "episode-based")`, or: - `first_isolate(x, method = "e", episode_days = 7)` - `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:

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:

To include every genus-species combination per patient once, set the `episode_days` to `Inf`. Although often inappropriate, this method makes sure that no duplicate isolates are selected from the same patient. In a large longitudinal data set, this could mean that isolates are *excluded* that were found years after the initial isolate.

Episode-based:

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 14, 30, 60 or 365. Short episodes are common for analysing specific hospital or ward data, 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:

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

Source

Methodology of this function 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 the chi-squared test statistic.

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

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. http://www.biostathandbook.com/gtestgof.html.

See Also

chisq.test()

Examples

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

x <- c(772, 1611, 737)
g.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

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 = TRUE,
  minimum = 30,
  language = get_AMR_locale(),
  nrow = NULL,
  colours = c(S = "#3CAEA3", 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 = TRUE,
  ...
)

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

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

scale_sir_colours(..., aesthetics = "fill")

theme_sir()

labels_sir_count(
  position = NULL,
  x = "antibiotic",
  translate_ab = "name",
  minimum = 30,
  language = get_AMR_locale(),
  combine_SI = TRUE,
  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 antibiotics data set to translate the antibiotic abbreviations to, using ab_property()
combine_SI	a logical to indicate whether all values of S, SDD, and I must be merged into one, so the output only consists of S+SDD+I vs. R (susceptible vs. resistant) - the default is TRUE
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.
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*.
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

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

The Functions:

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 antibiotics on the x axis.

`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][`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*.

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() +
    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(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 antibiotics 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 antibiotic columns must be detected that were transformed to class `sir` (see `as.sir()`) on beforehand (default is `FALSE`)

Details

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

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)
# NOTE: Using column 'tetr' as input for J01AA07 (tetracycline).

# 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, as.ab("augmentin"))
```

`intrinsic_resistant` *Data Set with Bacterial Intrinsic Resistance*

Description

Data set containing defined intrinsic resistance by EUCAST of all bug-drug combinations.

Usage

```
intrinsic_resistant
```

Format

A tibble with 301 583 observations and 2 variables:

- `mo`
  Microorganism ID
- `ab`
  Antibiotic ID

Details

This data set is based on 'EUCAST Expert Rules' and 'EUCAST Intrinsic Resistance and Unusual Phenotypes' v3.3 (2021).

Direct download:

Like all data sets in this package, this data set is publicly available for download in the following formats: R, MS Excel, Apache Feather, Apache Parquet, SPSS, and Stata. Please visit our website for the download links. The actual files are of course available on our GitHub repository.

They **allow for machine reading EUCAST and CLSI guidelines**, which is almost impossible with the MS Excel and PDF files distributed by EUCAST and CLSI.

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 = FALSE,
 ...
)

all_antimicrobials(x = NULL, only_sir_columns = FALSE, ...)

antimicrobials_equal(
  y,
  z,
  type = c("points", "keyantimicrobials"),
  ignore_I = TRUE,
  points_threshold = 2,
  ...
)
```

Arguments

x	a data.frame with antibiotics 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 columns must be included that were transformed to class sir (see as.sir()) on beforehand (default is FALSE)
...	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 antibiotics 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 unload it:
try(detach("package:data.table", unload = TRUE), 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 and custom guidelines.

Usage

```
mdro(
  x = NULL,
  guideline = "CMI2012",
  col_mo = NULL,
  info = interactive(),
  pct_required_classes = 0.5,
  combine_SI = TRUE,
  verbose = FALSE,
  only_sir_columns = FALSE,
  ...
)

custom_mdro_guideline(..., as_factor = TRUE)

brmo(x = NULL, only_sir_columns = FALSE, ...)

mrgn(x = NULL, only_sir_columns = FALSE, ...)

mdr_tb(x = NULL, only_sir_columns = FALSE, ...)

mdr_cmi2012(x = NULL, only_sir_columns = FALSE, ...)

eucast_exceptional_phenotypes(x = NULL, only_sir_columns = FALSE, ...)
```

Arguments

x	a data.frame with antibiotics 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().
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.
verbose	a logical to turn Verbose mode on and off (default is off). In Verbose mode, the function does not return the MDRO results, but instead returns a data set in logbook form with extensive info about which isolates would be MDRO-positive, or why they are not.
only_sir_columns	a logical to indicate whether only antibiotic columns must be detected that were transformed to class sir (see as.sir()) on beforehand (default is FALSE)
...	in case of custom_mdro_guideline(): a set of rules, see section *Using Custom Guidelines* below. Otherwise: column name of an antibiotic, see section *Antibiotics* below.
as_factor	a logical to indicate whether the returned value should be an ordered factor (TRUE, default), or otherwise a character vector

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.

Value

- CMI 2012 paper - function `mdr_cmi2012()` or `mdro()`:
  Ordered factor with levels `Negative` < `Multi-drug-resistant (MDR)` < `Extensively drug-resistant (XDR)` < `Pandrug-resistant (PDR)`
- TB guideline - function `mdr_tb()` or `mdro(..., guideline = "TB")`:
  Ordered factor with levels `Negative` < `Mono-resistant` < `Poly-resistant` < `Multi-drug-resistant` < `Extensively drug-resistant`
- German guideline - function `mrgn()` or `mdro(..., guideline = "MRGN")`:
  Ordered factor with levels `Negative` < `3MRGN` < `4MRGN`
- Everything else, 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. molecular) tests

Supported International / National Guidelines

Currently supported guidelines are (case-insensitive):

- `guideline = "CMI2012"` (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.14690691.2011.03570.x)
- `guideline = "EUCAST3.3"` (or simply `guideline = "EUCAST"`)

  The European international guideline - EUCAST Expert Rules Version 3.3 "Intrinsic Resistance and Unusual Phenotypes" (link)
- `guideline = "EUCAST3.2"`

  The European international guideline - EUCAST Expert Rules Version 3.2 "Intrinsic Resistance and Unusual Phenotypes" (link)
- `guideline = "EUCAST3.1"`

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

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

  The German national guideline - Mueller et al. (2015) Antimicrobial Resistance and Infection Control 4:7; doi:10.1186/s1375601500476

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

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

Using Custom Guidelines

Custom guidelines can be set with the `custom_mdro_guideline()` function. This is of great 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.

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 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. CIP is "R" and age is higher than 60 -> Elderly Type A
#>   2. ERY is "R" and age is higher than 60 -> Elderly Type B
#>   3. Otherwise -> Negative
#>
#> Unmatched rows will return NA.
```

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

```
x <- mdro(example_isolates,
          guideline = custom)
table(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")))
table(x)
#>       Negative Elderly Type A Elderly Type B Elderly Type C
#>            961            198            732            109
```

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()`.

Antibiotics

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

The following antibiotics are eligible for the functions `eucast_rules()` and `mdro()`. These are shown below in the format 'name (`antimicrobial ID`, ATC code)', sorted alphabetically:

Amikacin (`AMK`, J01GB06), amoxicillin (`AMX`, J01CA04), amoxicillin/clavulanic acid (`AMC`, J01CR02), ampicillin (`AMP`, J01CA01), ampicillin/sulbactam (`SAM`, J01CR01), apramycin (`APR`, QA07AA92), arbekacin (`ARB`, J01GB12), aspoxicillin (`APX`, J01CA19), azidocillin (`AZD`, J01CE04), azithromycin (`AZM`, J01FA10), azlocillin (`AZL`, J01CA09), aztreonam (`ATM`, J01DF01), bacampicillin (`BAM`, J01CA06), bekanamycin (`BEK`, J01GB13), benzathine benzylpenicillin (`BNB`, J01CE08), benzathine phenoxymethylpenicillin (`BNP`, J01CE10), benzylpenicillin (`PEN`, J01CE01), besifloxacin (`BES`, S01AE08), biapenem (`BIA`, J01DH05), carbenicillin (`CRB`, J01CA03), carindacillin (`CRN`, J01CA05), cefacetrile (`CAC`, J01DB10), cefaclor (`CEC`, J01DC04), cefadroxil (`CFR`, J01DB05), cefalexin (`LEX`, J01DB01), cefaloridine (`RID`, J01DB02), cefalotin (`CEP`, J01DB03), cefamandole (`MAN`, J01DC03), cefapirin (`HAP`, J01DB08), cefatrizine (`CTZ`, J01DB07), cefazedone (`CZD`, J01DB06), cefazolin (`CZO`, J01DB04), cefcapene (`CCP`, J01DD17), cefdinir (`CDR`, J01DD15), cefditoren (`DIT`, J01DD16), cefepime (`FEP`, J01DE01), cefetamet (`CAT`, J01DD10), cefiderocol (`FDC`, J01DI04), cefixime (`CFM`, J01DD08), cefmenoxime (`CMX`, J01DD05), cefmetazole (`CMZ`, J01DC09), cefodizime (`DIZ`, J01DD09), cefonicid (`CID`, J01DC06), cefoperazone (`CFP`, J01DD12), cefoperazone/sulbactam (`CSL`, J01DD62), ceforanide (`CND`, J01DC11), cefotaxime (`CTX`, J01DD01), cefotaxime/clavulanic acid (`CTC`, J01DD51), cefotetan (`CTT`, J01DC05), cefotiam (`CTF`, J01DC07), cefovecin (`FOV`, QJ01DD91), cefoxitin (`FOX`, J01DC01), cefozopran (`ZOP`, J01DE03), cefpiramide (`CPM`, J01DD11), cefpirome (`CPO`, J01DE02), cefpodoxime (`CPD`, J01DD13), cefprozil (`CPR`, J01DC10), cefquinome (`CEQ`, QG51AA07), cefroxadine (`CRD`, J01DB11), cefsulodin (`CFS`, J01DD03), ceftaroline (`CPT`, J01DI02), ceftazidime (`CAZ`, J01DD02), ceftazidime/clavulanic acid (`CCV`, J01DD52), cefteram (`CEM`, J01DD18), ceftezole (`CTL`, J01DB12), ceftibuten (`CTB`, J01DD14), ceftiofur (`TIO`, QJ01DD90), ceftizoxime (`CZX`, J01DD07), ceftobiprole medocaril (`CFM1`, J01DI01), ceftolozane/tazobactam (`CZT`, J01DI54), ceftriaxone (`CRO`, J01DD04), ceftriaxone/beta-lactamase inhibitor (`CEB`, J01DD63), cefuroxime (`CXM`, J01DC02), cephradine (`CED`, J01DB09), chloramphenicol (`CHL`, J01BA01), ciprofloxacin (`CIP`, J01MA02), clarithromycin (`CLR`, J01FA09), clindamycin (`CLI`, J01FF01), clometocillin (`CLM`, J01CE07), cloxacillin (`CLO`, J01CF02), colistin (`COL`, J01XB01), cycloserine (`CYC`, J04AB01), dalbavancin (`DAL`, J01XA04), danofloxacin (`DAN`, QJ01MA92), daptomycin (`DAP`, J01XX09), delafloxacin (`DFX`, J01MA23), dibekacin (`DKB`, J01GB09), dicloxacillin (`DIC`, J01CF01), difloxacin (`DIF`, QJ01MA94), dirithromycin (`DIR`, J01FA13), doripenem (`DOR`, J01DH04), doxycycline (`DOX`, J01AA02), enoxacin (`ENX`, J01MA04), enrofloxacin (`ENR`, QJ01MA90), epicillin (`EPC`, J01CA07), ertapenem (`ETP`, J01DH03), erythromycin (`ERY`, J01FA01), fleroxacin (`FLE`, J01MA08), flucloxacillin (`FLC`, J01CF05), flurithromycin (`FLR1`, J01FA14), fosfomycin (`FOS`, J01XX01), framycetin (`FRM`, D09AA01), fusidic acid (`FUS`, J01XC01), gamithromycin (`GAM`, QJ01FA95), garenoxacin (`GRN`, J01MA19), gatifloxacin (`GAT`, J01MA16), gemifloxacin (`GEM`, J01MA15), gentamicin (`GEN`, J01GB03), grepafloxacin (`GRX`, J01MA11), hetacillin (`HET`, J01CA18), imipenem (`IPM`, J01DH51), imipenem/relebactam (`IMR`, J01DH56), isepamicin (`ISE`, J01GB11), josamycin (`JOS`, J01FA07), kanamycin (`KAN`, J01GB04), kitasamycin (`KIT`, QJ01FA93), lascufloxacin (`LSC`, J01MA25), latamoxef (`LTM`, J01DD06), levofloxacin (`LVX`, J01MA12), levonadifloxacin (`LND`,

J01MA24), lincomycin (LIN, J01FF02), linezolid (LNZ, J01XX08), lomefloxacin (LOM, J01MA07), loracarbef (LOR, J01DC08), marbofloxacin (MAR, QJ01MA93), mecillinam (MEC, J01CA11), meropenem (MEM, J01DH02), meropenem/vaborbactam (MEV, J01DH52), metampicillin (MTM, J01CA14), meticillin (MET, J01CF03), mezlocillin (MEZ, J01CA10), micronomicin (MCR, S01AA22), midecamycin (MID, J01FA03), minocycline (MNO, J01AA08), miocamycin (MCM, J01FA11), moxifloxacin (MFX, J01MA14), nadifloxacin (NAD, D10AF05), nafcillin (NAF, J01CF06), nalidixic acid (NAL, J01MB02), neomycin (NEO, J01GB05), netilmicin (NET, J01GB07), nitrofurantoin (NIT, J01XE01), norfloxacin (NOR, J01MA06), novobiocin (NOV, QJ01XX95), ofloxacin (OFX, J01MA01), oleandomycin (OLE, J01FA05), orbifloxacin (ORB, QJ01MA95), oritavancin (ORI, J01XA05), oxacillin (OXA, J01CF04), panipenem (PAN, J01DH55), pazufloxacin (PAZ, J01MA18), pefloxacin (PEF, J01MA03), penamecillin (PNM, J01CE06), pheneticillin (PHE, J01CE05), phenoxymethylpenicillin (PHN, J01CE02), piperacillin (PIP, J01CA12), piperacillin/tazobactam (TZP, J01CR05), pirlimycin (PRL, QJ51FF90), pivampicillin (PVM, J01CA02), pivmecillinam (PME, J01CA08), plazomicin (PLZ, J01GB14), polymyxin B (PLB, J01XB02), pradofloxacin (PRA, QJ01MA97), pristinamycin (PRI, J01FG01), procaine benzylpenicillin (PRB, J01CE09), propicillin (PRP, J01CE03), prulifloxacin (PRU, J01MA17), quinupristin/dalfopristin (QDA, QJ01FG02), ribostamycin (RST, J01GB10), rifampicin (RIF, J04AB02), rokitamycin (ROK, J01FA12), roxithromycin (RXT, J01FA06), rufloxacin (RFL, J01MA10), sarafloxacin (SAR, QJ01MA98), sisomicin (SIS, J01GB08), sitafloxacin (SIT, J01MA21), solithromycin (SOL, J01FA16), sparfloxacin (SPX, J01MA09), spiramycin (SPI, J01FA02), streptoduocin (STR, J01GA02), streptomycin (STR1, J01GA01), sulbactam (SUL, J01CG01), sulbenicillin (SBC, J01CA16), sulfadiazine (SDI, J01EC02), sulfadiazine/trimethoprim (SLT1, J01EE02), sulfadimethoxine (SUD, J01ED01), sulfadimidine (SDM, J01EB03), sulfadimidine/trimethoprim (SLT2, J01EE05), sulfafurazole (SLF, J01EB05), sulfaisodimidine (SLF1, J01EB01), sulfalene (SLF2, J01ED02), sulfamazone (SZO, J01ED09), sulfamerazine (SLF3, J01ED07), sulfamerazine/trimethoprim (SLT3, J01EE07), sulfamethizole (SLF4, J01EB02), sulfamethoxazole (SMX, J01EC01), sulfamethoxypyridazine (SLF5, J01ED05), sulfametomidine (SLF6, J01ED03), sulfametoxydiazine (SLF7, J01ED04), sulfametrole/trimethoprim (SLT4, J01EE03), sulfamoxole (SLF8, J01EC03), sulfamoxole/trimethoprim (SLT5, J01EE04), sulfanilamide (SLF9, J01EB06), sulfaperin (SLF10, J01ED06), sulfaphenazole (SLF11, J01ED08), sulfapyridine (SLF12, J01EB04), sulfathiazole (SUT, J01EB07), sulfathiourea (SLF13, J01EB08), sultamicillin (SLT6, J01CR04), talampicillin (TAL, J01CA15), tazobactam (TAZ, J01CG02), tebipenem (TBP, J01DH06), tedizolid (TZD, J01XX11), teicoplanin (TEC, J01XA02), telavancin (TLV, J01XA03), telithromycin (TLT, J01FA15), temafloxacin (TMX, J01MA05), temocillin (TEM, J01CA17), tetracycline (TCY, J01AA07), ticarcillin (TIC, J01CA13), ticarcillin/clavulanic acid (TCC, J01CR03), tigecycline (TGC, J01AA12), tilbroquinol (TBQ, P01AA05), tildipirosin (TIP, QJ01FA96), tilmicosin (TIL, QJ01FA91), tobramycin (TOB, J01GB01), tosufloxacin (TFX, J01MA22), trimethoprim (TMP, J01EA01), trimethoprim/sulfamethoxazole (SXT, J01EE01), troleandomycin (TRL, J01FA08), trovafloxacin (TVA, J01MA13), tulathromycin (TUL, QJ01FA94), tylosin (TYL, QJ01FA90), tylvalosin (TYL1, QJ01FA92), vancomycin (VAN, J01XA01)

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 as shown below (https://www.eucast.org/newsiandr):

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

- **I - Susceptible, increased exposure**
  A microorganism is categorised as "Susceptible, Increased exposure" when there is a high likelihood of therapeutic success because exposure to the agent is increased by adjusting the dosing regimen or by its concentration at the site of infection.
- **R = Resistant**
  A microorganism is categorised as "Resistant" when there is a high likelihood of therapeutic failure even when there is increased exposure.
  - *Exposure* is a function of how the mode of administration, dose, dosing interval, infusion time, as well as distribution and excretion of the antimicrobial agent will influence the infecting organism at the site of infection.

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

Source

See the supported guidelines above for the list of publications used for this function.

Examples

```
out <- mdro(example_isolates, guideline = "EUCAST")
str(out)
table(out)

out <- mdro(example_isolates,
  guideline = custom_mdro_guideline(
    AMX == "R" ~ "Custom MDRO 1",
    VAN == "R" ~ "Custom MDRO 2"
  )
)
table(out)


if (require("dplyr")) {
  example_isolates %>%
    mdro() %>%
    table()

  # no need to define `x` when used inside dplyr verbs:
  example_isolates %>%
    mutate(MDRO = mdro()) %>%
    pull(MDRO) %>%
    table()
}
```

`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 `column1:column4` and `where(is.mic)`, and can thus also be antibiotic 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$amr_distance <- mean_amr_distance(y, where(is.mic))
y[order(y$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 78 678 Taxonomic Records of Microorganisms*

Description

A data set containing the full microbial taxonomy (**last updated: June 24th, 2024**) of six kingdoms. 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)* for all others taxons.

Usage

```
microorganisms
```

Format

A tibble with 78 678 observations and 26 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", "not validly published", "synonym", or "unknown"
- `kingdom`, `phylum`, `class`, `order`, `family`, `genus`, `species`, `subspecies`
  Taxonomic rank of the microorganism. Note that for fungi, *phylum* is equal to their taxonomic *division*. Also, for fungi, *subkingdom* and *subdivision* were left out since they do not occur in the bacterial taxonomy.
- `rank`
  Text of the taxonomic rank of the microorganism, such as `"species"` or `"genus"`
- `ref`
  Author(s) and year of related scientific publication. This contains only the *first surname* and year of the *latest* authors, e.g. "Wallis *et al.* 2006 *emend.* Smith and Jones 2018" becomes "Smith *et al.*, 2018". This field is directly retrieved from the source specified in the column `source`. Moreover, accents 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", or "microaerophile". 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 68.3% of all ~39 000 bacteria in the data set contain an oxygen tolerance.

- `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 ~33 000 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 ~18 000 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). **This is a unique identifier**, though available for only ~49 000 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 July 16th, 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 ~39 000 (sub)species from the kingdoms of Archaea and Bacteria
- ~28 000 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*).
- ~8 100 (sub)species from the kingdom of Protozoa
- ~1 600 (sub)species from 39 other relevant genera from the kingdom of Animalia (such as *Strongyloides* and *Taenia*)
- All ~22 000 previously accepted names of all included (sub)species (these were taxonomically renamed)
- The complete taxonomic tree of all included (sub)species: from kingdom to subspecies
- The identifier of the parent taxons
- The year and first author of the related scientific publication

Manual additions:

For convenience, some entries were added manually:

- ~1 500 entries of *Salmonella*, such as the city-like serovars and groups A to H
- 36 species groups (such as the beta-haemolytic *Streptococcus* groups A to K, coagulase-negative *Staphylococcus* (CoNS), *Mycobacterium tuberculosis* complex, etc.), of which the group compositions are stored in the microorganisms.groups data set
- 1 entry of *Blastocystis* (*B. hominis*), although it officially does not exist (Noel *et al.* 2005, PMID 15634993)
- 1 entry of *Moraxella* (*M. catarrhalis*), which was formally named *Branhamella catarrhalis* (Catlin, 1970) though this change was never accepted within the field of clinical microbiology
- 8 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.

Direct download:

Like all data sets in this package, this data set is publicly available for download in the following formats: R, MS Excel, Apache Feather, Apache Parquet, SPSS, and Stata. Please visit our website for the download links. The actual files are of course available on our GitHub repository.

Source

Taxonomic entries were imported in this order of importance:

1. List of Prokaryotic names with Standing in Nomenclature (LPSN):

   Parte, AC *et al.* (2020). **List of Prokaryotic names with Standing in Nomenclature (LPSN) moves to the DSMZ.** International Journal of Systematic and Evolutionary Microbiology, 70, 5607-5612; doi:10.1099/ijsem.0.004332. Accessed from `https://lpsn.dsmz.de` on June 24th, 2024.

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 June 24th, 2024.

3. Global Biodiversity Information Facility (GBIF):

   GBIF Secretariat (2023). GBIF Backbone Taxonomy. Checklist dataset doi:10.15468/39omei. Accessed from https://www.gbif.org on June 24th, 2024.

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 July 16th, 2024.

- 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://phinvads.cdc.gov on July 16th, 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 4 971 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 4 971 observations and 2 variables:

- code
  Commonly used code of a microorganism. **This is a unique identifier.**
- mo
  ID of the microorganism in the microorganisms data set

Details

Like all data sets in this package, this data set is publicly available for download in the following formats: R, MS Excel, Apache Feather, Apache Parquet, SPSS, and Stata. Please visit our website for the download links. The actual files are of course available on 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 521 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 521 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()`

Details

Like all data sets in this package, this data set is publicly available for download in the following formats: R, MS Excel, Apache Feather, Apache Parquet, SPSS, and Stata. Please visit our website for the download links. The actual files are of course available on 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 kingdom 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*, *Apophysomyces*, *Arthroderma*, *Aspergillus*, *Aureobasidium*, *Basidiobolus*, *Beauveria*, *Bipolaris*, *Blastobotrys*, *Blastocystis*, *Blastomyces*, *Candida*, *Capillaria*, *Chaetomium*,

Chilomastix, *Chrysonilia*, *Chrysosporium*, *Cladophialophora*, *Cladosporium*, *Clavispora*, *Coccidioides*, *Cokeromyces*, *Conidiobolus*, *Coniochaeta*, *Contracaecum*, *Cordylobia*, *Cryptococcus*, *Cryptosporidium*, *Cunninghamella*, *Curvularia*, *Cyberlindnera*, *Debaryozyma*, *Demodex*, *Dermatobia*, *Dientamoeba*, *Diphyllobothrium*, *Dirofilaria*, *Echinostoma*, *Entamoeba*, *Enterobius*, *Epidermophyton*, *Exidia*, *Exophiala*, *Exserohilum*, *Fasciola*, *Fonsecaea*, *Fusarium*, *Geotrichum*, *Giardia*, *Graphium*, *Haloarcula*, *Halobacterium*, *Halococcus*, *Hansenula*, *Hendersonula*, *Heterophyes*, *Histomonas*, *Histoplasma*, *Hortaea*, *Hymenolepis*, *Hypomyces*, *Hysterothylacium*, *Kloeckera*, *Kluyveromyces*, *Kodamaea*, *Lacazia*, *Leishmania*, *Lichtheimia*, *Lodderomyces*, *Lomentospora*, *Madurella*, *Malassezia*, *Malbranchea*, *Metagonimus*, *Meyerozyma*, *Microsporidium*, *Microsporum*, *Millerozyma*, *Mortierella*, *Mucor*, *Mycocentrospora*, *Nannizzia*, *Necator*, *Nectria*, *Ochroconis*, *Oesophagostomum*, *Oidiodendron*, *Opisthorchis*, *Paecilomyces*, *Paracoccidioides*, *Pediculus*, *Penicillium*, *Phaeoacremonium*, *Phaeomoniella*, *Phialophora*, *Phlebotomus*, *Phoma*, *Pichia*, *Piedraia*, *Pithomyces*, *Pityrosporum*, *Pneumocystis*, *Pseudallescheria*, *Pseudoscopulariopsis*, *Pseudoterranova*, *Pulex*, *Purpureocillium*, *Quambalaria*, *Rhinocladiella*, *Rhizomucor*, *Rhizopus*, *Rhodotorula*, *Saccharomyces*, *Saksenaea*, *Saprochaete*, *Sarcoptes*, *Scedosporium*, *Schistosoma*, *Schizosaccharomyces*, *Scolecobasidium*, *Scopulariopsis*, *Scytalidium*, *Spirometra*, *Sporobolomyces*, *Sporopachydermia*, *Sporothrix*, *Sporotrichum*, *Stachybotrys*, *Strongyloides*, *Syncephalastrum*, *Syngamus*, *Taenia*, *Talaromyces*, *Teleomorph*, *Toxocara*, *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.

Reference Data Publicly Available

All data sets in this `AMR` package (about microorganisms, antibiotics, SIR interpretation, EUCAST rules, etc.) are publicly and freely available for download in the following formats: R, MS Excel, Apache Feather, Apache Parquet, SPSS, and Stata. We also provide tab-separated plain text files that are machine-readable and suitable for input in any software program, such as laboratory information systems. Please visit our website for the download links. The actual files are of course available on 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),
  ...
)

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_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 old, previously valid taxonomic names must be preserved

	and not be corrected to currently accepted names. The default is FALSE, which will return a note if old 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'
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", "kingdom", "phylum", "class", "order", "family", "genus", "species", "subspecies", "rank", "ref", "oxygen_tolerance", "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)

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* (kingdom of Bacteria) as well as *Entamoeba coli* (kingdom 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 be considered *Escherichia coli*. As a result, mo_fullname(mo_shortname("Entamoeba coli")) returns "Escherichia coli".

Since the top-level of the taxonomy is sometimes referred to as 'kingdom' and sometimes as 'domain', the functions mo_kingdom() and mo_domain() return the exact same 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()) will be based on the taxonomic kingdom and phylum. Originally, Cavalier-Smith defined the so-called subkingdoms Negibacteria and Posibacteria (2002, PMID 11837318), and only considered these phyla as Posibacteria: Actinobacteria,

Chloroflexi, Firmicutes, and Tenericutes. These phyla were later renamed to Actinomycetota, Chloroflexota, Bacillota, and Mycoplasmatota (2021, PMID 34694987). 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()`) will be based on the taxonomic kingdom 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()`) will be based on the intrinsic_resistant data set, which is based on 'EUCAST Expert Rules' and 'EUCAST Intrinsic Resistance and Unusual Phenotypes' v3.3 (2021). The `mo_is_intrinsic_resistant()` function can be vectorised over both argument x (input for microorganisms) and ab (input for antibiotics).

Determination of bacterial oxygen tolerance (`mo_oxygen_tolerance()`) will be based on BacDive, see *Source*. The function `mo_is_anaerobic()` only returns TRUE if the oxygen tolerance is `"anaerobe"`, indicting an obligate anaerobic species or genus. It always returns FALSE for species outside the taxonomic kingdom 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 will be 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 July 16th, 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
- Parte, AC *et al.* (2020). **List of Prokaryotic names with Standing in Nomenclature (LPSN) moves to the DSMZ.** International Journal of Systematic and Evolutionary Microbiology, 70, 5607-5612; doi:10.1099/ijsem.0.004332. Accessed from `https://lpsn.dsmz.de` on June 24th, 2024.
- 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 June 24th, 2024.
- GBIF Secretariat (2023). GBIF Backbone Taxonomy. Checklist dataset doi:10.15468/39omei. Accessed from `https://www.gbif.org` on June 24th, 2024.
- 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 July 16th, 2024.
- 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://phinvads.cdc.gov`
- Bartlett A *et al.* (2022). **A comprehensive list of bacterial pathogens infecting humans** *Microbiology* 168:001269; doi:10.1099/mic.0.001269

Reference Data Publicly Available

All data sets in this AMR package (about microorganisms, antibiotics, SIR interpretation, EUCAST rules, etc.) are publicly and freely available for download in the following formats: R, MS Excel, Apache Feather, Apache Parquet, SPSS, and Stata. We also provide tab-separated plain text files that are machine-readable and suitable for input in any software program, such as laboratory information systems. Please visit our website for the download links. The actual files are of course available on our GitHub repository.

See Also

Data set microorganisms

Examples

```
# taxonomic tree ---------------------------------------------------------

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")


# full names and short names ----------------------------------------------

mo_name("Klebsiella pneumoniae")
mo_fullname("Klebsiella pneumoniae")
mo_shortname("Klebsiella pneumoniae")


# other properties --------------------------------------------------------

mo_pathogenicity("Klebsiella pneumoniae")
mo_gramstain("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(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_kingdom, but mo_kingdom will remain untranslated
mo_kingdom("Klebsiella pneumoniae")
mo_type("Klebsiella pneumoniae")
mo_kingdom("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 `"home/me/ourcodes.xlsx"`. Now we have to set it as a source:

```
set_mo_source("home/me/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("ggplot2")) {
    ggplot_pca(pca_result)

    ggplot_pca(pca_result) +
      scale_colour_viridis_d() +
      labs(title = "Title here")
  }
}
```

---

plot *Plotting for Classes* sir, mic *and* disk

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 allows custom MIC ranges and to plot intermediate log2 levels for missing MIC values.

Usage

```
scale_x_mic(keep_operators = "edges", mic_range = NULL, drop = FALSE, ...)

scale_y_mic(keep_operators = "edges", mic_range = NULL, drop = FALSE, ...)
```

```
scale_colour_mic(keep_operators = "edges", mic_range = NULL, drop = FALSE, ...)

scale_fill_mic(keep_operators = "edges", mic_range = NULL, drop = FALSE, ...)

## S3 method for class 'mic'
plot(
  x,
  mo = NULL,
  ab = NULL,
  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("#3CAEA3", "#F6D55C", "#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 = "EUCAST",
  title = deparse(substitute(object)),
  ylab = translate_AMR("Frequency", language = language),
  xlab = translate_AMR("Minimum Inhibitory Concentration (mg/L)", language = language),
  colours_SIR = c("#3CAEA3", "#F6D55C", "#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'
fortify(object, ...)

## 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 = "EUCAST",
  colours_SIR = c("#3CAEA3", "#F6D55C", "#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 = "EUCAST",
  colours_SIR = c("#3CAEA3", "#F6D55C", "#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'
fortify(object, ...)

## 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("#3CAEA3", "#F6D55C", "#ED553B"),
```

```
  language = get_AMR_locale(),
  ...
)

## S3 method for class 'sir'
fortify(object, ...)
```

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 limit the MIC values, e.g., mic_range = c(0.001, 32). Use NA to set no limit on one side, e.g., mic_range = c(NA, 32).
drop	a logical to remove intermediate MIC values, defaults to FALSE
...	arguments passed on to methods
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
colours_SIR	colours to use for filling in the bars, must be a vector of three values (in the order S, I and R). 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 or language = "" to prevent translation.
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".

Details

The interpretation of "I" will be named "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, supported guidelines to be used as input for the `guideline` argument are: "EUCAST 2024", "EUCAST 2023", "EUCAST 2022", "EUCAST 2021", "EUCAST 2020", "EUCAST 2019", "EUCAST 2018", "EUCAST 2017", "EUCAST 2016", "EUCAST 2015", "EUCAST 2014", "EUCAST 2013", "EUCAST 2012", "EUCAST 2011", "CLSI 2024", "CLSI 2023", "CLSI 2022", "CLSI 2021", "CLSI 2020", "CLSI 2019", "CLSI 2018", "CLSI 2017", "CLSI 2016", "CLSI 2015", "CLSI 2014", "CLSI 2013", "CLSI 2012", and "CLSI 2011".

Simply using `"CLSI"` or `"EUCAST"` as input will automatically select the latest version of that guideline.

Value

The `autoplot()` functions return a `ggplot` model that is extendible with any `ggplot2` function.

The `fortify()` functions return a data.frame as an extension for usage in the `ggplot2::ggplot()` 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))

plot(some_mic_values)
plot(some_disk_values)
plot(some_sir_values)

# when providing the microorganism and antibiotic, colours will show interpretations:
plot(some_mic_values, mo = "S. aureus", ab = "ampicillin")
plot(some_disk_values, mo = "Escherichia coli", ab = "cipro")
plot(some_disk_values, mo = "Escherichia coli", ab = "cipro", language = "nl")


# 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 = "without scale_x_mic()")
}
if (require("ggplot2")) {
  mic_plot +
    scale_x_mic() +
    labs(title = "with scale_x_mic()")
}
```

```
if (require("ggplot2")) {
  mic_plot +
    scale_x_mic(keep_operators = "all") +
    labs(title = "with scale_x_mic() keeping all 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")
}

if (require("ggplot2")) {
  autoplot(some_mic_values)
}
if (require("ggplot2")) {
  autoplot(some_disk_values, mo = "Escherichia coli", ab = "cipro")
}
if (require("ggplot2")) {
  autoplot(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)

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

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 = TRUE,
  confidence_level = 0.95
)

sir_df(
  data,
  translate_ab = "name",
  language = get_AMR_locale(),
  minimum = 30,
  as_percent = FALSE,
  combine_SI = TRUE,
  confidence_level = 0.95
)
```

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 antibiotics, see section *Combination Therapy* 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 antibiotics 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`
: a logical to indicate whether all values of S, SDD, and I must be merged into one, so the output only consists of S+SDD+I vs. R (susceptible vs. resistant) - the default is `TRUE`

Details

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.

Combination Therapy

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

		only_all_tested = FALSE		only_all_tested = TRUE	
Drug A	Drug B	include as numerator	include as denominator	include as numerator	include as denominator
S or I	S or I	X	X	X	X
R	S or I	X	X	X	X
<NA>	S or I	X	X	-	-
S or I	R	X	X	X	X
R	R	-	X	-	X
<NA>	R	-	-	-	-
S or I	<NA>	X	X	-	-
R	<NA>	-	-	-	-
<NA>	<NA>	-	-	-	-

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

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

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

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

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

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 as shown below (https://www.eucast.org/newsiandr):

- **S - Susceptible, standard dosing regimen**
  A microorganism is categorised as "Susceptible, standard dosing regimen", when there is a high likelihood of therapeutic success using a standard dosing regimen of the agent.
- **I - Susceptible, increased exposure**
  A microorganism is categorised as "Susceptible, Increased exposure" when there is a high likelihood of therapeutic success because exposure to the agent is increased by adjusting the dosing regimen or by its concentration at the site of infection.
- **R = Resistant**
  A microorganism is categorised as "Resistant" when there is a high likelihood of therapeutic failure even when there is increased exposure.
  - *Exposure* is a function of how the mode of administration, dose, dosing interval, infusion time, as well as distribution and excretion of the antimicrobial agent will influence the infecting organism at the site of infection.

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

Source

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 antibiotic 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, ...)

random_disk(size = NULL, mo = NULL, ab = NULL, ...)

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()
ab	any character that can be coerced to a valid antimicrobial drug code with as.ab()
...	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

The base R function sample() is used for generating values.

Generated values are based on the EUCAST 2024 guideline as implemented in the clinical_breakpoints data set. To create specific generated values per bug or drug, set the mo and/or ab argument.

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)


# make the random generation more realistic by setting a bug and/or drug:
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 on statistical solid ground. Standard errors (SE) will be returned as columns `se_min` and `se_max`. See *Examples* for a real live example.

Usage

```
resistance_predict(
  x,
  col_ab,
  col_date = NULL,
  year_min = NULL,
  year_max = NULL,
  year_every = 1,
  minimum = 30,
  model = 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 as shown below (https://www.eucast.org/newsiandr):

- **S - Susceptible, standard dosing regimen**
  A microorganism is categorised as "Susceptible, standard dosing regimen", when there is a high likelihood of therapeutic success using a standard dosing regimen of the agent.
- **I - Susceptible, increased exposure**
  A microorganism is categorised as "Susceptible, Increased exposure" when there is a high likelihood of therapeutic success because exposure to the agent is increased by adjusting the dosing regimen or by its concentration at the site of infection.
- **R = Resistant**
  A microorganism is categorised as "Resistant" when there is a high likelihood of therapeutic failure even when there is increased exposure.
  - *Exposure* is a function of how the mode of administration, dose, dosing interval, infusion time, as well as distribution and excretion of the antimicrobial agent will influence the infecting organism at the site of infection.

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

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)
}
```

---

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))
```

---

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), Chinese (zh), Czech (cs), Danish (da), Dutch (nl), Finnish (fi), French (fr), German (de), Greek (el), Italian (it), Japanese (ja), Norwegian (no), Polish (pl), Portuguese (pt), Romanian (ro), Russian (ru), Spanish (es), Swedish (sv), Turkish (tr), or Ukrainian (uk).
x	text to translate

Details

The currently 20 supported languages are English (en), Chinese (zh), Czech (cs), Danish (da), Dutch (nl), Finnish (fi), French (fr), German (de), Greek (el), Italian (it), Japanese (ja), Norwegian (no), Polish (pl), Portuguese (pt), Romanian (ro), Russian (ru), Spanish (es), Swedish (sv), Turkish (tr), and Ukrainian (uk). 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 (as returned by `Sys.getenv("LANG")` or, if `LANG` is not set, `Sys.getlocale("LC_COLLATE")`), if that language is supported. But the language to be used can be overwritten in two ways and will be checked in this order:

1. Setting the package option `AMR_locale`, either 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. Setting the system variable LANGUAGE or LANG, e.g. by adding LANGUAGE="de_DE.utf8" to your .Renviron file in your home directory.

Thus, if the package option AMR_locale is set, the system variables LANGUAGE and LANG 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/reg_hum_atc.htm).

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

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

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 antibiotic 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 antibiotics. You can lookup the abbreviations in the antibiotics data set, or use e.g. `ab_name("AMP")` to get the official name immediately. Before analysis, you should transform this to a valid antibiotic class, using `as.sir()`.

Details

Like all data sets in this package, this data set is publicly available for download in the following formats: R, MS Excel, Apache Feather, Apache Parquet, SPSS, and Stata. Please visit our website for the download links. The actual files are of course available on our GitHub repository.

Examples

```
WHONET
```

Index