Package 'tsibble'

July 6, 2018

Type Package

Title Tidy Temporal Data Frames and Tools

Version 0.4.0

Date 2018-07-06

Description Provides a 'tbl_ts' class (the 'tsibble') to store and manage temporal-context data in a data-centric format, which is built on top of the 'tibble'. The 'tsibble' aims at easily manipulating and analysing temporal data, including counting and filling time gaps, aggregate over calendar periods, performing rolling window calculations, and etc.

License GPL (>= 3)

URL https://pkg.earo.me/tsibble

BugReports https://github.com/tidyverts/tsibble/issues

Depends R (>= 3.1.3)

Imports dplyr (>= 0.7.3), lubridate, pillar (>= 1.0.1), purrr (>= 0.2.3), Rcpp (>= 0.12.3), rlang (>= 0.2.0), tibble (>= 1.4.1), tidyr, tidyselect, timeDate

Suggests covr, ggplot2 (>= 2.2.0), hms, hts, knitr, nycflights13 (>= 1.0.0), rmarkdown, testthat

LinkingTo Rcpp (>= 0.12.0)

VignetteBuilder knitr

ByteCompile true

Encoding UTF-8

LazyData true

RoxygenNote 6.0.1

NeedsCompilation yes

Author Earo Wang [aut, cre] (<https://orcid.org/0000-0001-6448-5260>), Di Cook [aut, ths] (<https://orcid.org/0000-0002-3813-7155>), Rob Hyndman [aut, ths] (<https://orcid.org/0000-0002-2140-5352>), Mitchell O'Hara-Wild [ctb]

Maintainer Earo Wang <earo.wang@gmail.com>

Repository CRAN

Date/Publication 2018-07-06 12:50:03 UTC

R topics documented:

`tsibble-package` *tsibble: tidy temporal data frames and tools*

Description

The **tsibble** package provides a data class of `tbl_ts` to store and manage temporal data frames in a "tidy" form. A tsibble consists of a time index, key(s) and other measured variables in a data-centric format, which is built on top of the tibble.

Index

The time indices are no longer an attribute (for example, the `tsp` attribute in a `ts` object), but preserved as the essential component of the tsibble. A few index classes, such as `Date`, `POSIXct`, and `difftime`, forms the basis of the tsibble, with new additions yearweek, yearmonth, and yearquarter representing year-week, year-month, and year-quarter respectively. `zoo::yearmth` and `zoo::yearqtr` are also supported. For a `tbl_ts` of regular interval, a choice of index representation has to be made. For example, a monthly data should correspond to time index created by yearmonth or `zoo::yearmth`, instead of `Date` or `POSIXct`. Because months in a year ensures the regularity, 12 months every year. However, if using `Date`, a month contains days ranging from 28 to 31 days, which results in irregular time space. This is also applicable to year-week and year-quarter.

Since the **tibble** that underlies the **tsibble** only accepts a 1d atomic vector or a list, a `tbl_ts` doesn't accept `POSIXlt` and `timeDate` columns.

Key

Key variable(s) together with the index uniquely identifies each record. And key(s) also imposes the structure on a tsibble, which can be created via the id function as identifiers:

- None: an implicit variable `id()` resulting a univariate time series.
- A single variable: an explicit variable. For example, `data(pedestrian)` uses the `id(Sensor)` column as the key.
- Nested variables: a nesting of one variable under another. For example, `data(tourism)` contains two geographical locations: `Region` and `State`. `Region` is the lower level than `State` in Australia; in other words, `Region` is nested into `State`, which naturally forms a hierarchy. A vertical bar (`|`) is used to describe this nesting relationship, and thus `Region | State`. In theory, nesting can involve infinite levels, so is `tsibble`.

- Crossed variables: a crossing of one variable with another. For example, the geographical locations are crossed with the purpose of visiting (`Purpose`) in the `data(tourism)`. A comma (`,`) is used to indicate this crossing relationship. Nested and crossed variables can be combined, such as `data(tourism)` using `id(Region | State, Purpose)`.

These key variables describe the data structure.

Interval

The interval function returns the interval associated with the tsibble.

- Regular: the value and its time unit including "second", "minute", "hour", "day", "week", "month", "quarter", "year". An unrecognisable time interval is labelled as "unit".
- Irregular: `as_tsibble(regular = FALSE)` gives the irregular tsibble. It is marked with `!`.
- Unknown: if there is only one entry for each key variable, the interval cannot be determined (`?`).

Print options

The tsibble package fully utilises the `print` method from the tibble. Please refer to tibble::tibble-package to change display options.

Author(s)

Maintainer: Earo Wang <earo.wang@gmail.com> (0000-0001-6448-5260)

Authors:

- Di Cook (0000-0002-3813-7155) [thesis advisor]
- Rob Hyndman (0000-0002-2140-5352) [thesis advisor]

Other contributors:

- Mitchell O'Hara-Wild [contributor]

See Also

Useful links:

- https://pkg.earo.me/tsibble
- Report bugs at https://github.com/tidyverts/tsibble/issues

Examples

```
# create a tsibble w/o a key ----
tsbl1 <- tsibble(
  date = seq(as.Date("2017-01-01"), as.Date("2017-01-10"), by = 1),
  value = rnorm(10),
  key = id(), index = date
)
tsbl1
```

```
# create a tsibble with one key ----
tsbl2 <- tsibble(
  qtr = rep(yearquarter(seq(2010, 2012.25, by = 1 / 4)), 3),
  group = rep(c("x", "y", "z"), each = 10),
  value = rnorm(30),
  key = id(group), index = qtr
)
tsbl2
```

`arrange.tbl_ts` *Arrange rows by variables*

Description

Arrange rows by variables

Usage

```
## S3 method for class 'tbl_ts'
arrange(.data, ...)

## S3 method for class 'grouped_ts'
arrange(.data, ..., .by_group = FALSE)
```

Arguments

`.data`	A `tbl_ts`.
`...`	A set of unquoted variables, separated by commas.
`.by_group`	`TRUE` will sort first by grouping variables.

Details

If not arranging key and index in ascending order, a warning is likely to be issued.

See Also

dplyr::arrange

dplyr::arrange

as.ts.tbl_ts *Coerce a tsibble to a time series*

Description

Coerce a tsibble to a time series

Usage

```
## S3 method for class 'tbl_ts'
as.ts(x, value, frequency = NULL, fill = NA, ...)
```

Arguments

x	A tbl_ts object.
value	A measured variable of interest to be spread over columns, if multiple measures.
frequency	A smart frequency with the default NULL. If set, the preferred frequency is passed to ts().
fill	A value replaces missing values.
...	Ignored for the function.

Value

A ts object.

Examples

```
# a monthly series ----
x1 <- as_tsibble(AirPassengers)
as.ts(x1)

# equally spaced over trading days, not smart enough to guess frequency ----
x2 <- as_tsibble(EuStockMarkets)
head(as.ts(x2, frequency = 260))
```

as_tibble.tbl_ts *Coerce to a tibble or data frame*

Description

Coerce to a tibble or data frame

Usage

```
## S3 method for class 'tbl_ts'
as_tibble(x, ...)

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

Arguments

x	A tbl_ts.
...	Ignored.

Examples

```
as_tibble(pedestrian)

# a grouped tbl_ts -----
grped_ped <- pedestrian %>% group_by(Sensor)
as_tibble(grped_ped)
```

---

as_tsibble *Coerce to a tsibble object*

Description

Coerce to a tsibble object

Usage

```
as_tsibble(x, ...)

## S3 method for class 'tbl_df'
as_tsibble(x, key = id(), index, regular = TRUE,
  validate = TRUE, ...)

## S3 method for class 'tbl_ts'
as_tsibble(x, ...)

## S3 method for class 'data.frame'
as_tsibble(x, key = id(), index, regular = TRUE,
  validate = TRUE, ...)

## S3 method for class 'list'
as_tsibble(x, key = id(), index, regular = TRUE,
  validate = TRUE, ...)

## S3 method for class 'ts'
```

```
as_tsibble(x, tz = "UTC", ...)

## S3 method for class 'mts'
as_tsibble(x, tz = "UTC", gather = TRUE, ...)

## S3 method for class 'msts'
as_tsibble(x, tz = "UTC", gather = TRUE, ...)

## S3 method for class 'hts'
as_tsibble(x, tz = "UTC", ...)
```

Arguments

x	Other objects to be coerced to a tsibble (tbl_ts).
...	Other arguments passed on to individual methods.
key	Structural variable(s) that define unique time indices, used with the helper id. If a univariate time series (without an explicit key), simply call id(). See below for details.
index	A bare (or unquoted) variable to specify the time index variable.
regular	Regular time interval (TRUE) or irregular (FALSE). TRUE finds the greatest common divisor of positive time distances as the interval.
validate	TRUE suggests to verify that each key or each combination of key variables lead to unique time indices (i.e. a valid tsibble). It will also make sure that the nested variables are arranged from lower level to higher, if nested variables are passed to key. If you are sure that it's a valid input, specify FALSE to skip the checks.
tz	Time zone. May be useful when a ts object is more frequent than daily.
gather	TRUE gives a "long" data form, otherwise as "wide" as x.

Value

A tsibble object.

See Also

tsibble

Examples

```
# coerce tibble to tsibble w/o a key ----
tbl1 <- tibble::tibble(
  date = seq(as.Date("2017-01-01"), as.Date("2017-01-10"), by = 1),
  value = rnorm(10)
)
as_tsibble(tbl1)
# specify the index var
as_tsibble(tbl1, index = date)

# coerce tibble to tsibble with one key ----
```

```
# "date" is automatically considered as the index var, and "group" is the key
tbl2 <- tibble::tibble(
  mth = rep(yearmonth(seq(2017, 2017 + 9 / 12, by = 1 / 12)), 3),
  group = rep(c("x", "y", "z"), each = 10),
  value = rnorm(30)
)
as_tsibble(tbl2, key = id(group))
as_tsibble(tbl2, key = id(group), index = mth)

# coerce ts to tsibble
as_tsibble(AirPassengers)
as_tsibble(sunspot.year)
as_tsibble(sunspot.month)
as_tsibble(austres)

# coerce mts to tsibble
z <- ts(matrix(rnorm(300), 100, 3), start = c(1961, 1), frequency = 12)
as_tsibble(z)
as_tsibble(z, gather = FALSE)

# coerce hts from the "hts" package to tsibble
if (!requireNamespace("hts", quietly = TRUE)) {
  stop("Please install the hts package to run these following examples.")
}
as_tsibble(hts::htseg1)
as_tsibble(hts::htseg2)
```

`build_tsibble` *Construct a tsibble object*

Description

A relatively more controllable function to create a `tbl_ts` object. It is useful for creating a `tbl_ts` internally inside a function, and it allows users to determine if the time needs ordering and the interval needs calculating.

Usage

```
build_tsibble(x, key, index, index2, groups = id(), regular = TRUE,
  validate = TRUE, ordered = NULL, interval = NULL)
```

Arguments

`x`	A `data.frame`, `tbl_df`, `tbl_ts`, or other tabular objects.
`key`	Structural variable(s) that define unique time indices, used with the helper id. If a univariate time series (without an explicit key), simply call `id()`. See below for details.
`index`	A bare (or unquoted) variable to specify the time index variable.

index2	A candidate of index to update the index to a new one when index_by. By default, it's identical to index.
groups	Grouping variable(s) when group_by.tbl_ts.
regular	Regular time interval (TRUE) or irregular (FALSE). TRUE finds the greatest common divisor of positive time distances as the interval.
validate	TRUE suggests to verify that each key or each combination of key variables lead to unique time indices (i.e. a valid tsibble). It will also make sure that the nested variables are arranged from lower level to higher, if nested variables are passed to key. If you are sure that it's a valid input, specify FALSE to skip the checks.
ordered	The default of NULL arranges the key variable(s) first and then index from past to future. TRUE suggests to skip the ordering as x in the correct order. FALSE also skips the ordering but gives a warning instead.
interval	NULL computes the interval. Use the specified interval as is, if an class of interval is supplied.

Examples

```
# Prepare `pedestrian` to use a new index `Date` ----
pedestrian %>%
  build_tsibble(
    key = key(.), index = !! index(.), index2 = Date, interval = interval(.)
  )
```

---

case_na *A thin wrapper of* dplyr::case_when() *if there are* NA*s*

Description

A thin wrapper of dplyr::case_when() if there are NAs

Usage

```
case_na(formula)
```

Arguments

formula	A two-sided formula. The LHS expects a vector containing NA, and the RHS gives the replacement value.

See Also

dplyr::case_when

Examples

```
x <- rnorm(10)
x[c(3, 7)] <- NA_real_
case_na(x ~ 10)
case_na(x ~ mean(x, na.rm = TRUE))
```

`count_gaps` *Count implicit gaps*

Description

`count_gaps()` counts gaps for a tsibble; `gaps()` find where the gaps in `x` with respect to `y`.

Usage

```
count_gaps(.data, ...)

## S3 method for class 'tbl_ts'
count_gaps(.data, ...)

## S3 method for class 'grouped_ts'
count_gaps(.data, .full = FALSE, ...)

gaps(x, y)
```

Arguments

`.data`	A `tbl_ts`.
`...`	Other arguments passed on to individual methods.
`.full`	`FALSE` to find gaps for each group within its own period. `TRUE` to find gaps over the entire time span of the data.
`x, y`	A vector of numbers, dates, or date-times. The length of `y` must be greater than the length of `x`.

Value

A tibble contains:

- the "key" of the `tbl_ts`
- "from": the starting time point of the gap
- "end": the ending time point of the gap
- "n": the implicit missing observations during the time period

See Also

fill_na

Examples

```
# Implicit missing time without group_by ----
# All the sensors have 2 common missing time points in the data
count_gaps(pedestrian)
# Time gaps for each sensor per month ----
pedestrian %>%
  index_by(yrmth = yearmonth(Date)) %>%
  group_by(Sensor) %>%
  count_gaps()
# Time gaps for each sensor ----
ped_gaps <- pedestrian %>%
  group_by(Sensor) %>%
  count_gaps(.full = TRUE)
if (!requireNamespace("ggplot2", quietly = TRUE)) {
  stop("Please install the ggplot2 package to run these following examples.")
}
library(ggplot2)
ggplot(ped_gaps, aes(colour = Sensor)) +
  geom_linerange(aes(x = Sensor, ymin = from, ymax = to)) +
  geom_point(aes(x = Sensor, y = from)) +
  geom_point(aes(x = Sensor, y = to)) +
  coord_flip() +
  theme(legend.position = "bottom")
# Vectors ----
gaps(x = c(1:3, 5:6, 9:10), y = 1:10)
```

---

difference *Lagged differences*

Description

Lagged differences

Usage

```
difference(x, lag = 1, differences = 1, default = NA, order_by = NULL)
```

Arguments

x	A numeric vector.
lag	An positive integer indicating which lag to use.
differences	An positive integer indicating the order of the difference.
default	Value used for non-existent rows, defaults to NA.
order_by	Override the default ordering to use another vector.

Value

A numeric vector of the same length as x.

See Also

dplyr::lead and dplyr::lag

Examples

```
# examples from base
difference(1:10, 2)
difference(1:10, 2, 2)
x <- cumsum(cumsum(1:10))
difference(x, lag = 2)
difference(x, differences = 2)
# Use order_by if data not already ordered (example from dplyr)
tsbl <- tsibble(year = 2000:2005, value = (0:5) ^ 2, index = year)
scrambled <- tsbl %>% slice(sample(nrow(tsbl)))

wrong <- mutate(scrambled, diff = difference(value))
arrange(wrong, year)

right <- mutate(scrambled, diff = difference(value, order_by = year))
arrange(right, year)
```

---

fill_na *Turn implicit missing values into explicit missing values*

Description

Turn implicit missing values into explicit missing values

Usage

```
fill_na(.data, ...)

## S3 method for class 'tbl_ts'
fill_na(.data, ..., .full = FALSE)
```

Arguments

.data	A data frame.
...	A set of name-value pairs. The values will replace existing explicit missing values by variable, otherwise NA. The replacement values must be of the same type as the original one. If using a function to fill the NA, please make sure that na.rm = TRUE is switched on.
.full	FALSE to insert NA for each key within its own period. TRUE to fill NA over the entire time span of the data (a.k.a. fully balanced panel).

See Also

count_gaps, case_na, tidyr::fill, tidyr::replace_na

Examples

```
harvest <- tsibble(
  year = c(2010, 2011, 2013, 2011, 2012, 2014),
  fruit = rep(c("kiwi", "cherry"), each = 3),
  kilo = sample(1:10, size = 6),
  key = id(fruit), index = year
)

# leave NA as is ----
fill_na(harvest, .full = TRUE)
full_harvest <- fill_na(harvest, .full = FALSE)
full_harvest

# use fill() to fill `NA` by previous/next entry
full_harvest %>%
  group_by(fruit) %>%
  tidyr::fill(kilo, .direction = "down")

# replace NA with a specific value ----
harvest %>%
  fill_na(kilo = 0L)

# replace NA using a function by variable ----
# enable `na.rm = TRUE` when necessary ----
harvest %>%
  fill_na(kilo = sum(kilo, na.rm = TRUE))

# replace NA using a function for each group ----
harvest %>%
  group_by(fruit) %>%
  fill_na(kilo = sum(kilo, na.rm = TRUE))

# replace NA ----
pedestrian %>%
  group_by(Sensor) %>%
  fill_na(
    Date = lubridate::as_date(Date_Time),
    Time = lubridate::hour(Date_Time),
    Count = as.integer(median(Count, na.rm = TRUE))
  )
```

`filter.tbl_ts` *Return rows with matching conditions*

Description

Return rows with matching conditions

Usage

```
## S3 method for class 'tbl_ts'
filter(.data, ...)
```

Arguments

.data	A tbl_ts.
...	Logical predicates defined in terms of the variables.

See Also

dplyr::filter

---

find_duplicates *Find duplication of key and index variables*

Description

Find which row has duplicated key and index elements

Usage

```
find_duplicates(data, key = id(), index, fromLast = FALSE)
```

Arguments

data	A tbl_ts object.
key	Structural variable(s) that define unique time indices, used with the helper id. If a univariate time series (without an explicit key), simply call id().
index	A bare (or unquoted) variable to specify the time index variable.
fromLast	TRUE does the duplication check from the last of identical elements.

Value

A logical vector of the same length as the row number of data

`gather.tbl_ts` *Gather columns into key-value pairs.*

Description

Gather columns into key-value pairs.

Usage

```
## S3 method for class 'tbl_ts'
gather(data, key = "key", value = "value", ...,
  na.rm = FALSE, convert = FALSE, factor_key = FALSE)
```

Arguments

`data`	A `tbl_ts`.
`key`	Names of new key and value columns, as strings or symbols. This argument is passed by expression and supports quasiquotation (you can unquote strings and symbols). The name is captured from the expression with `rlang::quo_name()` (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).
`value`	Names of new key and value columns, as strings or symbols. This argument is passed by expression and supports quasiquotation (you can unquote strings and symbols). The name is captured from the expression with `rlang::quo_name()` (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).
`...`	A selection of columns. If empty, all variables are selected. You can supply bare variable names, select all variables between x and z with `x:z`, exclude y with `-y`. For more options, see the `dplyr::select()` documentation. See also the section on selection rules below.
`na.rm`	If `TRUE`, will remove rows from output where the value column in `NA`.
`convert`	If `TRUE` will automatically run `type.convert()` on the key column. This is useful if the column names are actually numeric, integer, or logical.
`factor_key`	If `FALSE`, the default, the key values will be stored as a character vector. If `TRUE`, will be stored as a factor, which preserves the original ordering of the columns.

See Also

tidyr::gather

Examples

```
# example from tidyr
stocks <- tsibble(
  time = as.Date('2009-01-01') + 0:9,
  X = rnorm(10, 0, 1),
  Y = rnorm(10, 0, 2),
  Z = rnorm(10, 0, 4)
)
stocks %>% gather(stock, price, -time)
```

`group_by.tbl_ts` *Group by one or more variables*

Description

Group by one or more variables

Usage

```
## S3 method for class 'tbl_ts'
group_by(.data, ..., add = FALSE)

## S3 method for class 'grouped_ts'
ungroup(x, ...)
```

Arguments

`.data`	A tsibble.
`...`	Variables to group by. All tbls accept variable names. Some tbls will accept functions of variables. Duplicated groups will be silently dropped.
`add`	When `add = FALSE`, the default, `group_by()` will override existing groups. To add to the existing groups, use `add = TRUE`.
`x`	A (grouped) tsibble.

See Also

dplyr::group_by

dplyr::ungroup

Examples

```
data(tourism)
tourism %>%
  group_by(Region, State) %>%
  summarise(geo_trips = sum(Trips))
```

guess_frequency *Guess a time frequency from other index objects*

Description

A possible frequency passed to the `ts()` function

Usage

```
guess_frequency(x)
```

Arguments

x	An index object including "yearmonth", "yearquarter", "Date" and others.

Details

If a series of observations are collected more frequently than weekly, it is more likely to have multiple seasonalities. This function returns a frequency value at its nearest ceiling time resolution. For example, hourly data would have daily, weekly and annual frequencies of 24, 168 and 8766 respectively, and hence it gives 24.

References

https://robjhyndman.com/hyndsight/seasonal-periods/

Examples

```
guess_frequency(yearquarter(seq(2016, 2018, by = 1 / 4)))
guess_frequency(yearmonth(seq(2016, 2018, by = 1 / 12)))
guess_frequency(seq(as.Date("2017-01-01"), as.Date("2017-01-31"), by = 1))
guess_frequency(seq(
  as.POSIXct("2017-01-01 00:00"), as.POSIXct("2017-01-10 23:00"),
  by = "1 hour"
))
```

holiday_aus *Australian national and state-based public holiday*

Description

Australian national and state-based public holiday

Usage

```
holiday_aus(year, state = "national")
```

Arguments

year	A vector of integer(s) indicating year(s).
state	A state in Australia including "ACT", "NSW", "NT", "QLD", "SA", "TAS", "VIC", "WA", as well as "national".

Details

Not documented public holidays:

- AFL public holidays for Victoria
- Queen's Birthday for Western Australia
- Royal Queensland Show for Queensland, which is for Brisbane only

Value

A tibble consisting of `holiday` labels and their associated dates in the year(s).

References

Public holidays

Examples

```
holiday_aus(2016, state = "VIC")
holiday_aus(2013:2016, state = "ACT")
```

---

id *Identifier to construct structural variables*

Description

Impose a structure to a tsibble

Usage

```
id(...)
```

Arguments

...	Variables passed to tsibble()/as_tsibble().

See Also

tsibble, as_tsibble

index_by *Group by time index*

Description

index_by() is the counterpart of group_by() in temporal context, but it only groups the time index. It adds a new column and then group it. The following operation is applied to each group of the index, similar to group_by() but dealing with index only. index_by() + summarise() will update the grouping index variable to be the new index. Use ungroup() or index_by() with no arguments to remove the index grouping vars.

Usage

```
index_by(.data, ...)
```

Arguments

.data	A tbl_ts.
...	• A single name-value pair of expression: a new index on LHS and the current index on RHS • An existing variable to be used as index The index functions that can be used, but not limited: • lubridate::year: yearly aggregation • yearquarter: quarterly aggregation • yearmonth: monthly aggregation • yearweek: weekly aggregation • as.Date or lubridate::as_date: daily aggregation • lubridate::ceiling_date, lubridate::floor_date, or lubridate::round_date: sub-daily aggregation • other index functions from other packages

Details

- A index_by()-ed tsibble is indicated by @ in the "Groups" when displaying on the screen.
- Time index will not be collapsed by summarise.tbl_ts.
- The scoped variants of summarise() only operate on the non-key and non-index variables.

Examples

```
# Monthly counts across sensors ----
monthly_ped <- pedestrian %>%
  group_by(Sensor) %>%
  index_by(Year_Month = yearmonth(Date_Time)) %>%
  summarise(
    Max_Count = max(Count),
    Min_Count = min(Count)
```

```
  )
monthly_ped
index(monthly_ped)

# Using existing variable ----
pedestrian %>%
  group_by(Sensor) %>%
  index_by(Date) %>%
  summarise(
    Max_Count = max(Count),
    Min_Count = min(Count)
  )

# Annual trips by Region and State ----
tourism %>%
  index_by(Year = lubridate::year(Quarter)) %>%
  group_by(Region, State) %>%
  summarise(Total = sum(Trips))
```

`index_valid` *Extensible index type to tsibble*

Description

S3 method to add an index type support for a tsibble.

Usage

```
index_valid(x)
```

Arguments

x	An object of index type that the tsibble supports.

Details

This method is primarily used for adding an index type support in as_tsibble.

Value

TRUE/FALSE or NA (unsure)

See Also

pull_interval for obtaining interval for regularly spaced time.

Examples

```
index_valid(seq(as.Date("2017-01-01"), as.Date("2017-01-10"), by = 1))
```

interval *Return index and interval from a tsibble*

Description

Return index and interval from a tsibble

Usage

```
interval(x)

index(x)

index2(x)
```

Arguments

x	A tsibble object.

Examples

```
data(pedestrian)
index(pedestrian)
interval(pedestrian)
```

is_regular *is_regular checks if a tsibble is spaced at regular time or not; is_ordered checks if a tsibble is ordered by key and index.*

Description

is_regular checks if a tsibble is spaced at regular time or not; is_ordered checks if a tsibble is ordered by key and index.

Usage

```
is_regular(x)

is.regular(x)

is_ordered(x)
```

Arguments

x	A tsibble object.

Examples

```
data(pedestrian)
is_regular(pedestrian)
is_ordered(pedestrian)
```

`is_tsibble` *Test if the object is a tsibble*

Description

Test if the object is a tsibble

Usage

```
is_tsibble(x)

is_grouped_ts(x)
```

Arguments

x	An object.

Value

TRUE if the object inherits from the tbl_ts class.

Examples

```
# A tibble is not a tsibble ----
tbl <- tibble::tibble(
  date = seq(as.Date("2017-10-01"), as.Date("2017-10-31"), by = 1),
  value = rnorm(31)
)
is_tsibble(tbl)

# A tsibble ----
tsbl <- as_tsibble(tbl, index = date)
is_tsibble(tsbl)
```

key *Return key variables*

Description

key() returns a list of symbols; key_vars() gives a character vector.

Usage

```
key(x)

key_vars(x)

unkey(x)
```

Arguments

x	A data frame.

Examples

```
# A single key for pedestrian data ----
key(pedestrian)
key_vars(pedestrian)

# Nested and crossed keys for tourism data ----
key(tourism)
key_vars(tourism)
# unkey() only works for a tsibble with 1 key size ----
sx <- pedestrian %>%
  filter(Sensor == "Southern Cross Station")
unkey(sx)
```

key_size *Compute sizes of key variables*

Description

Compute sizes of key variables

Usage

```
key_size(x)

n_keys(x)

key_indices(x)
```

Arguments

x	A data frame.

Examples

```
key_size(pedestrian)
n_keys(pedestrian)
key_indices(pedestrian)
```

---

key_sum *Summary of key variables*

Description

Summary of key variables

Usage

```
key_sum(x)
```

Arguments

x	An object that contains "key".

Examples

```
key_sum(pedestrian)
```

---

key_update *Change/update key variables for a given* tbl_ts

Description

Change/update key variables for a given `tbl_ts`

Usage

```
key_update(.data, ..., validate = TRUE)
```

Arguments

.data	A tbl_ts.
...	Expressions used to construct the key: • unspecified: drop every single variable from the old key. • | and , for nesting and crossing factors.
validate	TRUE suggests to verify that each key or each combination of key variables lead to unique time indices (i.e. a valid tsibble). It will also make sure that the nested variables are arranged from lower level to higher, if nested variables are passed to key. If you are sure that it's a valid input, specify FALSE to skip the checks.

Examples

```
# tourism could be identified by Region and Purpose ----
tourism %>%
  key_update(Region, Purpose)
```

---

measured_vars *Return measured variables*

Description

Return measured variables

Usage

```
measured_vars(x)
```

Arguments

x	A tbl_ts.

Examples

```
measured_vars(pedestrian)
measured_vars(tourism)
```

mutate.tbl_ts *Add new variables*

Description

mutate() adds new variables; transmute() keeps the newly created variables along with index and keys;

Usage

```
## S3 method for class 'tbl_ts'
mutate(.data, ..., .drop = FALSE)

## S3 method for class 'tbl_ts'
transmute(.data, ..., .drop = FALSE)
```

Arguments

.data	A tsibble.
...	Name-value pairs of expressions.
.drop	FALSE returns a tsibble object as the input. TRUE drops a tsibble and returns a tibble.

Details

These column-wise verbs from dplyr have an additional argument of .drop = FALSE for tsibble. If any key variable is changed, it will validate whether it's a tsibble internally. Turning .drop = TRUE converts to a tibble first and then do the operations.

- summarise() will not collapse on the index variable.

See Also

dplyr::mutate

dplyr::transmute

nest.tbl_ts *Nest repeated values in a list-variable.*

Description

Nest repeated values in a list-variable.

Usage

```
## S3 method for class 'tbl_ts'
nest(data, ..., .key = "data")
```

Arguments

data	A tbl_ts.
...	A selection of columns. If empty, all variables are selected. You can supply bare variable names, select all variables between x and z with x:z, exclude y with -y. For more options, see the dplyr::select() documentation. See also the section on selection rules below.
.key	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::quo_name() (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).

Value

A tibble containing a list column of tbl_ts.

See Also

tidyr::nest, unnest.lst_ts for the inverse operation.

Examples

```
pedestrian %>%
  nest(-Sensor)
pedestrian %>%
  group_by(Sensor) %>%
  nest()
```

---

pedestrian *Pedestrian counts in the city of Melbourne*

Description

A dataset containing the hourly pedestrian counts from 2015-01-01 to 2016-12-31 at 4 sensors in the city of Melbourne.

Usage

```
pedestrian
```

Format

A tsibble with 66,071 rows and 5 variables:

- **Sensor**: Sensor names (key)
- **Date_Time**: Date time when the pedestrian counts are recorded (index)
- **Date**: Date when the pedestrian counts are recorded
- **Time**: Hour associated with Date_Time
- **Counts**: Hourly pedestrian counts

References

Melbourne Open Data Portal

Examples

```
data(pedestrian)
# make implicit missingness to be explicit ----
pedestrian %>% fill_na()
# compute daily maximum counts across sensors ----
pedestrian %>%
  group_by(Sensor) %>%
  index_by(Date) %>% # group by Date and use it as new index
  summarise(MaxC = max(Count))
```

---

pull_interval *Extract time interval from a vector*

Description

Assuming regularly spaced time, the `pull_interval()` returns a list of time components as the "interval" class; the `time_unit()` returns the value of time units.

Usage

```
pull_interval(x)

time_unit(x)
```

Arguments

x	A vector of `POSIXt`, `Date`, `yearmonth`, `yearquarter`, `difftime`, `hms`, `integer`, `numeric`.

Details

The `pull_interval()` and `time_unit()` make a tsibble extensible to support custom time index.

Value

`pull_interval()`: an "interval" class (a list) includes "year", "quarter", "month", #' "week", "day", "hour", "minute", "second", "unit", and other self-defined interval.

Examples

```
x <- seq(as.Date("2017-10-01"), as.Date("2017-10-31"), by = 3)
pull_interval(x)
# at two months interval ----
x <- yearmonth(seq(2016, 2018, by = 0.5))
time_unit(x)
```

`select.tbl_ts` *Select/rename variables by name*

Description

Select/rename variables by name

Usage

```
## S3 method for class 'tbl_ts'
select(.data, ..., .drop = FALSE)

## S3 method for class 'tbl_ts'
rename(.data, ...)
```

Arguments

`.data`	A tsibble.
`...`	Unquoted variable names separated by commas. `rename()` requires named arguments.
`.drop`	`FALSE` returns a tsibble object as the input. `TRUE` drops a tsibble and returns a tibble.

Details

These column-wise verbs from dplyr have an additional argument of `.drop = FALSE` for tsibble. The index variable cannot be dropped for a tsibble. If any key variable is changed, it will validate whether it's a tsibble internally. Turning `.drop = TRUE` converts to a tibble first and then do the operations.

See Also

dplyr::select

dplyr::rename

slice.tbl_ts *Selects rows by position*

Description

Selects rows by position

Usage

```
## S3 method for class 'tbl_ts'
slice(.data, ...)
```

Arguments

`.data`	A `tbl_ts`.
`...`	Unique integers of row numbers to be selected.

Details

If row numbers are not in ascending order, a warning is likely to be issued.

See Also

dplyr::slice

slide *Sliding window calculation*

Description

Rolling window with overlapping observations:

- `slide()` always returns a list.
- `slide_lgl()`, `slide_int()`, `slide_dbl()`, `slide_chr()` use the same arguments as `slide()`, but return vectors of the corresponding type.
- `slide_dfr()` `slide_dfc()` return data frames using row-binding & column-binding.

Usage

```
slide(.x, .f, ..., .size = 1, .fill = NA, .partial = FALSE)

slide_dfr(.x, .f, ..., .size = 1, .fill = NA, .partial = FALSE,
  .id = NULL)

slide_dfc(.x, .f, ..., .size = 1, .fill = NA, .partial = FALSE)
```

Arguments

.x	An object to slide over.
.f	A function, formula, or atomic vector. If a **function**, it is used as is. If a **formula**, e.g. `~ .x + 2`, it is converted to a function. There are three ways to refer to the arguments: • For a single argument function, use `.` • For a two argument function, use `.x` and `.y` • For more arguments, use `..1`, `..2`, `..3` etc This syntax allows you to create very compact anonymous functions. If **character vector**, **numeric vector**, or **list**, it is converted to an extractor function. Character vectors index by name and numeric vectors index by position; use a list to index by position and name at different levels. Within a list, wrap strings in `get-attr()` to extract named attributes. If a component is not present, the value of `.default` will be returned.
...	Additional arguments passed on to `.f`.
.size	An integer for window size. If positive, moving forward from left to right; if negative, moving backward (from right to left).
.fill	A single value or data frame to replace `NA`.
.partial	if `TRUE`, partial sliding.
.id	If not `NULL` a variable with this name will be created giving either the name or the index of the data frame.

Details

The `slide()` function attempts to tackle more general problems using the purrr-like syntax. For some specialist functions like `mean` and `sum`, you may like to check out for RcppRoll for faster performance.

See Also

- slide2, pslide
- tile for tiling window without overlapping observations
- stretch for expanding more observations

Examples

```
.x <- 1:5
.lst <- list(x = .x, y = 6:10, z = 11:15)
slide_dbl(.x, mean, .size = 2)
slide_lgl(.x, ~ mean(.) > 2, .size = 2)
slide(.lst, ~ ., .size = 2)
```

slide2 *Sliding window calculation over multiple inputs simultaneously*

Description

Rolling window with overlapping observations:

- `slide2()` and `pslide()` always returns a list.
- `slide2_lgl()`, `slide2_int()`, `slide2_dbl()`, `slide2_chr()` use the same arguments as `slide2()`, but return vectors of the corresponding type.
- `slide2_dfr()` `slide2_dfc()` return data frames using row-binding & column-binding.

Usage

```
slide2(.x, .y, .f, ..., .size = 1, .fill = NA, .partial = FALSE)

slide2_dfr(.x, .y, .f, ..., .size = 1, .fill = NA, .partial = FALSE,
  .id = NULL)

slide2_dfc(.x, .y, .f, ..., .size = 1, .fill = NA, .partial = FALSE)

pslide(.l, .f, ..., .size = 1, .fill = NA, .partial = FALSE)

pslide_dfr(.l, .f, ..., .size = 1, .fill = NA, .partial = FALSE,
  .id = NULL)

pslide_dfc(.l, .f, ..., .size = 1, .fill = NA, .partial = FALSE)
```

Arguments

`.x, .y`	Objects to slide over simultaneously.
`.f`	A function, formula, or atomic vector. If a **function**, it is used as is. If a **formula**, e.g. `~ .x + 2`, it is converted to a function. There are three ways to refer to the arguments: • For a single argument function, use `.` • For a two argument function, use `.x` and `.y` • For more arguments, use `..1`, `..2`, `..3` etc This syntax allows you to create very compact anonymous functions. If **character vector**, **numeric vector**, or **list**, it is converted to an extractor function. Character vectors index by name and numeric vectors index by position; use a list to index by position and name at different levels. Within a list, wrap strings in `get-attr()` to extract named attributes. If a component is not present, the value of `.default` will be returned.
`...`	Additional arguments passed on to `.f`.

.size	An integer for window size. If positive, moving forward from left to right; if negative, moving backward (from right to left).
.fill	A single value or data frame to replace NA.
.partial	if TRUE, partial sliding.
.id	If not NULL a variable with this name will be created giving either the name or the index of the data frame.
.l	A list of lists. The length of .l determines the number of arguments that .f will be called with. List names will be used if present.

See Also

- slide
- tile2 for tiling window without overlapping observations
- stretch2 for expanding more observations

Examples

```
.x <- 1:5
.y <- 6:10
.z <- 11:15
.lst <- list(x = .x, y = .y, z = .z)
.df <- as.data.frame(.lst)
slide2(.x, .y, sum, .size = 2)
slide2(.lst, .lst, ~ ., .size = 2)
slide2(.df, .df, ~ ., .size = 2)
pslide(.lst, ~ ., size = 1)
pslide(list(.lst, .lst), ~ ., .size = 2)
## window over 2 months
pedestrian %>%
  filter(Sensor == "Southern Cross Station") %>%
  index_by(yrmth = yearmonth(Date_Time)) %>%
  nest(-yrmth) %>%
  mutate(ma = slide_dbl(data, ~ mean(do.call(rbind, .)$Count), .size = 2))
# row-oriented workflow
## Not run:
my_diag <- function(...) {
  data <- list(...)
  fit <- lm(data$Count ~ data$Time)
  tibble::tibble(fitted = fitted(fit), resid = residuals(fit))
}
pedestrian %>%
  filter(Date <= as.Date("2015-01-31")) %>%
  nest(-Sensor) %>%
  mutate(diag = purrr::map(data, ~ pslide_dfr(., my_diag, .size = 48)))

## End(Not run)
```

slider *Splits the input to a list according to the rolling window size.*

Description

Splits the input to a list according to the rolling window size.

Usage

```
slider(.x, .size = 1, .fill = NA, .partial = FALSE)

pslider(..., .size = 1, .fill = NA, .partial = FALSE)
```

Arguments

.x	An object to slide over.
.size	An integer for window size. If positive, moving forward from left to right; if negative, moving backward (from right to left).
.fill	A single value or data frame to replace NA.
.partial	if TRUE, split to partial set (FALSE ignores specified .fill).
...	Multiple objects to be split in parallel.
x	An objects to be split.

Examples

```
.x <- 1:5
.y <- 6:10
.z <- 11:15
.lst <- list(x = .x, y = .y, z = .z)
.df <- as.data.frame(.lst)

slider(.x, .size = 2)
slider(.lst, .size = 2)
pslider(list(.x, .y), list(.y))
slider(.df, .size = 2)
pslider(.df, .df, .size = 2)
```

split_by *Split a data frame into a list of subsets by variables*

Description

Split a data frame into a list of subsets by variables

Usage

```
split_by(x, ...)
```

Arguments

x	A data frame.
...	A list of unquoted variables, separated by commas, to split a dataset.

Examples

```
pedestrian %>%
  split_by(Sensor)
```

spread.tbl_ts *Spread a key-value pair across multiple columns.*

Description

Spread a key-value pair across multiple columns.

Usage

```
## S3 method for class 'tbl_ts'
spread(data, key, value, fill = NA, convert = FALSE,
  drop = TRUE, sep = NULL)
```

Arguments

data	A tbl_ts.
key	Column names or positions. This is passed to tidyselect::vars_pull(). These arguments are passed by expression and support quasiquotation (you can unquote column names or column positions).
value	Column names or positions. This is passed to tidyselect::vars_pull(). These arguments are passed by expression and support quasiquotation (you can unquote column names or column positions).

fill	If set, missing values will be replaced with this value. Note that there are two types of missingness in the input: explicit missing values (i.e. NA), and implicit missings, rows that simply aren't present. Both types of missing value will be replaced by fill.
convert	If TRUE, type.convert() with asis = TRUE will be run on each of the new columns. This is useful if the value column was a mix of variables that was coerced to a string. If the class of the value column was factor or date, note that will not be true of the new columns that are produced, which are coerced to character before type conversion.
drop	If FALSE, will keep factor levels that don't appear in the data, filling in missing combinations with fill.
sep	If NULL, the column names will be taken from the values of key variable. If non-NULL, the column names will be given by "<key_name><sep><key_value>".

See Also

tidyr::spread

Examples

```
# example from tidyr
stocks <- tsibble(
  time = as.Date('2009-01-01') + 0:9,
  X = rnorm(10, 0, 1),
  Y = rnorm(10, 0, 2),
  Z = rnorm(10, 0, 4)
)
stocksm <- stocks %>% gather(stock, price, -time)
stocksm %>% spread(stock, price)
```

---

stretch *Stretching window calculation*

Description

Fixing an initial window and expanding more observations:

- stretch() always returns a list.
- stretch_lgl(), stretch_int(), stretch_dbl(), stretch_chr() use the same arguments as stretch(), but return vectors of the corresponding type.
- stretch_dfr() stretch_dfc() return data frames using row-binding & column-binding.

Usage

```
stretch(.x, .f, ..., .size = 1, .init = 1)

stretch_dfr(.x, .f, ..., .size = 1, .init = 1, .id = NULL)

stretch_dfc(.x, .f, ..., .size = 1, .init = 1)
```

Arguments

`.x`	An object to slide over.
`.f`	A function, formula, or atomic vector. If a **function**, it is used as is. If a **formula**, e.g. `~ .x + 2`, it is converted to a function. There are three ways to refer to the arguments: • For a single argument function, use `.` • For a two argument function, use `.x` and `.y` • For more arguments, use `..1`, `..2`, `..3` etc This syntax allows you to create very compact anonymous functions. If **character vector**, **numeric vector**, or **list**, it is converted to an extractor function. Character vectors index by name and numeric vectors index by position; use a list to index by position and name at different levels. Within a list, wrap strings in `get-attr()` to extract named attributes. If a component is not present, the value of `.default` will be returned.
`...`	Additional arguments passed on to `.f`.
`.size`	An integer for window size. If positive, moving forward from left to right; if negative, moving backward (from right to left).
`.init`	A positive integer for an initial window size.
`.id`	If not `NULL` a variable with this name will be created giving either the name or the index of the data frame.

See Also

- stretch2, pstretch
- slide for sliding window with overlapping observations
- tile for tiling window without overlapping observations

Examples

```
.x <- 1:5
.lst <- list(x = .x, y = 6:10, z = 11:15)
stretch_dbl(.x, mean, .size = 2)
stretch_lgl(.x, ~ mean(.) > 2, .size = 2)
stretch(.lst, ~ ., .size = 2)
```

`stretch2` *Stretching window calculation over multiple simultaneously*

Description

Fixing an initial window and expanding more observations:

- `stretch2()` and `pstretch()` always returns a list.
- `stretch2_lgl()`, `stretch2_int()`, `stretch2_dbl()`, `stretch2_chr()` use the same arguments as `stretch2()`, but return vectors of the corresponding type.
- `stretch2_dfr()` `stretch2_dfc()` return data frames using row-binding & column-binding.

Usage

```
stretch2(.x, .y, .f, ..., .size = 1, .init = 1)

stretch2_dfr(.x, .y, .f, ..., .size = 1, .init = 1, .id = NULL)

stretch2_dfc(.x, .y, .f, ..., .size = 1, .init = 1)

pstretch(.l, .f, ..., .size = 1, .init = 1)

pstretch_dfr(.l, .f, ..., .size = 1, .init = 1, .id = NULL)

pstretch_dfc(.l, .f, ..., .size = 1, .init = 1)
```

Arguments

`.x`	Objects to slide over simultaneously.
`.y`	Objects to slide over simultaneously.
`.f`	A function, formula, or atomic vector. If a **function**, it is used as is. If a **formula**, e.g. `~ .x + 2`, it is converted to a function. There are three ways to refer to the arguments: • For a single argument function, use `.` • For a two argument function, use `.x` and `.y` • For more arguments, use `..1`, `..2`, `..3` etc This syntax allows you to create very compact anonymous functions. If **character vector**, **numeric vector**, or **list**, it is converted to an extractor function. Character vectors index by name and numeric vectors index by position; use a list to index by position and name at different levels. Within a list, wrap strings in `get-attr()` to extract named attributes. If a component is not present, the value of `.default` will be returned.
`...`	Additional arguments passed on to `.f`.
`.size, .init`	An integer for moving and initial window size.
`.id`	If not `NULL` a variable with this name will be created giving either the name or the index of the data frame.
`.l`	A list of lists. The length of `.l` determines the number of arguments that `.f` will be called with. List names will be used if present.

See Also

- stretch
- slide2 for sliding window with overlapping observations
- tile2 for tiling window without overlapping observations

Examples

```
.x <- 1:5
.y <- 6:10
.z <- 11:15
.lst <- list(x = .x, y = .y, z = .z)
.df <- as.data.frame(.lst)
stretch2(.x, .y, sum, .size = 2)
stretch2(.lst, .lst, ~ ., .size = 2)
stretch2(.df, .df, ~ ., .size = 2)
pstretch(.lst, sum, size = 1)
pstretch(list(.lst, .lst), ~ ., .size = 2)
```

---

stretcher *Splits the input to a list according to the stretching window size.*

Description

Splits the input to a list according to the stretching window size.

Usage

```
stretcher(.x, .size = 1, .init = 1)

pstretcher(..., .size = 1, .init = 1)
```

Arguments

.x	An object to slide over.
.size	An integer for window size. If positive, moving forward from left to right; if negative, moving backward (from right to left).
.init	A positive integer for an initial window size.
...	Multiple objects to be splitted in parallel.
x	An objects to be splitted.

Examples

```
.x <- 1:5
.y <- 6:10
.z <- 11:15
.lst <- list(x = .x, y = .y, z = .z)
.df <- as.data.frame(.lst)

stretcher(.x, .size = 2)
stretcher(.lst, .size = 2)
pstretcher(.lst, .size = 2)
stretcher(.df, .size = 2)
pstretcher(.df, .df, .size = 2)
```

---

summarise.tbl_ts *Collapse multiple rows to a single value*

Description

Collapse multiple rows to a single value

Usage

```
## S3 method for class 'tbl_ts'
summarise(.data, ..., .drop = FALSE)

## S3 method for class 'tbl_ts'
summarize(.data, ..., .drop = FALSE)
```

Arguments

.data	A tsibble.
...	Name-value pairs of expressions.
.drop	FALSE returns a tsibble object as the input. TRUE drops a tsibble and returns a tibble.

Details

Time index will not be collapsed by summarise.tbl_ts.

See Also

dplyr::summarise

dplyr::summarize

Examples

```
# Sum over sensors ----
pedestrian %>%
  summarise(Total = sum(Count))
# Sum over sensors by days ----
pedestrian %>%
  index_by(Date) %>%
  summarise(Total = sum(Count))
## .drop = TRUE ----
pedestrian %>%
  summarise(Total = sum(Count), .drop = TRUE)
```

`tile` *Tiling window calculation*

Description

Tiling window without overlapping observations:

- `tile()` always returns a list.
- `tile_lgl()`, `tile_int()`, `tile_dbl()`, `tile_chr()` use the same arguments as `tile()`, but return vectors of the corresponding type.
- `tile_dfr()` `tile_dfc()` return data frames using row-binding & column-binding.

Usage

```
tile(.x, .f, ..., .size = 1)

tile_dfr(.x, .f, ..., .size = 1, .id = NULL)

tile_dfc(.x, .f, ..., .size = 1)
```

Arguments

`.x`	An object to slide over.
`.f`	A function, formula, or atomic vector. If a **function**, it is used as is. If a **formula**, e.g. `~ .x + 2`, it is converted to a function. There are three ways to refer to the arguments: • For a single argument function, use `.` • For a two argument function, use `.x` and `.y` • For more arguments, use `..1`, `..2`, `..3` etc

This syntax allows you to create very compact anonymous functions.

If **character vector**, **numeric vector**, or **list**, it is converted to an extractor function. Character vectors index by name and numeric vectors index by position; use a list to index by position and name at different levels. Within a list, wrap strings in `get-attr()` to extract named attributes. If a component is not present, the value of `.default` will be returned.

`...` Additional arguments passed on to `.f`.

`.size` An integer for window size. If positive, moving forward from left to right; if negative, moving backward (from right to left).

`.id` If not `NULL` a variable with this name will be created giving either the name or the index of the data frame.

See Also

- tile2, ptile
- slide for sliding window with overlapping observations
- stretch for expanding more observations

Examples

```
.x <- 1:5
.lst <- list(x = .x, y = 6:10, z = 11:15)
tile_dbl(.x, mean, .size = 2)
tile_lgl(.x, ~ mean(.) > 2, .size = 2)
tile(.lst, ~ ., .size = 2)
```

`tile2` *Tiling window calculation over multiple inputs simultaneously*

Description

Tiling window without overlapping observations:

- `tile2()` and `ptile()` always returns a list.
- `tile2_lgl()`, `tile2_int()`, `tile2_dbl()`, `tile2_chr()` use the same arguments as `tile2()`, but return vectors of the corresponding type.
- `tile2_dfr()` `tile2_dfc()` return data frames using row-binding & column-binding.

Usage

```
tile2(.x, .y, .f, ..., .size = 1)

tile2_dfr(.x, .y, .f, ..., .size = 1, .id = NULL)

tile2_dfc(.x, .y, .f, ..., .size = 1)
```

```
ptile(.l, .f, ..., .size = 1)

ptile_dfr(.l, .f, ..., .size = 1, .id = NULL)

ptile_dfc(.l, .f, ..., .size = 1)
```

Arguments

`.x`	Objects to slide over simultaneously.
`.y`	Objects to slide over simultaneously.
`.f`	A function, formula, or atomic vector. If a **function**, it is used as is. If a **formula**, e.g. `~ .x + 2`, it is converted to a function. There are three ways to refer to the arguments: • For a single argument function, use `.` • For a two argument function, use `.x` and `.y` • For more arguments, use `..1`, `..2`, `..3` etc This syntax allows you to create very compact anonymous functions. If **character vector**, **numeric vector**, or **list**, it is converted to an extractor function. Character vectors index by name and numeric vectors index by position; use a list to index by position and name at different levels. Within a list, wrap strings in `get-attr()` to extract named attributes. If a component is not present, the value of `.default` will be returned.
`...`	Additional arguments passed on to `.f`.
`.size`	An integer for window size. If positive, moving forward from left to right; if negative, moving backward (from right to left).
`.id`	If not `NULL` a variable with this name will be created giving either the name or the index of the data frame.
`.l`	A list of lists. The length of `.l` determines the number of arguments that `.f` will be called with. List names will be used if present.

See Also

- tile
- slide2 for sliding window with overlapping observations
- stretch2 for expanding more observations

Examples

```
.x <- 1:5
.y <- 6:10
.z <- 11:15
.lst <- list(x = .x, y = .y, z = .z)
.df <- as.data.frame(.lst)
tile2(.x, .y, sum, .size = 2)
tile2(.lst, .lst, ~ ., .size = 2)
```

```
tile2(.df, .df, ~ ., .size = 2)
ptile(.lst, sum, size = 1)
ptile(list(.lst, .lst), ~ ., .size = 2)
```

`tiler` *Splits the input to a list according to the tiling window size.*

Description

Splits the input to a list according to the tiling window size.

Usage

```
tiler(.x, .size = 1)

ptiler(..., .size = 1)
```

Arguments

`.x`	An object to slide over.
`.size`	An integer for window size. If positive, moving forward from left to right; if negative, moving backward (from right to left).
`...`	Multiple objects to be split in parallel.

Examples

```
.x <- 1:5
.y <- 6:10
.z <- 11:15
.lst <- list(x = .x, y = .y, z = .z)
.df <- as.data.frame(.lst)

tiler(.x, .size = 2)
tiler(.lst, .size = 2)
ptiler(.lst, .size = 2)
ptiler(list(.x, .y), list(.y))
ptiler(.df, .size = 2)
ptiler(.df, .df, .size = 2)
```

`tourism` *Australian domestic overnight trips*

Description

A dataset containing the quarterly overnight trips from 1998 Q1 to 2016 Q4 across Australia.

Usage

```
tourism
```

Format

A tsibble with 23,408 rows and 5 variables:

- **Quarter**: Year quarter (index)
- **Region**: The tourism regions are formed through the aggregation of Statistical Local Areas (SLAs) which are defined by the various State and Territory tourism authorities according to their research and marketing needs
- **State**: States and territories of Australia
- **Purpose**: Stopover purpose of visit:
  - "Holiday"
  - "Visiting friends and relatives"
  - "Business"
  - "Other reason"
- **Trips**: Overnight trips in thousands

Details

This data gives an example of nested and crossed time series structure. *Region* and *State* together form a geographical hierarchy. In other words, *Region* is nested into *State*. These two geographical variables are crossed with *Purpose* of visit. The resulting structure is `Region | State, Purpose`.

References

Tourism Research Australia

Examples

```
data(tourism)
# nesting and crossed structure
key(tourism)
# Total trips over geographical regions
tourism %>%
  group_by(Region, State) %>%
  summarise(Total_Trips = sum(Trips))
```

`tsibble` *Create a tsibble object*

Description

Create a tsibble object

Usage

```
tsibble(..., key = id(), index, regular = TRUE)
```

Arguments

`...`	A set of name-value pairs. The names of "key" and "index" should be avoided as they are used as the arguments.
`key`	Structural variable(s) that define unique time indices, used with the helper id. If a univariate time series (without an explicit key), simply call `id()`. See below for details.
`index`	A bare (or unquoted) variable to specify the time index variable.
`regular`	Regular time interval (`TRUE`) or irregular (`FALSE`). `TRUE` finds the greatest common divisor of positive time distances as the interval.

Details

A tsibble is sorted by its key first and index.

Value

A tsibble object.

Index

The time indices are no longer an attribute (for example, the `tsp` attribute in a `ts` object), but preserved as the essential component of the tsibble. A few index classes, such as `Date`, `POSIXct`, and `difftime`, forms the basis of the tsibble, with new additions yearweek, yearmonth, and yearquarter representing year-week, year-month, and year-quarter respectively. `zoo::yearmth` and `zoo::yearqtr` are also supported. For a `tbl_ts` of regular interval, a choice of index representation has to be made. For example, a monthly data should correspond to time index created by yearmonth or `zoo::yearmth`, instead of `Date` or `POSIXct`. Because months in a year ensures the regularity, 12 months every year. However, if using `Date`, a month contains days ranging from 28 to 31 days, which results in irregular time space. This is also applicable to year-week and year-quarter.

Since the **tibble** that underlies the **tsibble** only accepts a 1d atomic vector or a list, a `tbl_ts` doesn't accept `POSIXlt` and `timeDate` columns.

Key

Key variable(s) together with the index uniquely identifies each record. And key(s) also imposes the structure on a tsibble, which can be created via the id function as identifiers:

- None: an implicit variable `id()` resulting a univariate time series.
- A single variable: an explicit variable. For example, `data(pedestrian)` uses the `id(Sensor)` column as the key.
- Nested variables: a nesting of one variable under another. For example, `data(tourism)` contains two geographical locations: `Region` and `State`. `Region` is the lower level than `State` in Australia; in other words, `Region` is nested into `State`, which naturally forms a hierarchy. A vertical bar (`|`) is used to describe this nesting relationship, and thus `Region | State`. In theory, nesting can involve infinite levels, so is `tsibble`.
- Crossed variables: a crossing of one variable with another. For example, the geographical locations are crossed with the purpose of visiting (`Purpose`) in the `data(tourism)`. A comma (`,`) is used to indicate this crossing relationship. Nested and crossed variables can be combined, such as `data(tourism)` using `id(Region | State, Purpose)`.

These key variables describe the data structure.

Interval

The interval function returns the interval associated with the tsibble.

- Regular: the value and its time unit including "second", "minute", "hour", "day", "week", "month", "quarter", "year". An unrecognisable time interval is labelled as "unit".
- Irregular: `as_tsibble(regular = FALSE)` gives the irregular tsibble. It is marked with `!`.
- Unknown: if there is only one entry for each key variable, the interval cannot be determined (?).

See Also

build_tsibble

Examples

```
# create a tsibble w/o a key ----
tsbl1 <- tsibble(
  date = seq(as.Date("2017-01-01"), as.Date("2017-01-10"), by = 1),
  value = rnorm(10),
  key = id(), index = date
)
tsbl1

# create a tsibble with one key ----
tsbl2 <- tsibble(
  qtr = rep(yearquarter(seq(2010, 2012.25, by = 1 / 4)), 3),
  group = rep(c("x", "y", "z"), each = 10),
  value = rnorm(30),
  key = id(group), index = qtr
```

```
)
tsbl2
```

`units_since` *Time units since Unix Epoch*

Description

Time units since Unix Epoch

Usage

```
units_since(x)
```

Arguments

`x`	An object of `POSIXct`, `Date`, `yearweek`, `yearmonth`, `yearquarter`.

Details

origin:

- `POSIXct`: 1970-01-01 00:00:00
- `Date`: 1970-01-01
- `yearweek`: 1970 W01 (i.e. 1969-12-29)
- `yearmonth`: 1970 Jan
- `yearquarter`: 1970 Qtr1

Examples

```
units_since(x = yearmonth(2012 + (0:11) / 12))
```

`unnest.lst_ts` *Unnest a list column.*

Description

Unnest a list column.

Usage

```
## S3 method for class 'lst_ts'
unnest(data, ..., key = id(), .drop = NA, .id = NULL,
  .sep = NULL, .preserve = NULL)
```

Arguments

data	A lst_ts.
...	Specification of columns to unnest. Use bare variable names or functions of variables. If omitted, defaults to all list-cols.
key	Unquoted variables to create the key (via id) after unnesting.
.drop	Should additional list columns be dropped? By default, unnest will drop them if unnesting the specified columns requires the rows to be duplicated.
.id	Data frame identifier - if supplied, will create a new column with name .id, giving a unique identifier. This is most useful if the list column is named.
.sep	If non-NULL, the names of unnested data frame columns will combine the name of the original list-col with the names from nested data frame, separated by .sep.
.preserve	Optionally, list-columns to preserve in the output. These will be duplicated in the same way as atomic vectors. This has dplyr::select semantics so you can preserve multiple variables with .preserve = c(x, y) or .preserve = starts_with("list").

Value

A tbl_ts.

See Also

tidyr::unnest, nest.tbl_ts for the inverse operation.

Examples

```
nested_ped <- pedestrian %>%
  nest(-Sensor)
nested_ped %>%
  unnest(key = id(Sensor))
nested_tourism <- tourism %>%
  nest(-Region, -State)
nested_tourism %>%
  unnest(key = id(Region | State))
```

---

yearweek *Represent year-week (ISO), year-month or year-quarter objects*

Description

Create or coerce using yearweek(), yearmonth(), or yearquarter()

Usage

```
yearweek(x)

is_53weeks(year)

yearmonth(x)

yearquarter(x)
```

Arguments

x	Other object.
year	A vector of years.

Details

It's a known issue that these attributes will be dropped when using group_by and mutate together. It is recommended to ungroup first, and then use mutate.

Value

Year-week (yearweek), year-month (yearmonth) or year-quarter (yearquarter) objects.

TRUE/FALSE if the year has 53 ISO weeks.

Index functions

The tsibble yearmonth() and yearquarter() function preserve the time zone of the input x, contrasting to their zoo counterparts.

See Also

pull_interval

Examples

```
# coerce POSIXct/Dates to yearweek, yearmonth, yearquarter ----
x <- seq(as.Date("2016-01-01"), as.Date("2016-12-31"), by = "1 month")
yearweek(x)
yearmonth(yearweek(x)); yearmonth(x)
yearquarter(x)

# coerce numerics to yearmonth, yearquarter ----
yearmonth(seq(2010, 2017, by = 1 / 12))
yearquarter(seq(2010, 2017, by = 1 / 4))

# coerce yearmonths to yearquarter ----
y <- yearmonth(x)
yearquarter(y)

# S3 method seq() ----
```

```
wk1 <- yearweek(as.Date("2017-11-01"))
wk2 <- yearweek(as.Date("2018-04-29"))
seq(from = wk1, to = wk2, by = 2) # by two weeks
mth <- yearmonth(as.Date("2017-11-01"))
seq(mth, length.out = 5, by = 1) # by 1 month
seq(yearquarter(mth), length.out = 5, by = 1) # by 1 quarter
is_53weeks(2015:2016)
```

Index