Package 'ir'

October 13, 2022

Title Functions to Handle and Preprocess Infrared Spectra

Version 0.2.1

Description Functions to import and handle infrared spectra (import from '.csv' and Thermo Galactic's '.spc', baseline correction, binning, clipping, interpolating, smoothing, averaging, adding, subtracting, dividing, multiplying, plotting).

License GPL-3

Encoding UTF-8

LazyData true

RoxygenNote 7.1.2.9000

Depends R (>= 4.0.0)

Imports tidyr, dplyr, purrr, tibble, ggplot2, stringr, ChemoSpec (>= 5.2.12), hyperSpec (>= 0.99.20200527), signal, grDevices, rlang, methods, units, Rdpack, magrittr, stats, lifecycle, baseline

Suggests kableExtra, fda, knitr, rmarkdown, spelling, vctrs, tidyselect

VignetteBuilder knitr

RdMacros Rdpack

Date 2022-05-02

URL https://henningte.github.io/ir/

Language en-US

NeedsCompilation no

Author Henning Teickner [aut, cre, cph] (<https://orcid.org/0000-0002-3993-1182>)

Maintainer Henning Teickner <henning.teickner@uni-muenster.de>

Repository CRAN

Date/Publication 2022-05-02 11:50:08 UTC

R topics documented:

`arrange.ir` *Arrange rows in* `ir` *objects by column values*

Description

Arrange rows in `ir` objects by column values

Usage

```
arrange.ir(.data, ..., .by_group = FALSE)
```

Arguments

`.data`	An object of class `ir`.
`...`	<data-masking> Variables, or functions of variables. Use `desc()` to sort a variable in descending order.
`.by_group`	If `TRUE`, will sort first by grouping variable. Applies to grouped data frames only.

Value

`.data` with arranged rows.

Source

`dplyr::arrange()`

See Also

Other tidyverse: `distinct.ir()`, `extract.ir()`, `filter-joins`, `filter.ir()`, `group_by`, `mutate-joins`, `mutate`, `nest`, `pivot_longer.ir()`, `pivot_wider.ir()`, `rename`, `rowwise.ir()`, `select.ir()`, `separate.ir()`, `separate_rows.ir()`, `slice`, `summarize`, `unite.ir()`

Examples

```
## arrange
dplyr::arrange(ir_sample_data, dplyr::desc(sample_type))
```

`bind` *Bind rows of* `ir` *objects*

Description

Bind rows of `ir` objects

Usage

```
## S3 method for class 'ir'
rbind(..., deparse.level = 1)

## S3 method for class 'ir'
cbind(..., deparse.level = 1)
```

Arguments

`...`	Objects to bind together. For `cbind`, only the first of the objects is allowed to be of class `ir`.
`deparse.level`	An integer value; see `rbind()`.

Value

An object of class `ir`. `rbind` returns all input `ir` objects combined row-wise. `cbind` returns the input `ir` object and the other objects combined column-wise.

Examples

```
# rbind
rbind(ir_sample_data, ir_sample_data)
rbind(ir_sample_data %>% dplyr::select(spectra),
      ir_sample_data %>% dplyr::select(spectra))

# cbind
cbind(ir_sample_data, a = seq_len(nrow(ir_sample_data)))
```

distinct.ir *Subset distinct/unique rows in* ir *objects*

Description

Subset distinct/unique rows in ir objects

Usage

```
distinct.ir(.data, ..., .keep_all = FALSE)
```

Arguments

.data	An object of class ir.
...	<data-masking> Optional variables to use when determining uniqueness. If there are multiple rows for a given combination of inputs, only the first row will be preserved. If omitted, will use all variables.
.keep_all	If TRUE, keep all variables in .data. If a combination of ... is not distinct, this keeps the first row of values.

Value

.data with distinct rows.

Source

dplyr::distinct()

See Also

Other tidyverse: arrange.ir(), extract.ir(), filter-joins, filter.ir(), group_by, mutate-joins, mutate, nest, pivot_longer.ir(), pivot_wider.ir(), rename, rowwise.ir(), select.ir(), separate.ir(), separate_rows.ir(), slice, summarize, unite.ir()

Examples

```
## distinct
dplyr::distinct(rep(ir_sample_data, 2))
```

`extract.ir` *Extract a character column in an* `ir` *object into multiple columns using regular expression groups*

Description

Extract a character column in an `ir` object into multiple columns using regular expression groups

Usage

```
extract.ir(
  data,
  col,
  into,
  regex = "([[:alnum:]]+)",
  remove = TRUE,
  convert = FALSE,
  ...
)
```

Arguments

`data`	An object of class `ir`.
`col`	Column name or position. This is passed to `tidyselect::vars_pull()`. This argument is passed by expression and supports quasiquotation (you can unquote column names or column positions).
`into`	Names of new variables to create as character vector. Use `NA` to omit the variable in the output.
`regex`	A string representing a regular expression used to extract the desired values. There should be one group (defined by `()`) for each element of `into`.
`remove`	If `TRUE`, remove input column from output data frame.
`convert`	If `TRUE`, will run `type.convert()` with `as.is = TRUE` on new columns. This is useful if the component columns are integer, numeric or logical. NB: this will cause string `"NA"`s to be converted to `NA`s.
`...`	Additional arguments passed on to methods.

Value

`data` with an extracted character column. See `tidyr::extract()`.

Source

`tidyr::extract()`

See Also

Other tidyverse: arrange.ir(), distinct.ir(), filter-joins, filter.ir(), group_by, mutate-joins, mutate, nest, pivot_longer.ir(), pivot_wider.ir(), rename, rowwise.ir(), select.ir(), separate.ir(), separate_rows.ir(), slice, summarize, unite.ir()

Examples

```
## extract
ir_sample_data %>%
  tidyr::extract(
    id_sample,  "a"
  )
```

---

filter-joins *Filtering joins for an* ir *object*

Description

Filtering joins for an ir object

Usage

```
semi_join.ir(x, y, by = NULL, copy = FALSE, ..., na_matches = c("na", "never"))

anti_join.ir(x, y, by = NULL, copy = FALSE, ..., na_matches = c("na", "never"))
```

Arguments

x	An object of class ir.
y	A data frame.
by	A character vector of variables to join by. If NULL, the default, *_join() will perform a natural join, using all variables in common across x and y. A message lists the variables so that you can check they're correct; suppress the message by supplying by explicitly. To join by different variables on x and y, use a named vector. For example, by = c("a" = "b") will match x$a to y$b. To join by multiple variables, use a vector with length > 1. For example, by = c("a", "b") will match x$a to y$a and x$b to y$b. Use a named vector to match different variables in x and y. For example, by = c("a" = "b", "c" = "d") will match x$a to y$b and x$c to y$d. To perform a cross-join, generating all combinations of x and y, use by = character().
copy	If x and y are not from the same data source, and copy is TRUE, then y will be copied into the same src as x. This allows you to join tables across srcs, but it is a potentially expensive operation so you must opt into it.

...	Other parameters passed onto methods.
na_matches	Should NA and NaN values match one another? The default, "na", treats two NA or NaN values as equal, like %in%, match(), merge(). Use "never" to always treat two NA or NaN values as different, like joins for database sources, similarly to merge(incomparables = FALSE).

Value

x and y joined. If the spectra column is renamed, the ir class is dropped. See filter-joins.

Source

filter-joins

See Also

Other tidyverse: arrange.ir(), distinct.ir(), extract.ir(), filter.ir(), group_by, mutate-joins, mutate, nest, pivot_longer.ir(), pivot_wider.ir(), rename, rowwise.ir(), select.ir(), separate.ir(), separate_rows.ir(), slice, summarize, unite.ir()

Examples

```
## semi_join
set.seed(234)
dplyr::semi_join(
  ir_sample_data,
  tibble::tibble(
    id_measurement = c(1:5, 101:105),
    nitrogen_content = rbeta(n = 10, 0.2, 0.1)
  ),
  by = "id_measurement"
)


## anti_join
set.seed(234)
dplyr::anti_join(
  ir_sample_data,
  tibble::tibble(
    id_measurement = c(1:5, 101:105),
    nitrogen_content = rbeta(n = 10, 0.2, 0.1)
  ),
  by = "id_measurement"
)
```

filter.ir *Subset rows in* ir *objects using column values*

Description

Subset rows in ir objects using column values

Usage

```
filter.ir(.data, ..., .preserve = FALSE)
```

Arguments

.data	An object of class ir.
...	<data-masking> Expressions that return a logical value, and are defined in terms of the variables in .data. If multiple expressions are included, they are combined with the & operator. Only rows for which all conditions evaluate to TRUE are kept.
.preserve	Relevant when the .data input is grouped. If .preserve = FALSE (the default), the grouping structure is recalculated based on the resulting data, otherwise the grouping is kept as is.

Value

.data with filtered rows.

Source

dplyr::filter()

See Also

Other tidyverse: arrange.ir(), distinct.ir(), extract.ir(), filter-joins, group_by, mutate-joins, mutate, nest, pivot_longer.ir(), pivot_wider.ir(), rename, rowwise.ir(), select.ir(), separate.ir(), separate_rows.ir(), slice, summarize, unite.ir()

Examples

```
## filter
dplyr::filter(ir_sample_data, sample_type == "office paper")
```

group_by *Group rows in* ir *objects by one or more variables*

Description

Group rows in ir objects by one or more variables

Usage

```
group_by.ir(
  .data,
  ...,
  .add = FALSE,
  .drop = dplyr::group_by_drop_default(.data)
)

ungroup.ir(.data, ...)
```

Arguments

.data	An object of class ir.
...	In group_by(), variables or computations to group by. Computations are always done on the ungrouped data frame. To perform computations on the grouped data, you need to use a separate mutate() step before the group_by(). Computations are not allowed in nest_by(). In ungroup(), variables to remove from the grouping.
.add	When FALSE, the default, group_by() will override existing groups. To add to the existing groups, use .add = TRUE. This argument was previously called add, but that prevented creating a new grouping variable called add, and conflicts with our naming conventions.
.drop	Drop groups formed by factor levels that don't appear in the data? The default is TRUE except when .data has been previously grouped with .drop = FALSE. See group_by_drop_default() for details.

Value

.data with grouped rows (group_by.ir()) or ungrouped rows (ungroup.ir()).

Source

dplyr::group_by()

See Also

Other tidyverse: arrange.ir(), distinct.ir(), extract.ir(), filter-joins, filter.ir(), mutate-joins, mutate, nest, pivot_longer.ir(), pivot_wider.ir(), rename, rowwise.ir(), select.ir(), separate.ir(), separate_rows.ir(), slice, summarize, unite.ir()

Examples

```
## group_by
dplyr::group_by(ir_sample_data, sample_type)


## ungroup
dplyr::ungroup(dplyr::group_by(ir_sample_data, sample_type))
```

`ir_add` *Add infrared spectra*

Description

`ir_add` takes two objects of class `ir`, `x` and `y`, and adds the intensity values of spectra in matching rows from `y` to that of `x`.

Usage

```
ir_add(x, y)
```

Arguments

`x`	An object of class `ir`.
`y`	An object of class `ir` or a numeic value. If `y` is an object of class `ir`, it must have the same number of rows as `x` and the same x axis values (e.g. wavenumber values) in each matching spectrum as in `x`.

Value

`x` where for each spectrum the respective intensity values in `y` are added.

Examples

```
x1 <-
  ir::ir_add(ir::ir_sample_data, ir::ir_sample_data)
x1 <-
  ir::ir_add(ir::ir_sample_data, ir::ir_sample_data[1, ])
```

ir_as_ir *Generic to convert objects to class* ir

Description

ir_as_ir ir the generic to convert an object to an object of class ir.

Usage

```
ir_as_ir(x, ...)

## S3 method for class 'ir'
ir_as_ir(x, ...)

## S3 method for class 'data.frame'
ir_as_ir(x, ...)
```

Arguments

x	An object.
...	Further arguments passed to individual methods. • If x is a data frame or an object of class ir, these are ignored.

Value

An object of class ir.

Examples

```
# conversion from an ir object
ir::ir_sample_data %>%
  ir_as_ir()

# conversion from a data frame
x_ir <- ir::ir_sample_data

x_df <-
  x_ir %>%
  ir_drop_spectra() %>%
  dplyr::mutate(
    spectra = x_ir$spectra
  ) %>%
  ir_as_ir()

# check that ir_as_ir preserves the input class
ir_sample_data %>%
  structure(class = setdiff(class(.), "ir")) %>%
  dplyr::group_by(sample_type) %>%
```

```
ir_as_ir()
```

`ir_average` *Averages infrared spectra within groups*

Description

`ir_average` averages infrared spectra within a user-defined group. `NA` values are omitted by default.

Usage

```
ir_average(x, ..., na.rm = TRUE)
```

Arguments

`x`	An object of class `ir`.
`...`	Variables in `x` to use as groups.
`na.rm`	A logical value indicating if `NA` values should be dropped (`TRUE`) or not (`FALSE`).

Value

An object of class `ir` where spectra have been averaged within groups defined by `...`.

Examples

```
# average the sample data spectra across sample types
x <-
  ir::ir_sample_data %>%
  ir::ir_average(sample_type)
```

`ir_bc` *Performs baseline correction on infrared spectra*

Description

`ir_bc` performs baseline correction for infrared spectra. Baseline correction is either performed by using a polynomial with user defined degree fitted to each spectrum (see `ChemoSpec::baselineSpectra()`), or by using a rubberband function that is fitted to each spectrum (see `hyperSpec::spc.rubberband()`), or using a Savitzky-Golay smoothed version of the input spectra (see `ir_bc_sg()`).

Usage

```
ir_bc(x, method = "rubberband", ..., return_bl = FALSE)
```

Arguments

x	An object of class ir.
method	A character value indicating which method should be used for baseline correction. If method = "polynomial", a polynomial is used for baseline correction. If method = "rubberband", a rubberband function is used for baseline correction. If method = "sg", a Savitzky-Golay smoothed version of the input spectra is used for baseline correction.
...	Further arguments passed to ir_bc_polynomial() or ir_bc_sg().
return_bl	A logical value indicating if for each spectrum the baseline should be returned instead of the corrected intensity values (return_bl = TRUE) or not (return_bl = FALSE).

Value

An object of class ir with the baseline corrected spectra, or if return_bl = TRUE, the baselines instead of the spectra in column spectra.

Examples

```
library(dplyr)

# rubberband baseline correction
x1 <-
   ir::ir_sample_data %>%
   dplyr::slice(1:10) %>%
   ir::ir_bc(method = "rubberband")

# polynomial baseline correction
x2 <-
   ir::ir_sample_data %>%
   dplyr::slice(1:10) %>%
   ir::ir_bc(method = "polynomial", degree = 2)

# Savitzky-Golay baseline correction
x3 <-
   ir::ir_sample_data %>%
   dplyr::slice(1:10) %>%
   ir::ir_bc(method = "sg", p = 3, n = 199, ts = 1, m = 0)

# return the baseline instead of the baseline corrected spectra
x1_bl <-
   ir::ir_sample_data %>%
   dplyr::slice(1:10) %>%
   ir::ir_bc(method = "rubberband", return_bl = TRUE)
```

`ir_bc_polynomial` *Performs baseline correction on infrared spectra using a polynomial*

Description

`ir_bc_polynomial` performs baseline correction for infrared spectra using a polynomial. `ir_bc_polynomial` is an extended wrapper function for `ChemoSpec::baselineSpectra()`.

Usage

```
ir_bc_polynomial(x, degree = 2, return_bl = FALSE)
```

Arguments

`x`	An object of class `ir`.
`degree`	An integer value representing the degree of the polynomial used for baseline correction.
`return_bl`	A logical value indicating if for each spectrum the baseline should be returned instead of the corrected intensity values (`return_bl = TRUE`) or not (`return_bl = FALSE`).

Value

An object of class `ir` with the baseline corrected spectra if `returnbl = FALSE` or the baselines if `returnbl = TRUE`.

See Also

`ir_bc()`

Examples

```
x2 <-
   ir::ir_sample_data %>%
   ir::ir_bc_polynomial(degree = 2, return_bl = FALSE)
```

ir_bc_rubberband — *Performs baseline correction on infrared spectra using a rubberband algorithm*

Description

ir_bc_rubberband performs baseline correction for infrared spectra using a rubberband algorithm. ir_bc_rubberband is an extended wrapper function for hyperSpec::spc.rubberband().

Usage

```
ir_bc_rubberband(x, return_bl = FALSE)
```

Arguments

x	An object of class ir.
return_bl	A logical value indicating if for each spectrum the baseline should be returned instead of the corrected intensity values (return_bl = TRUE) or not (return_bl = FALSE).

Value

An object of class ir with the baseline corrected spectra and, if returnbl = TRUE, the baselines.

See Also

ir_bc()

Examples

```
x1 <-
   ir::ir_sample_data %>%
   ir::ir_bc_rubberband(return_bl = FALSE)
```

ir_bc_sg — *Performs baseline correction on infrared spectra using a Savitzky-Golay baseline*

Description

ir_bc_sg computes a smoothed version of spectra using ir_smooth() with method = "sg" and uses this as baseline which is subtracted from the spectra to perform a baseline correction (Lasch 2012).

Usage

```
ir_bc_sg(x, ..., return_bl = FALSE)
```

Arguments

x	An object of class ir.
...	Arguments passed to ir_smooth() (except for method which is always set to "sg").
return_bl	A logical value indicating if for each spectrum the baseline should be returned instead of the corrected intensity values (return_bl = TRUE) or not (return_bl = FALSE).

Value

An object of class ir with the baseline corrected spectra and, if returnbl = TRUE, the baselines.

References

Lasch P (2012). "Spectral Pre-Processing for Biomedical Vibrational Spectroscopy and Microspectroscopic Imaging." *Chemometrics and Intelligent Laboratory Systems*, **117**, 100–114. ISSN 01697439, doi:10.1016/j.chemolab.2012.03.011.

Examples

```
x <-
   ir::ir_sample_data %>%
   ir::ir_bc_sg(p = 3, n = 199, ts = 1, m = 0, return_bl = FALSE)
```

---

ir_bin *Bins infrared spectra*

Description

ir_bin bins intensity values of infrared spectra into bins of a defined width or into a defined number of bins.

Usage

```
ir_bin(x, width = 10)
```

Arguments

x	An object of class ir with integer wavenumber values increasing by 1.
width	An integer value indicating the wavenumber width of each resulting bin. Must be set to NULL if n is specified.

Details

If the last bin contains fewer input values than the remaining bins, it will be dropped and a warning will be printed. If a wavenumber value exactly matches the boundary of a bin window, the respective intensity value will be assigned to both neighboring bins.

Value

An object of class `ir` where spectra have been binned.

Examples

```
x <-
   ir::ir_sample_data %>%
   ir_bin(width = 50)
```

---

ir_clip *Clips infrared spectra to new wavenumber ranges*

Description

`ir_clip` clips infrared spectra to a new, specified, wavenumber range or multiple new specified wavenumber ranges.

Usage

```
ir_clip(x, range)
```

Arguments

`x`	An object of class `ir`.
`range`	A `data.frame` with two columns and a row for each wavenumber range to keep. The columns are: **start** A numeric vector with start values for wavenumber ranges. **end** A numeric vector with end values for wavenumber ranges. If `range` has more than one row, multiple ranges are clipped from `x` and merged together. Overlapping ranges are not allowed.

Value

An object of class `ir` where spectra have been clipped.

Examples

```
## clipping with one range

# define clipping range
range <-
  data.frame(start = 900, end = 1000)

# clip
x <-
   ir::ir_sample_data %>%
   ir::ir_clip(range = range)

## clipping with mutliple ranges

range <-
  data.frame(start = c(900, 1900), end = c(1000, 2200))

# clip
x <-
   ir::ir_sample_data %>%
   ir::ir_clip(range = range)
```

`ir_divide` *Divide infrared spectra or divide infrared spectra by a numeric value*

Description

`ir_divide` takes two objects of class `ir`, `x` and `y`, and divides their intensity values, or it takes one object of class `ir`, `x`, and one numeric value, `y`, and divides all intensity values in `x` by `y`.

Usage

```
ir_divide(x, y)
```

Arguments

`x`	An object of class `ir`.
`y`	An object of class `ir` or a numeic value. If `y` is an object of class `ir`, it must have the same number of rows as `x` and the same x axis values (e.g. wavenumber values) in each matching spectrum as in `x`.

Value

`x` where for each spectrum intensity values are divided by the respective intensity values in `y` (if `y` is an object of class `ir`), or where all intensity values are divided by `y` if `y` is a numeric value.

Examples

```
# division with y as ir object
x1 <-
  ir::ir_divide(ir::ir_sample_data, ir::ir_sample_data)
x1 <-
  ir::ir_divide(ir::ir_sample_data, ir::ir_sample_data[1, ])

# division with y being a numeric value
x2 <-
  ir::ir_divide(ir::ir_sample_data, y = 20)
```

---

ir_drop_spectra *Drops the column* spectra *from an object is of class* ir

---

Description

ir_drop_spectra removes the column spectra from an object of class ir and removes the "ir" class attribute.

Usage

```
ir_drop_spectra(x)
```

Arguments

x	An object of class ir.

Value

x without column spectra and without "ir" class attribute.

Examples

```
ir::ir_sample_data %>%
  ir_drop_spectra()
```

ir_flatten *Converts objects of class* ir *to objects of class* ir_flat

Description

ir_flatten takes and object of class ir, extracts the spectra column and combines the spectra into an object of class ir_flat. Metadata are not retained during flattening.

Usage

```
ir_flatten(x, measurement_id = as.character(seq_len(nrow(x))))
```

Arguments

x	An object of class ir.
measurement_id	A character vector an element for each row in x that contains the names to use as column names for the spectra in the ir_flat object to create.

Value

An object of class ir_flat.

Examples

```
x_flat <-
   ir::ir_sample_data %>%
   ir::ir_flatten()
```

ir_flat_clean *Cleans objects of class* ir_flat

Description

ir_flatten_clean takes an object of class ir_flat and either returns all non-empty spectra or all empty spectra as object of class ir_flat.

Usage

```
ir_flat_clean(x, return_empty = FALSE)
```

Arguments

x	An object of class ir_flat.
return_empty	A logical value indicating if the empty spectra should be returned (return_empty = TRUE) or the non-empty spectra (return_empty = FALSE).

Value

x where empty spectra are dropped (if return_empty = TRUE) or only empty spectra are returned (return_empty = FALSE).

---

ir_get_intensity *Extracts intensities from spectra in an* ir *object for specific spectral channels*

Description

ir_get_intensity extracts intensity values of spectra for specific user-defined spectral channels ("x axis values", e.g. wavenumber values).

Usage

```
ir_get_intensity(x, wavenumber, warn = TRUE)
```

Arguments

x	An object of class ir.
wavenumber	A numeric vector with spectral channels ("x axis values", e.g. wavenumber values) for which to extract intensities.
warn	logical value indicating if warnings should be displayed (TRUE) or not (FALSE).

Value

x with an additional column intensity. x$intensity is a list column with each element representing a data.frame with a row for each element in wavenumber and two columns:

- **x** The "x axis values" extracted with ir_get_wavenumberindex() applied on wavenumber and the corresponding spectrum in x.
- **y** The extracted intensity values.

Examples

```
x <-
   ir::ir_sample_data %>%
   ir::ir_get_intensity(wavenumber = 1090)
```

ir_get_spectrum *Extracts selected spectra from an object of class* ir

Description

ir_get_spectrum extracts selected spectra from an object of class ir.

Usage

```
ir_get_spectrum(x, what)
```

Arguments

x	An object of class ir.
what	A numeric vector with each element representing a row in x for which to extract the spectrum.

Value

An integer vector with the same length as wavenumber with the row indices of x corresponding to the wavenumber values in wavenumber.

Examples

```
x <-
   ir::ir_sample_data %>%
   ir::ir_get_spectrum(what = c(5, 9))
```

ir_get_wavenumberindex *Gets the index of a defined wavenumber value for a spectrum*

Description

ir_get_wavenumberindex gets for a defined wavenumber value or set of wavenumber values the corresponding indices (row number) in an object of class ir that has been flattened with ir_flatten(). If the specified wavenumber values do not match exactly the wavenumber values in the ir object, the indices for the next wavenumber values will be returned, along with a warning.

Usage

```
ir_get_wavenumberindex(x, wavenumber, warn = TRUE)
```

Arguments

x	A data.frame with a column x representing the x units of a spectrum or several spectra (e.g. in the form of an object of class ir_flat).
wavenumber	A numeric vector with wavenumber values for which to get indices.
warn	logical value indicating if warnings should be displayed (TRUE) or not (FALSE).

Value

An integer vector with the same length as wavenumber with the row indices of x corresponding to the wavenumber values in wavenumber.

Examples

```
x_index_1090 <-
   ir::ir_sample_data %>%
   ir::ir_flatten() %>%
   ir::ir_get_wavenumberindex(wavenumber = 1090)
```

---

ir_import_csv *Imports infrared spectra from various files*

Description

ir_import_csv imports raw infrared spectra from one or more .csv file that contains at least one spectrum, with x axis values (e.g. wavenumbers) in the first column and intensity values of spectra in remaining columns. Note that the function does not perform any checks for the validity of the content read from the .csv file.

Usage

```
ir_import_csv(filenames, sample_id = "from_filenames", ...)
```

Arguments

filenames	A character vector representing the complete paths to the .csv files to import.
sample_id	Either: • NULL: Nothing additional happens. • A character vector with the same length as filenames: This vector will be added as column sample_id to the ir object. • "from_filenames": The file name(s) will be used as values for a new column sample_id to add (the default). • "from_colnames": The header in the csv file will be used as values for a new column sample_id to add.
...	Further arguments passed to read.csv().

Value

An object of class ir containing the infrared spectra extracted from the .csv file(s).

Examples

```
# import data from csv files
d <- ir::ir_import_csv(
  system.file(package = "ir", "extdata/klh_hodgkins_mir.csv"),
  sample_id = "from_colnames")
```

---

ir_import_spc *Imports infrared spectra from Thermo Galactic's files*

Description

ir_import_spc imports raw infrared spectra from a Thermo Galactic's .spc file or several of such files. ir_import_spc is a wrapper function to hyperSpec::read.spc().

Usage

```
ir_import_spc(filenames, log.txt = TRUE)
```

Arguments

filenames	A character vector representing the complete paths to the .spc files to import.
log.txt	A logical value indicating whether to import metadata (TRUE) or not (FALSE). See the details section. If set to FALSE, only the metadata variables exponentiation_factor to measurement_device listed in the Value section below are included in the ir object.

Details

Currently, log.txt must be set to FALSE due to a bug in hyperSpec::read.spc(). This bug fill be fixed in the upcoming weeks and currently can be circumvented by using the development version of 'hyperSpec'. See https://github.com/r-hyperspec/hyperSpec/issues/80.

Value

An object of class ir containing the infrared spectra extracted from the .spc file(s) and the metadata as extracted by hyperSpec::read.spc(). Metadata variables are:

scan_number An integer value representing the number of scans.

detection_gain_factor The detection gain factor.

scan_speed The scan speed [kHz].

laser_wavenumber The wavenumber of the laser.

detector_name The name of the detector.

source_name The name of the infrared radiation source.

purge_delay The duration of purge delay before a measurement [s].

zero_filling_factor A numeric value representing the zero filling factor.

apodisation_function The name of the apodisation function.

exponentiation_factor The exponentiation factor used for file compression.

data_point_number The number of data points in the spectrum

x_variable_type The type of the x variable.

y_variable_type The type of the y variable.

measurement_date A POSIXct representing the measurement date and time.

measurement_device The name of the measurement device.

Examples

```
# import a sample .spc file
x <-
  ir::ir_import_spc(
    system.file("extdata/1.spc", package = "ir"),
    log.txt = FALSE
  )
```

`ir_interpolate` *Interpolates intensity values of infrared spectra in an* `ir` *object for new wavenumber values*

Description

`ir_interpolate` interpolates intensity values for infrared spectra for new wavenumber values.

Usage

```
ir_interpolate(x, start = NULL, dw = 1)
```

Arguments

`x`	An object of class `ir`.
`start`	A numerical value indicating the start wavenumber value relative to which new wavenumber values will be interpolated. The value is not allowed to be < `floor(firstvalue) - 2`, whereby `firstvalue` is the first wavenumber value within `x`. If `start = NULL`, `floor(firstvalue)` will be used as first wavenumber value.
`dw`	A numerical value representing the desired wavenumber value difference between adjacent values.

Value

An object of class `ir` containing the interpolated spectra. Any `NA` values resulting from interpolation will be automatically dropped.

Examples

```
x <-
   ir::ir_sample_data %>%
   ir::ir_interpolate(start = NULL, dw = 1)
```

`ir_interpolate_region` *Interpolates selected regions in infrared spectra in an* `ir` *object*

Description

`ir_interpolate_region` linearly interpolates a user-defined region in infrared spectra.

Usage

```
ir_interpolate_region(x, range)
```

Arguments

`x`	An object of class `ir`.
`range`	A `data.frame` with a row for each region to interpolate linearly and two columns: **start** A numeric vector with start values for regions to interpolate linearly (x axis values). **end** A numeric vector with end values for regions to interpolate linearly (x axis values). For each row in `range`, the values in `range$start` have to be smaller than the values in `range$end`.

Value

`x` with the defined wavenumber region(s) interpolated linearly.

Examples

```
# interpolation range
range <- data.frame(start = 1000, end = 1500)

# do the interpolation
x <-
   ir::ir_sample_data %>%
   ir::ir_interpolate_region(range = range)
```

`ir_multiply` *Multiply infrared spectra or multiply infrared spectra with a numeric value*

Description

`ir_multiply` takes two objects of class `ir`, `x` and `y`, and multiplies their intensity values, or it takes one object of class `ir`, `x`, and one numeric value, `y`, and multiplies all intensity values in `x` with `y`.

Usage

```
ir_multiply(x, y)
```

Arguments

`x`	An object of class `ir`.
`y`	An object of class `ir` or a numeic value. If `y` is an object of class `ir`, it must have the same number of rows as `x` and the same x axis values (e.g. wavenumber values) in each matching spectrum as in `x`.

Value

`x` where for each spectrum intensity values are multiplied with the respective intensity values in `y` (if `y` is an object of class `ir`), or where all intensity values are multiplied with `y` if `y` is a numeric value.

Examples

```
# multiplication with y as ir object
x1 <-
  ir::ir_multiply(ir::ir_sample_data, ir::ir_sample_data)
x1 <-
  ir::ir_multiply(ir::ir_sample_data, ir::ir_sample_data[1, ])

# multiplication with y being a numeric value
x2 <-
  ir::ir_multiply(ir::ir_sample_data, y = -1)
```

`ir_new_ir` *Creates an object of class* `ir`

Description

`ir_new_ir` is the constructor function for objects of class `ir`. An object of class `ir` is a `tibble::tbl_df()` with a sample in each row and a list column containing spectra for each sample.

Usage

```
ir_new_ir(spectra, metadata = tibble::tibble())
```

Arguments

`spectra`	A named list in which each element contains spectral data for one measurement. Each list element must be a `data.frame` with two columns and a row for each wavenumber value in the spectra data. The first column must contain unique wavenumber values and the second column intensity values of the measured spectrum of the sample.
`metadata`	An optional `data.frame` with additional columns containing metadata for the spectra in `spectra`. Optionally, an empty `data.frame` can be defined if no metadata are available.

Value

An object of class `ir` with the following columns:

spectra A list column identical to `spectra`.

... Additional columns contained in `metadata`.

Examples

```
ir_new_ir(
  spectra = ir_sample_data$spectra,
  metadata = ir_sample_data %>% dplyr::select(-spectra)
)
```

ir_new_ir_flat *Creates an object of class* ir_flat

Description

ir_new_ir_flat is the constructor function for objects of class ir_flat. An object of class ir_flat is a data.frame where the first column ("x") contains unique x values of spectra (e.g. wavenumbers) and all remaining columns represent intensity values from spectra corresponding to the x values.

Usage

```
ir_new_ir_flat(x)
```

Arguments

x	A data.frame with only numeric columns and only the first column name being "x".

Value

An object of class ir_flat.

Examples

```
x_flat <-
   ir::ir_sample_data %>%
   ir::ir_flatten()
```

ir_normalize *Normalizes infrared spectra in an* ir *object*

Description

ir_normalize normalizes the intensity values of infrared spectra. Spectra can be normalized in three ways (value for argument method):

"zeroone" Normalization so that the intensity values range in [0;1]. Note that for different spectra, for different wavenumber values the intensity may be 1 after normalization, depending on the location of the peak with the maximum height.

"area" Normalization so that the intensity values of each spectrum sum to 1. Note that in the case of negative intensities values, these will be count as negative values during summation.

A numeric value Normalization so that the intensity at a specified wavenumber value has value 1 and the minimum intensity value is 0.

Usage

```
ir_normalize(x, method = "area")

ir_normalise(x, method = "area")
```

Arguments

x	An object of class ir.
method	A character value specifying which normalization method to apply. If method = "zeroone", all intensity values will be normalized to [0;1]. If method = "area", all intensity values will be divided by the sum of the intensity values at all wavenumber values of the spectrum. If method is convertible to a numeric value, e.g. method = "980", the intensity of all spectra at a wavenumber value of 980 will be set to 1 and the minimum intensity value of each spectrum will be set to 0, i.e. the spectra will be normalized referring to a specific wavenumber value.

Value

An object of class ir representing a normalized version of x.

Examples

```
# with method = "area"
x <-
   ir::ir_sample_data %>%
   ir::ir_normalize(method = "area")

# normalizing to a specific peak
x <-
   ir::ir_sample_data %>%
   ir::ir_normalize(method = 1090)
```

---

ir_remove_missing *Removes empty data values in an object of class* ir

---

Description

ir_remove_missing takes and object of class ir and removes all rows in the data.frames of the list column spectra that have NA intensity values (column y). Additionally, one can specify to remove rows in the ir object to discard if they contain empty spectra.

Usage

```
ir_remove_missing(x, remove_rows = FALSE)
```

Arguments

x	An object of class ir.
remove_rows	A logical value indicating if rows in x with empty spectra should be discarded (remove_rows = TRUE) or not (remove_rows = FALSE).

Value

x with cleaned spectra.

Examples

```
# create sample data with some missing rows and one entire missing spectra
x <-
   ir::ir_sample_data
x$spectra[[1]] <- x$spectra[[1]][0, ]
x$spectra[[2]][1:100, "y"] <- NA_real_

# remove missing values (but remove no rows in x)
x1 <-
   x %>%
   ir::ir_remove_missing(remove_rows = FALSE)

# remove missing values (and remove rows in x if a compete spectrum is
# missing)
x2 <-
   x %>%
   ir::ir_remove_missing(remove_rows = TRUE)

nrow(x)
nrow(x1)
nrow(x2)
```

---

ir_sample_data *Sample object of class* ir

Description

A sample object of class ir. The data set contains ATR-MIR spectra for a set of organic reference materials along with their metadata (types of samples and a description) and accessory data (Klason lignin mass fraction and holocellulose mass fraction).

Usage

```
ir_sample_data
```

Format

A data frame with 58 rows and 7 variables:

id_measurement An integer vector with a unique id for each spectrum.

id_sample A character vector with a unique id for each sample.

sample_type A character vector containing class labels for the types of reference materials.

sample_comment A character vector containing comments to each sample.

klason_lignin A numeric vector with the mass fractions of Klason lignin in each sample.

holocellulose A numeric vector with the mass fractions of holocellulose in each sample.

spectra See `ir_new_ir()`.

Source

The data set was derived from https://www.nature.com/articles/s41467-018-06050-2 and published by Hodgkins et al. (2018) under the CC BY 4.0 license https://creativecommons.org/licenses/by/4.0/. Hodgkins et al. (2018) originally derived the data on Klason Lignin and Holocellulose content from De La Cruz, Florentino B. et al. (2016).

References

De La Cruz, Florentino B., Osborne J, Barlaz MA (2016). "Determination of Sources of Organic Matter in Solid Waste by Analysis of Phenolic Copper Oxide Oxidation Products of Lignin." *Journal of Environmental Engineering*, **142**(2), 04015076. ISSN 0733-9372, doi:10.1061/(ASCE)EE.1943-7870.0001038.

Hodgkins SB, Richardson CJ, Dommain R, Wang H, Glaser PH, Verbeke B, Winkler BR, Cobb AR, Rich VI, Missilmani M, Flanagan N, Ho M, Hoyt AM, Harvey CF, Vining SR, Hough MA, Moore TR, Richard PJH, De La Cruz, Florentino B., Toufaily J, Hamdan R, Cooper WT, Chanton JP (2018). "Tropical peatland carbon storage linked to global latitudinal trends in peat recalcitrance." *Nature communications*, **9**(1), 3640. doi:10.1038/s41467018060502.

`ir_smooth` *Smooths infrared spectra in an* `ir` *object*

Description

`ir_smooth` applies smoothing functions to infrared spectra. `ir_smooth` either performs Savitzky-Golay smoothing, using on `signal::sgolayfilt()`, or Fourier smoothing using `fda::smooth.basis()`. Savitzky-Golay smoothing can also be used to compute derivatives of spectra.

Usage

```
ir_smooth(
  x,
  method = "sg",
  p = 3,
  n = p + 3 - p%%2,
  ts = 1,
  m = 0,
  k = 111,
  ...
)
```

Arguments

x	An object of class ir.
method	A character value specifying which smoothing method to apply. If method = "sg", a Savitzky-Golay filter will be applied on the spectra. The Savitzky-Golay smoothing will be performed using the function signal::sgolayfilt(). If method = "fourier", Fourier smoothing will be performed. Fourier transformation of the spectra is performed using the fast discrete Fourier transformation (FFT) as implemented in fda::smooth.basis(). A smoothing function can be defined by the argment f.
p	An integer value representing the filter order (i.e. the degree of the polynom) of the Savitzky-Golay filter if method = "sg".
n	An odd integer value representing the length (i.e. the number of wavenumber values used to construct the polynom) of the Savitzky-Golay filter if method = "sg".
ts	time scaling factor. See signal::sgolayfilt().
m	An integer value representing the mth derivative to compute. This option can be used to compute derivatives of spectra. See signal::sgolayfilt().
k	A positive odd integer representing the number of Fourier basis functions to use as smoothed representation of the spectra if method = "fourier".
...	additional arguments (ignored).

Details

When x contains spectra with different wavenumber values, the filters are applied for each spectra only on existing wavenumber values. This means that the filter window (if method == "sg") will be different for these different spectra.

Value

x with smoothed spectra.

Examples

```
#' # Savitzky-Golay smoothing
x1 <-
   ir::ir_sample_data[1:5, ] %>%
   ir::ir_smooth(method = "sg", p = 3, n = 51, ts = 1, m = 0)

# Fourier smoothing
x2 <-
   ir::ir_sample_data[1:5, ] %>%
   ir::ir_smooth(method = "fourier", k = 21)

# computing derivative spectra with Savitzky-Golay smoothing (here: first
# derivative)
x3 <-
   ir::ir_sample_data[1:5, ] %>%
   ir::ir_smooth(method = "sg", p = 3, n = 51, ts = 1, m = 1)
```

`ir_stack` *Stacks a matrix or data frame with spectra into a list column*

Description

`ir_stack` takes a matrix or data frame with infrared spectra and converts it into a list column corresponding to the column `spectra` in objects of class `ir`.

Usage

```
ir_stack(x)
```

Arguments

`x`	A matrix or data frame with a first column (`x`) containing "x axis values" of the spectra (e.g. wavenumbers) and all remaining columns containing intensity values of spectra.

Value

A `tibble::tibble()` with the stacked spectra in column `spectra`.

Examples

```
# from data frame
x1 <-
   ir::ir_sample_data %>%
   ir::ir_flatten() %>%
   ir::ir_stack()

# from matrix
```

```
x2 <-
   ir::ir_sample_data %>%
   ir::ir_flatten() %>%
   as.matrix() %>%
   ir::ir_stack()
```

`ir_subtract` *Subtract infrared spectra*

Description

`ir_subtract` takes two objects of class `ir`, `x` and `y`, and subtracts the intensity values of spectra in matching rows from `y` from that of `x`. Alternatively, takes an object of class `ir`, `x`, and a numeric value, `y`, and subtracts `y` from all intensity values in `x`.

Usage

```
ir_subtract(x, y)
```

Arguments

`x`	An object of class `ir`.
`y`	An object of class `ir` or a numeic value. If `y` is an object of class `ir`, it must have the same number of rows as `x` and the same x axis values (e.g. wavenumber values) in each matching spectrum as in `x`.

Value

`x` where for each spectrum the respective intensity values in `y` are subtracted (if `y` is an object of class `ir`), or where for each spectrum `y` has been subtracted from the intensity values.

Examples

```
# subtracting two objects of class ir
x1 <-
  ir::ir_subtract(ir::ir_sample_data, ir::ir_sample_data)
x1 <-
  ir::ir_subtract(ir::ir_sample_data, ir::ir_sample_data[1, ])

# subtracting a numeric value from an object of class `ir`.
x2 <-
  ir::ir_subtract(ir::ir_sample_data, 20)
```

`ir_to_transmittance` *Converts absorbance spectra to transmittance spectra or vice versa*

Description

`ir_to_transmittance` converts absorbance spectra to transmittance spectra. `ir_to_absorbance` converts transmittance spectra to absorbance spectra. Note that neither function checks whether the input spectra are absorbance or transmittance spectra.

Usage

```
ir_to_transmittance(x)

ir_to_absorbance(x)
```

Arguments

`x`	An object of class `ir`.

Value

`x` with y values fore each spectrum as transmittance values (in case of `ir_to_transmittance`) or absorbance values (in case of `ir_to_absorbance`).

Source

(Stuart 2004).

References

Stuart BH (2004). *Infrared Spectroscopy: Fundamentals and Applications: Stuart/Infrared Spectroscopy: Fundamentals and Applications*, Analytical Techniques in the Sciences. John Wiley \& Sons, Ltd, Chichester, UK. ISBN 978-0-470-01114-0 978-0-470-85428-0, doi:10.1002/0470011149.

Examples

```
# convert from absorbance to transmittance
x1 <-
    ir_sample_data %>%
    ir_to_transmittance()

# convert from transmittance to absorbance
x2 <-
    x1 %>%
    ir::ir_to_absorbance()

vapply(
  seq_along(x2$spectra),
  FUN = function(i) all.equal(x2$spectra[[i]], ir::ir_sample_data$spectra[[i]]),
```

```
  FUN.VALUE = logical(1L)
) %>%
  all()
```

---

ir_variance_region *Computes the variance of a spectrum in an* ir *object in a given region*

Description

ir_variance_region takes a spectrum x and, depending on the arguments computes the following summary:

if subtract_smoothed = FALSE it computes the variance of the intensity values for each spectrum in x. If in addition range is not NULL, it computes the variance only for the region(s) represented by range.

if subtract_smoothed = TRUE it smoothes x, subtracts the smoothed x from the unsmoothed x and computes the variance of the difference intensity values. If in addition range is not NULL, it computes the variance only for the region(s) represented by range.

Usage

```
ir_variance_region(
  x,
  subtract_smoothed = FALSE,
  do_normalize = FALSE,
  normalize_method,
  ...,
  range = NULL
)
```

Arguments

x	An object of class ir. These are the spectra for which to compute the variance.
subtract_smoothed	A logical value. If subtract_smoothed = TRUE, x is copied, the copy smoothed using ir_smooth with method = "sg" and subtracted from x before the variance of the intensity values from x is computed. This allows e.g. to estimate the noise level in a specific region of spectra. If subtract_smoothed = FALSE (the default), nothing is subtracted from x before computing the variance of the intensity values.
do_normalize	A logical value. If set to TRUE, the spectra in x are normalized after subtraction of a smoothed version, else no normalization is performed.
normalize_method	See ir_normalize().

...	Arguments passed to ir_smooth() (except for method which is always set to "sg" if subtract_smoothed is TRUE). If subtract_smoothed = FALSE, these arguments will be ignored.
range	See ir_clip(). This is the range for which the variance of the intensity values will be computed.

Value

x with two additional columns:

variance A numeric vector with the computed variances of the intensity values for the respective spectra.

n_variance An integer vector with the number of intensity values used during computing the variance.

Examples

```
# Whole spectra variance
x1 <-
   ir::ir_sample_data %>%
   ir::ir_variance_region(
      subtract_smoothed = FALSE,
      do_normalize = TRUE,
      normalize_method = "area",
      range = NULL
   )

# Spectra variance, but only from a specific region
range <- data.frame(start = 2700, end = 2800)

x2 <-
   ir::ir_sample_data %>%
   ir::ir_normalize(method = "area") %>%
   ir::ir_variance_region(
      subtract_smoothed = FALSE,
      do_normalize = TRUE,
      normalize_method = "area",
      range = range
   )

# Spectra variance after subtracting a smoothed version of the spectra and
# only from a specific region
x3 <-
   ir::ir_sample_data %>%
   ir::ir_variance_region(
      subtract_smoothed = TRUE,
      do_normalize = FALSE,
      range = range,
      p = 3, n = 31, ts = 1, m = 0
   )
```

`mutate` *Mutate an* `ir` *object by adding new or replacing existing columns*

Description

Mutate an `ir` object by adding new or replacing existing columns

Usage

```
mutate.ir(
  .data,
  ...,
  .keep = c("all", "used", "unused", "none"),
  .before = NULL,
  .after = NULL
)

transmute.ir(.data, ...)
```

Arguments

`.data`	An object of class `ir`.
`...`	<data-masking> Name-value pairs. The name gives the name of the column in the output. The value can be: • A vector of length 1, which will be recycled to the correct length. • A vector the same length as the current group (or the whole data frame if ungrouped). • `NULL`, to remove the column. • A data frame or tibble, to create multiple columns in the output.
`.keep`	**[Experimental]** Control which columns from `.data` are retained in the output. Grouping columns and columns created by `...` are always kept. • `"all"` retains all columns from `.data`. This is the default. • `"used"` retains only the columns used in `...` to create new columns. This is useful for checking your work, as it displays inputs and outputs side-by-side. • `"unused"` retains only the columns *not* used in `...` to create new columns. This is useful if you generate new columns, but no longer need the columns used to generate them. • `"none"` doesn't retain any extra columns from `.data`. Only the grouping variables and columns created by `...` are kept.
`.before, .after`	**[Experimental]** <tidy-select> Optionally, control where new columns should appear (the default is to add to the right hand side). See `relocate()` for more details.

Value

.data with modified columns. If the spectra column is dropped or invalidated (see ir_new_ir()), the ir class is dropped, else the object is of class ir.

Source

dplyr::mutate()

See Also

Other tidyverse: arrange.ir(), distinct.ir(), extract.ir(), filter-joins, filter.ir(), group_by, mutate-joins, nest, pivot_longer.ir(), pivot_wider.ir(), rename, rowwise.ir(), select.ir(), separate.ir(), separate_rows.ir(), slice, summarize, unite.ir()

Examples

```
## mutate
dplyr::mutate(ir_sample_data, hkl = klason_lignin + holocellulose)


## transmute
dplyr::transmute(ir_sample_data, hkl = klason_lignin + holocellulose)
```

---

mutate-joins *Mutating joins for an* ir *object*

Description

Mutating joins for an ir object

Usage

```
inner_join.ir(
  x,
  y,
  by = NULL,
  copy = FALSE,
  suffix = c(".x", ".y"),
  ...,
  keep = FALSE,
  na_matches = c("na", "never")
)

left_join.ir(
  x,
  y,
```

```
  by = NULL,
  copy = FALSE,
  suffix = c(".x", ".y"),
  ...,
  keep = FALSE,
  na_matches = c("na", "never")
)

right_join.ir(
  x,
  y,
  by = NULL,
  copy = FALSE,
  suffix = c(".x", ".y"),
  ...,
  keep = FALSE,
  na_matches = c("na", "never")
)

full_join.ir(
  x,
  y,
  by = NULL,
  copy = FALSE,
  suffix = c(".x", ".y"),
  ...,
  keep = FALSE,
  na_matches = c("na", "never")
)
```

Arguments

x	An object of class `ir`.
y	A data frame.
by	A character vector of variables to join by. If `NULL`, the default, `*_join()` will perform a natural join, using all variables in common across `x` and `y`. A message lists the variables so that you can check they're correct; suppress the message by supplying `by` explicitly. To join by different variables on `x` and `y`, use a named vector. For example, `by = c("a" = "b")` will match `x$a` to `y$b`. To join by multiple variables, use a vector with length > 1. For example, `by = c("a", "b")` will match `x$a` to `y$a` and `x$b` to `y$b`. Use a named vector to match different variables in `x` and `y`. For example, `by = c("a" = "b", "c" = "d")` will match `x$a` to `y$b` and `x$c` to `y$d`. To perform a cross-join, generating all combinations of `x` and `y`, use `by = character()`.
copy	If `x` and `y` are not from the same data source, and `copy` is `TRUE`, then `y` will be copied into the same src as `x`. This allows you to join tables across srcs, but it is a potentially expensive operation so you must opt into it.

suffix	If there are non-joined duplicate variables in x and y, these suffixes will be added to the output to disambiguate them. Should be a character vector of length 2.
...	Other parameters passed onto methods.
keep	Should the join keys from both x and y be preserved in the output?
na_matches	Should NA and NaN values match one another? The default, "na", treats two NA or NaN values as equal, like %in%, match(), merge(). Use "never" to always treat two NA or NaN values as different, like joins for database sources, similarly to merge(incomparables = FALSE).

Value

x and y joined. If the spectra column is renamed, the ir class is dropped. See mutate-joins.

Source

mutate-joins

See Also

Other tidyverse: arrange.ir(), distinct.ir(), extract.ir(), filter-joins, filter.ir(), group_by, mutate, nest, pivot_longer.ir(), pivot_wider.ir(), rename, rowwise.ir(), select.ir(), separate.ir(), separate_rows.ir(), slice, summarize, unite.ir()

Examples

```
## inner_join
set.seed(234)
dplyr::inner_join(
  ir_sample_data,
  tibble::tibble(
    id_measurement = c(1:5, 101:105),
    nitrogen_content = rbeta(n = 10, 0.2, 0.1)
  ),
  by = "id_measurement"
)


## left_join
set.seed(234)
dplyr::left_join(
  ir_sample_data,
  tibble::tibble(
    id_measurement = c(1:5, 101:105),
    nitrogen_content = rbeta(n = 10, 0.2, 0.1)
  ),
  by = "id_measurement"
)
```

```
## right_join
set.seed(234)
dplyr::right_join(
  ir_sample_data,
  tibble::tibble(
    id_measurement = c(1:5, 101:105),
    nitrogen_content = rbeta(n = 10, 0.2, 0.1)
  ),
  by = "id_measurement"
)


## full_join
set.seed(234)
dplyr::full_join(
  ir_sample_data,
  tibble::tibble(
    id_measurement = c(1:5, 101:105),
    nitrogen_content = rbeta(n = 10, 0.2, 0.1)
  ),
  by = "id_measurement"
)
```

`nest` *Nest and un-nest an* `ir` *object*

Description

Nest and un-nest an `ir` object

Usage

```
nest.ir(.data, ..., .names_sep = NULL, .key = deprecated())

unnest.ir(
  data,
  cols,
  ...,
  keep_empty = FALSE,
  ptype = NULL,
  names_sep = NULL,
  names_repair = "check_unique",
  .drop = deprecated(),
  .id = deprecated(),
  .sep = deprecated(),
  .preserve = deprecated()
)
```

Arguments

.data	An object of class ir.
...	<tidy-select> Columns to nest, specified using name-variable pairs of the form new_col = c(col1, col2, col3). The right hand side can be any valid tidy select expression. **[Deprecated]**: previously you could write df %>% nest(x, y, z) and df %>% unnest(x, y, z). Convert to df %>% nest(data = c(x, y, z)). and df %>% unnest(c(x, y, z)). If you previously created new variable in unnest() you'll now need to do it explicitly with mutate(). Convert df %>% unnest(y = fun(x, y, z)) to df %>% mutate(y = fun(x, y, z)) %>% unnest(y).
.key	**[Deprecated]**: No longer needed because of the new new_col = c(col1, col2, col3) syntax.
data	A data frame.
cols	<tidy-select> Columns to unnest. If you unnest() multiple columns, parallel entries must be of compatible sizes, i.e. they're either equal or length 1 (following the standard tidyverse recycling rules).
keep_empty	By default, you get one row of output for each element of the list your unchopping/unnesting. This means that if there's a size-0 element (like NULL or an empty data frame), that entire row will be dropped from the output. If you want to preserve all rows, use keep_empty = TRUE to replace size-0 elements with a single row of missing values.
ptype	Optionally, a named list of column name-prototype pairs to coerce cols to, overriding the default that will be guessed from combining the individual values. Alternatively, a single empty ptype can be supplied, which will be applied to all cols.
names_sep, .names_sep	If NULL, the default, the names will be left as is. In nest(), inner names will come from the former outer names; in unnest(), the new outer names will come from the inner names. If a string, the inner and outer names will be used together. In unnest(), the names of the new outer columns will be formed by pasting together the outer and the inner column names, separated by names_sep. In nest(), the new inner names will have the outer names + names_sep automatically stripped. This makes names_sep roughly symmetric between nesting and unnesting.
names_repair	Used to check that output data frame has valid names. Must be one of the following options: • "minimal": no name repair or checks, beyond basic existence, • "unique": make sure names are unique and not empty, • "check_unique": (the default), no name repair, but check they are unique, • "universal": make the names unique and syntactic • a function: apply custom name repair. • tidyr_legacy: use the name repair from tidyr 0.8.

- a formula: a purrr-style anonymous function (see `rlang::as_function()`)

See `vctrs::vec_as_names()` for more details on these terms and the strategies used to enforce them.

`.drop, .preserve`
: **[Deprecated]**: all list-columns are now preserved; If there are any that you don't want in the output use `select()` to remove them prior to unnesting.

`.id`
: **[Deprecated]**: convert `df %>% unnest(x, .id = "id")` to `df %>% mutate(id = names(x)) %>% unnest(`

`.sep`
: **[Deprecated]**: use `names_sep` instead.

Value

`.data` with nested or unnested columns. If the `spectra` column is dropped or invalidated (see `ir_new_ir()`), the `ir` class is dropped, else the object is of class `ir`.

Source

`tidyr::nest()`

See Also

Other tidyverse: `arrange.ir()`, `distinct.ir()`, `extract.ir()`, `filter-joins`, `filter.ir()`, `group_by`, `mutate-joins`, `mutate`, `pivot_longer.ir()`, `pivot_wider.ir()`, `rename`, `rowwise.ir()`, `select.ir()`, `separate.ir()`, `separate_rows.ir()`, `slice`, `summarize`, `unite.ir()`

Examples

```
## nest
ir_sample_data %>%
  tidyr::nest(
    contents = c(holocellulose, klason_lignin)
  )


## unnest
ir_sample_data %>%
  tidyr::nest(
    contents = c(holocellulose, klason_lignin)
  ) %>%
  tidyr::unnest("contents")
```

`Ops.ir` *Arithmetic operations for* `ir` *objects*

Description

Arithmetic operations for `ir` objects

Usage

```
## S3 method for class 'ir'
Ops(e1, e2)
```

Arguments

`e1`	An object of class `ir`.
`e2`	An object of class `ir` or a numeric value.

Value

`e1` with intensity values of the spectra added to/subtracted with/multiplied with/divided by those in `e2`:

- If `e2` is a numeric value, all intensity values in the spectra of `e1` are added/subtracted/multiplied/divided by `e2`.
- If `e2` is an `ir` object with one row, it is replicated (see rep.ir) so that the row numbers match to those of `e1` and intensity values are added/subtracted/multiplied/divided row-wise.
- If `e2` is an `ir` object with the same number of rows as `e1`, intensity values are added/subtracted/multiplied/divided row-wise.

Examples

```
## addition
ir::ir_sample_data + ir::ir_sample_data
ir::ir_sample_data + 2

## subtraction
ir::ir_sample_data - ir::ir_sample_data
ir::ir_sample_data - 2

## multiplication
ir::ir_sample_data * ir::ir_sample_data
ir::ir_sample_data * 2

## division
ir::ir_sample_data / ir::ir_sample_data
ir::ir_sample_data / 2
```

pivot_longer.ir *Pivot an* ir *object from wide to long*

Description

Pivot an ir object from wide to long

Usage

```
pivot_longer.ir(
  data,
  cols,
  names_to = "name",
  names_prefix = NULL,
  names_sep = NULL,
  names_pattern = NULL,
  names_ptypes = list(),
  names_transform = list(),
  names_repair = "check_unique",
  values_to = "value",
  values_drop_na = FALSE,
  values_ptypes = list(),
  values_transform = list(),
  ...
)
```

Arguments

data	An object of class ir.
cols	<tidy-select> Columns to pivot into longer format.
names_to	A character vector specifying the new column or columns to create from the information stored in the column names of data specified by cols. • If length 0, or if NULL is supplied, no columns will be created. • If length 1, a single column will be created which will contain the column names specified by cols. • If length >1, multiple columns will be created. In this case, one of names_sep or names_pattern must be supplied to specify how the column names should be split. There are also two additional character values you can take advantage of: – NA will discard the corresponding component of the column name. – ".value" indicates that the corresponding component of the column name defines the name of the output column containing the cell values, overriding values_to entirely.
names_prefix	A regular expression used to remove matching text from the start of each variable name.

`names_sep, names_pattern`
: If `names_to` contains multiple values, these arguments control how the column name is broken up.

  `names_sep` takes the same specification as `separate()`, and can either be a numeric vector (specifying positions to break on), or a single string (specifying a regular expression to split on).

  `names_pattern` takes the same specification as `extract()`, a regular expression containing matching groups (`()`).

  If these arguments do not give you enough control, use `pivot_longer_spec()` to create a spec object and process manually as needed.

`names_ptypes, values_ptypes`
: Optionally, a list of column name-prototype pairs. Alternatively, a single empty prototype can be supplied, which will be applied to all columns. A prototype (or ptype for short) is a zero-length vector (like `integer()` or `numeric()`) that defines the type, class, and attributes of a vector. Use these arguments if you want to confirm that the created columns are the types that you expect. Note that if you want to change (instead of confirm) the types of specific columns, you should use `names_transform` or `values_transform` instead.

  For backwards compatibility reasons, supplying `list()` is interpreted as being identical to `NULL` rather than as using a list prototype on all columns. Expect this to change in the future.

`names_transform, values_transform`
: Optionally, a list of column name-function pairs. Alternatively, a single function can be supplied, which will be applied to all columns. Use these arguments if you need to change the types of specific columns. For example, `names_transform = list(week = as.integer)` would convert a character variable called `week` to an integer.

  If not specified, the type of the columns generated from `names_to` will be character, and the type of the variables generated from `values_to` will be the common type of the input columns used to generate them.

`names_repair`
: What happens if the output has invalid column names? The default, `"check_unique"` is to error if the columns are duplicated. Use `"minimal"` to allow duplicates in the output, or `"unique"` to de-duplicated by adding numeric suffixes. See `vctrs::vec_as_names()` for more options.

`values_to`
: A string specifying the name of the column to create from the data stored in cell values. If `names_to` is a character containing the special `.value` sentinel, this value will be ignored, and the name of the value column will be derived from part of the existing column names.

`values_drop_na`
: If `TRUE`, will drop rows that contain only `NA`s in the `value_to` column. This effectively converts explicit missing values to implicit missing values, and should generally be used only when missing values in `data` were created by its structure.

`...`
: Additional arguments passed on to methods.

Value

`data` in a long format. If the `spectra` column is dropped or invalidated (see `ir_new_ir()`), the `ir` class is dropped, else the object is of class `ir`.

Source

tidyr::pivot_longer()

See Also

Other tidyverse: arrange.ir(), distinct.ir(), extract.ir(), filter-joins, filter.ir(), group_by, mutate-joins, mutate, nest, pivot_wider.ir(), rename, rowwise.ir(), select.ir(), separate.ir(), separate_rows.ir(), slice, summarize, unite.ir()

Examples

```
## pivot_longer
ir_sample_data %>%
  tidyr::pivot_longer(
    cols = dplyr::any_of(c("holocellulose", "klason_lignin"))
  )
```

---

pivot_wider.ir *Pivot an* ir *object from wide to long*

Description

Pivot an ir object from wide to long

Usage

```
pivot_wider.ir(
  data,
  id_cols = NULL,
  names_from = "name",
  names_prefix = "",
  names_sep = "_",
  names_glue = NULL,
  names_sort = FALSE,
  names_repair = "check_unique",
  values_from = "value",
  values_fill = NULL,
  values_fn = NULL,
  ...
)
```

Arguments

data	An object of class ir.
id_cols	<tidy-select> A set of columns that uniquely identifies each observation. Defaults to all columns in data except for the columns specified in names_from and values_from. Typically used when you have redundant variables, i.e. variables whose values are perfectly correlated with existing variables.
names_from, values_from	<tidy-select> A pair of arguments describing which column (or columns) to get the name of the output column (names_from), and which column (or columns) to get the cell values from (values_from). If values_from contains multiple values, the value will be added to the front of the output column.
names_prefix	String added to the start of every variable name. This is particularly useful if names_from is a numeric vector and you want to create syntactic variable names.
names_sep	If names_from or values_from contains multiple variables, this will be used to join their values together into a single string to use as a column name.
names_glue	Instead of names_sep and names_prefix, you can supply a glue specification that uses the names_from columns (and special .value) to create custom column names.
names_sort	Should the column names be sorted? If FALSE, the default, column names are ordered by first appearance.
names_repair	What happens if the output has invalid column names? The default, "check_unique" is to error if the columns are duplicated. Use "minimal" to allow duplicates in the output, or "unique" to de-duplicated by adding numeric suffixes. See vctrs::vec_as_names() for more options.
values_fill	Optionally, a (scalar) value that specifies what each value should be filled in with when missing. This can be a named list if you want to apply different fill values to different value columns.
values_fn	Optionally, a function applied to the value in each cell in the output. You will typically use this when the combination of id_cols and names_from columns does not uniquely identify an observation. This can be a named list if you want to apply different aggregations to different values_from columns.
...	Additional arguments passed on to methods.

Value

data in a wide format. If the spectra column is dropped or invalidated (see ir_new_ir()), the ir class is dropped, else the object is of class ir.

Source

tidyr::pivot_wider()

See Also

Other tidyverse: arrange.ir(), distinct.ir(), extract.ir(), filter-joins, filter.ir(), group_by, mutate-joins, mutate, nest, pivot_longer.ir(), rename, rowwise.ir(), select.ir(), separate.ir(), separate_rows.ir(), slice, summarize, unite.ir()

Examples

```
## pivot_wider
ir_sample_data %>%
  tidyr::pivot_longer(
    cols = dplyr::any_of(c("holocellulose", "klason_lignin"))
  ) %>%
  tidyr::pivot_wider(names_from = "name", values_from = "value")
```

---

plot.ir *Plots an object of class* ir

Description

plot.ir is the plot method for objects of class ir.

Usage

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

Arguments

x	An object of class ir.
...	Further arguments, will be ignored.

Value

An object of class ggplot2.

Examples

```
# simple plotting
plot(ir::ir_sample_data[1:2, ])

# advanced functions
plot(ir::ir_sample_data) +
   ggplot2::facet_wrap(~ sample_type)
```

range *Get the minima/maxima/range/median of x axis values or intensity values of infrared spectra*

Description

`range.ir` extracts the range of x axis values (e.g. wavenumbers) or intensity values of infrared spectra.

`min.ir` extracts the minimum x axis value (e.g. wavenumber) or intensity value of infrared spectra.

`max.ir` extracts the maximum x axis value (e.g. wavenumber) or intensity value of infrared spectra.

`median.ir` extracts the median x axis value (e.g. wavenumber) or intensity value of infrared spectra.

Usage

```
## S3 method for class 'ir'
range(
  x,
  ...,
  na.rm = FALSE,
  .dimension = "y",
  .col_names = c("y_min", "y_max")
)

## S3 method for class 'ir'
min(x, ..., na.rm = FALSE, .dimension = "y", .col_name = "y_min")

## S3 method for class 'ir'
max(x, ..., na.rm = FALSE, .dimension = "y", .col_name = "y_max")

## S3 method for class 'ir'
median(x, na.rm = FALSE, ..., .dimension = "y", .col_name = "y_median")
```

Arguments

`x`	An object of class `ir`.
`...`	Further arguments, ignored.
`na.rm`	A logical value. See `max()`.
`.dimension`	A character value. Must be one of the following: **"x"** In this case, the minimum/maximum/range/median of x axis values of the spectra in `x` are extracted. **"y"** In this case, the minimum/maximum/range/median of intensity values of the spectra in `x` are extracted.
`.col_names`	A character vector of length 2 representing the names of the columns in which to store the extracted values. The first element is the name for the column with minima values, the second the name for the column with maxima values.

.col_name	A character value representing the name of the column in which to store the extracted values.

Value

x with the extracted values.

Examples

```
# range of intensity values
x1 <-
   ir::ir_sample_data %>%
   range(.dimension = "y")

# minimum intensity values
x1 <-
   ir::ir_sample_data %>%
   min(.dimension = "y")

# maximum intensity values
x1 <-
   ir::ir_sample_data %>%
   max(.dimension = "y")

# median intensity values
x1 <-
   ir::ir_sample_data %>%
   stats::median(.dimension = "y")
```

---

rename *Rename columns in* ir *objects*

Description

Rename columns in ir objects

Usage

```
rename.ir(.data, ...)

rename_with.ir(.data, .fn, .cols = dplyr::everything(), ...)
```

Arguments

.data	An object of class ir.
...	For rename(): <tidy-select> Use new_name = old_name to rename selected variables. For rename_with(): additional arguments passed onto .fn.

.fn	A function used to transform the selected .cols. Should return a character vector the same length as the input.
.cols	<tidy-select> Columns to rename; defaults to all columns.

Value

.data with renamed columns. If the spectra column is renamed, and no new valid spectra column is created, the ir class is dropped, else the object is of class ir.

Source

dplyr::rename()

See Also

Other tidyverse: arrange.ir(), distinct.ir(), extract.ir(), filter-joins, filter.ir(), group_by, mutate-joins, mutate, nest, pivot_longer.ir(), pivot_wider.ir(), rowwise.ir(), select.ir(), separate.ir(), separate_rows.ir(), slice, summarize, unite.ir()

Examples

```
## rename
dplyr::rename(ir_sample_data, hol = "holocellulose")
dplyr::rename(ir_sample_data, spec = "spectra") # drops ir class


## rename_with
dplyr::rename_with(ir_sample_data, .cols = dplyr::starts_with("id_"),
  toupper)
dplyr::rename_with(ir_sample_data, toupper) # drops ir class
```

---

rep.ir *Replicate ir objects*

Description

rep.ir is the replicate method for ir objects. Replicating and ir object means to replicate its rows and bind these together to a larger ir object.

Usage

```
## S3 method for class 'ir'
rep(x, ...)
```

Arguments

x	An object of class ir.
...	See rep().

Value

An object of class ir with replicated spectra.

Examples

```
# replicate the sample data
x <- rep(ir::ir_sample_data, times = 2)
x <- rep(ir::ir_sample_data, each = 2)
x <- rep(ir::ir_sample_data, length.out = 3)
```

---

rowwise.ir *Group input* ir *objects by rows*

Description

Group input ir objects by rows

Usage

```
rowwise.ir(.data, ...)
```

Arguments

.data	Input data frame.
...	<tidy-select> Variables to be preserved when calling summarise(). This is typically a set of variables whose combination uniquely identify each row. **NB**: unlike group_by() you can not create new variables here but instead you can select multiple variables with (e.g.) everything().
data	An object of class ir.

Value

data as row-wise data frame. See dplyr::rowwise().

Source

dplyr::rowwise()

See Also

Other tidyverse: arrange.ir(), distinct.ir(), extract.ir(), filter-joins, filter.ir(), group_by, mutate-joins, mutate, nest, pivot_longer.ir(), pivot_wider.ir(), rename, select.ir(), separate.ir(), separate_rows.ir(), slice, summarize, unite.ir()

Examples

```
## rowwise
dplyr::rowwise(ir_sample_data) %>%
  dplyr::mutate(
    hkl =
      mean(
        units::drop_units(klason_lignin),
        units::drop_units(holocellulose)
      )
  )
```

---

select.ir *Subset columns in* ir *objects using column names and types*

Description

Subset columns in ir objects using column names and types

Usage

```
select.ir(.data, ...)
```

Arguments

.data	An object of class ir.
...	<tidy-select> One or more unquoted expressions separated by commas. Variable names can be used as if they were positions in the data frame, so expressions like x:y can be used to select a range of variables.

Value

.data with the selected columns. If the spectra column is dropped, the ir class is dropped, else the object is of class ir.

Source

dplyr::select()

See Also

Other tidyverse: arrange.ir(), distinct.ir(), extract.ir(), filter-joins, filter.ir(), group_by, mutate-joins, mutate, nest, pivot_longer.ir(), pivot_wider.ir(), rename, rowwise.ir(), separate.ir(), separate_rows.ir(), slice, summarize, unite.ir()

Examples

```
## select
dplyr::select(ir_sample_data, spectra)
dplyr::select(ir_sample_data, holocellulose) # drops ir class
```

`separate.ir` *Separate a character column in an* `ir` *object into multiple columns with a regular expression or numeric locations*

Description

Separate a character column in an `ir` object into multiple columns with a regular expression or numeric locations

Usage

```
separate.ir(
  data,
  col,
  into,
  sep = "[^[:alnum:]]+",
  remove = TRUE,
  convert = FALSE,
  extra = "warn",
  fill = "warn",
  ...
)
```

Arguments

`data`	An object of class `ir`.
`col`	Column name or position. This is passed to `tidyselect::vars_pull()`. This argument is passed by expression and supports quasiquotation (you can unquote column names or column positions).
`into`	Names of new variables to create as character vector. Use `NA` to omit the variable in the output.

sep	Separator between columns. If character, sep is interpreted as a regular expression. The default value is a regular expression that matches any sequence of non-alphanumeric values. If numeric, sep is interpreted as character positions to split at. Positive values start at 1 at the far-left of the string; negative value start at -1 at the far-right of the string. The length of sep should be one less than into.
remove	If TRUE, remove input column from output data frame.
convert	If TRUE, will run type.convert() with as.is = TRUE on new columns. This is useful if the component columns are integer, numeric or logical. NB: this will cause string "NA"s to be converted to NAs.
extra	If sep is a character vector, this controls what happens when there are too many pieces. There are three valid options: • "warn" (the default): emit a warning and drop extra values. • "drop": drop any extra values without a warning. • "merge": only splits at most length(into) times
fill	If sep is a character vector, this controls what happens when there are not enough pieces. There are three valid options: • "warn" (the default): emit a warning and fill from the right • "right": fill with missing values on the right • "left": fill with missing values on the left
...	Additional arguments passed on to methods.

Value

.data with separated columns. If the spectra column is dropped or invalidated (see ir_new_ir()), the ir class is dropped, else the object is of class ir.

Source

tidyr::separate()

See Also

Other tidyverse: arrange.ir(), distinct.ir(), extract.ir(), filter-joins, filter.ir(), group_by, mutate-joins, mutate, nest, pivot_longer.ir(), pivot_wider.ir(), rename, rowwise.ir(), select.ir(), separate_rows.ir(), slice, summarize, unite.ir()

Examples

```
## separate
ir_sample_data %>%
  tidyr::separate(
    col = "id_sample",  c("a", "b", "c")
  )
```

`separate_rows.ir` *Separate a collapsed column in an* `ir` *object into multiple rows*

Description

Separate a collapsed column in an `ir` object into multiple rows

Usage

```
separate_rows.ir(data, ..., sep = "[^[:alnum:].]+", convert = FALSE)
```

Arguments

`data`	An object of class `ir`.
`...`	<tidy-select> Columns to separate across multiple rows
`sep`	Separator delimiting collapsed values.
`convert`	If `TRUE` will automatically run `type.convert()` on the key column. This is useful if the column types are actually numeric, integer, or logical.

Value

`data` with a collapsed column separated into multiple rows. See `tidyr::separate_rows()`.

Source

`tidyr::separate_rows()`

See Also

Other tidyverse: `arrange.ir()`, `distinct.ir()`, `extract.ir()`, `filter-joins`, `filter.ir()`, `group_by`, `mutate-joins`, `mutate`, `nest`, `pivot_longer.ir()`, `pivot_wider.ir()`, `rename`, `rowwise.ir()`, `select.ir()`, `separate.ir()`, `slice`, `summarize`, `unite.ir()`

Examples

```
## separate_rows
ir_sample_data %>%
  tidyr::unite(
    col = content, holocellulose, klason_lignin
  ) %>%
  tidyr::separate_rows(
    col
  )
```

slice *Subset rows in* ir *objects using their positions*

Description

Subset rows in ir objects using their positions

Usage

```
slice.ir(.data, ..., .preserve = FALSE)

slice_sample.ir(.data, ..., n, prop, weight_by = NULL, replace = FALSE)
```

Arguments

.data	An object of class ir.
...	For slice(): <data-masking> Integer row values. Provide either positive values to keep, or negative values to drop. The values provided must be either all positive or all negative. Indices beyond the number of rows in the input are silently ignored. For slice_helpers(), these arguments are passed on to methods.
.preserve	Relevant when the .data input is grouped. If .preserve = FALSE (the default), the grouping structure is recalculated based on the resulting data, otherwise the grouping is kept as is.
n, prop	Provide either n, the number of rows, or prop, the proportion of rows to select. If neither are supplied, n = 1 will be used. If a negative value of n or prop is provided, the specified number or proportion of rows will be removed. If n is greater than the number of rows in the group (or prop > 1), the result will be silently truncated to the group size. If the proportion of a group size does not yield an integer number of rows, the absolute value of prop*nrow(.data) is rounded down.
weight_by	Sampling weights. This must evaluate to a vector of non-negative numbers the same length as the input. Weights are automatically standardised to sum to 1.
replace	Should sampling be performed with (TRUE) or without (FALSE, the default) replacement.

Value

.data with subsetted rows.

Source

dplyr::slice()

See Also

Other tidyverse: arrange.ir(), distinct.ir(), extract.ir(), filter-joins, filter.ir(), group_by, mutate-joins, mutate, nest, pivot_longer.ir(), pivot_wider.ir(), rename, rowwise.ir(), select.ir(), separate.ir(), separate_rows.ir(), summarize, unite.ir()

Examples

```
## slice
dplyr::slice(ir_sample_data, 1:5)
dplyr::slice_min(ir_sample_data, holocellulose, n = 3)
dplyr::slice_max(ir_sample_data, holocellulose, n = 3)
dplyr::slice_head(ir_sample_data, n = 5)
dplyr::slice_tail(ir_sample_data, n = 5)

## slice_sample
set.seed(234)
dplyr::slice_sample(ir_sample_data, n = 3)
```

---

subsetting *Subsetting* ir *objects*

Description

Subsetting ir objects

Usage

```
## S3 method for class 'ir'
x[i, j, ..., exact = TRUE]

## S3 method for class 'ir'
x$i

## S3 method for class 'ir'
x[[i, j, ..., exact = TRUE]]

## S3 replacement method for class 'ir'
x$i, j, ... <- value

## S3 replacement method for class 'ir'
i[j, ..., exact = TRUE] <- value

## S3 replacement method for class 'ir'
i[[j, ..., exact = TRUE]] <- value
```

Arguments

x	An object of class ir.
i, j	Row and column indices. If j is omitted, i is used as column index.
...	Ignored.
exact	Ignored, with a warning.
value	A value to store in a row, column, range or cell. Tibbles are stricter than data frames in what is accepted here.

Value

If the subsetting operation preserves a valid spectra column (see ir()), an object of class ir with accordingly subsetted rows or columns. Else a tibble::tbl_df() or vector.

Examples

```
# subsetting rows
ir_sample_data[1, ]
ir_sample_data[10:15, ]
ir_sample_data[ir_sample_data$sample_type == "office paper", ]

# subsetting columns
ir_sample_data[, "spectra"]
ir_sample_data[["spectra"]]
ir_sample_data$spectra

# not explicitly selecting the spectra column drops the ir class
class(ir_sample_data[, 1])
class(ir_sample_data[, "spectra"])

# subsetting values
ir_sample_data[, 1] # drops the ir class
ir_sample_data[, c("id_sample", "spectra")]
ir_sample_data$id_sample
ir_sample_data[[1, 1]]

# setting and replacing columns
x <- ir::ir_sample_data
x$a <- 3
x[, "a"] <- 4
x$sample_type <- "a"
x[[1]] <- rev(x[[1]])

# deleting the spectra column drops the ir class
x$spectra <- NULL
class(x)

# setting and replacing rows
x <- ir::ir_sample_data
x[1, ] <- x[2, ]
class(x)
```

```
# setting invalid values in the spectra column drops the ir class
x_replacement <- x[1, ]
x_replacement$spectra <- list(1)
x[1, ] <- x_replacement
class(x)

# setting and replacing values
x <- ir::ir_sample_data
x[[1, 1]] <- 100

# replacing an element in the spectra column by an invalid element drops the
# ir class attribute
x[[3, "spectra"]] <- list(1)
class(x)
```

`summarize` *Summarize each group in a* `ir` *object to fewer rows*

Description

Summarize each group in a `ir` object to fewer rows

Usage

```
summarize.ir(.data, ..., .groups = NULL)

summarise.ir(.data, ..., .groups = NULL)
```

Arguments

`.data`	An object of class `ir`.
`...`	<data-masking> Name-value pairs of summary functions. The name will be the name of the variable in the result. The value can be: • A vector of length 1, e.g. `min(x)`, `n()`, or `sum(is.na(y))`. • A vector of length `n`, e.g. `quantile()`. • A data frame, to add multiple columns from a single expression.
`.groups`	**[Experimental]** Grouping structure of the result. • "drop_last": dropping the last level of grouping. This was the only supported option before version 1.0.0. • "drop": All levels of grouping are dropped. • "keep": Same grouping structure as `.data`. • "rowwise": Each row is its own group.

When .groups is not specified, it is chosen based on the number of rows of the results:

- If all the results have 1 row, you get "drop_last".
- If the number of rows varies, you get "keep".

In addition, a message informs you of that choice, unless the result is ungrouped, the option "dplyr.summarise.inform" is set to FALSE, or when summarise() is called from a function in a package.

Value

.data with summarized columns. If the spectra column is dropped or invalidated (see ir_new_ir()), the ir class is dropped, else the object is of class ir.

Source

dplyr::summarize()

See Also

Other tidyverse: arrange.ir(), distinct.ir(), extract.ir(), filter-joins, filter.ir(), group_by, mutate-joins, mutate, nest, pivot_longer.ir(), pivot_wider.ir(), rename, rowwise.ir(), select.ir(), separate.ir(), separate_rows.ir(), slice, unite.ir()

Examples

```
## summarize

# select in each sample_type groups the first spectrum
ir_sample_data %>%
  dplyr::group_by(sample_type) %>%
  dplyr::summarize(spectra = spectra[[1]])
```

---

unite.ir *Unite multiple columns in an* ir *object into one by pasting strings together*

Description

Unite multiple columns in an ir object into one by pasting strings together

Usage

```
unite.ir(data, col, ..., sep = "_", remove = TRUE, na.rm = FALSE)
```

Arguments

data	An object of class ir.
col	The name of the new column, as a string or symbol. This argument is passed by expression and supports quasiquotation (you can unquote strings and symbols). The name is captured from the expression with rlang::ensym() (note that this kind of interface where symbols do not represent actual objects is now discouraged in the tidyverse; we support it here for backward compatibility).
...	<tidy-select> Columns to unite
sep	Separator to use between values.
remove	If TRUE, remove input columns from output data frame.
na.rm	If TRUE, missing values will be remove prior to uniting each value.

Value

.data with united columns. If the spectra column is dropped or invalidated (see ir_new_ir()), the ir class is dropped, else the object is of class ir.

Source

tidyr::unite()

See Also

Other tidyverse: arrange.ir(), distinct.ir(), extract.ir(), filter-joins, filter.ir(), group_by, mutate-joins, mutate, nest, pivot_longer.ir(), pivot_wider.ir(), rename, rowwise.ir(), select.ir(), separate.ir(), separate_rows.ir(), slice, summarize

Examples

```
## unite
ir_sample_data %>%
  tidyr::separate(
    "id_sample",  c("a", "b", "c")
  ) %>%
  tidyr::unite(id_sample, a, b, c)
```

Index