Package 'tsibble'

May 11, 2018

Type Package

Title Tidy Temporal Data Frames and Tools

Version 0.2.0

Date 2018-05-11

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.

Depends R (>= 3.1.3)

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

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

LinkingTo Rcpp (>= 0.12.0)

ByteCompile true

VignetteBuilder knitr

License GPL (>= 3)

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

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

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 2018-05-11 08:58:28 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. It is extensible to work with custom index, for example trading days and weekly data.

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

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

```
## 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 in ascending order. 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* NAs

---

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)
```

`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

---

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"
))
```

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))
```

`inform_duplicates` *Defunct functions*

Description

Defunct functions

Usage

```
inform_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.

`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` *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_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. 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.

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

See Also

dplyr::mutate

dplyr::transmute

`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 vector of numerics
- `slide_lst()` returns a list
- `slide_dfr()` return data frame using row-binding
- `slider()` splits the input `x` to a list according to the window size.

Usage

```
slide(x, .f, ..., size = 1, fill = NA_real_)

slide_lst(x, .f, ..., size = 1, fill = NA)

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

slider(x, size = 1)
```

Arguments

`x`	A vector of numerics, or data frame. If a data frame, row-wise rolling window is performed.
`.f`	A function or one-sided formula using purrr-like syntax. If a formula, it is converted to a function.
`...`	Additional arguments passed on to `.f`.
`size`	An integer for window size.
`fill`	A single value or data frame to replace `NA`.
`.id`	If not `NULL` a variable with this name will be created giving either the name or the index of the data frame, which is passed to dplyr::bind_rows.

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

tile for tiling window without overlapping observations; stretch for expanding more observations

Examples

```
# sliding through a vector ----
x <- 1:10
slide(x, mean, size = 3)
slide(x, ~ mean(.), size = 3)
slide(x, mean, size = 3, fill = 0)

# slider ----
slider(x, size = 3)

## Not run:
# takes a little longer for cran check
# sliding a 2-day window for a data frame ----
jan <- pedestrian %>%
  filter(Date <= as.Date("2015-01-31")) %>%
  split_by(Sensor)
# returns a data frame of fitted values and residuals for each sensor,
# and then combines
diag_jan <- jan %>%
  purrr::map_dfr(
    ~ slide_dfr(., function(x) {
      fit <- lm(Count ~ Time, data = x)
      data.frame(fitted = fitted(fit), resid = residuals(fit))
    }, size = 48)
  )
diag_jan[48:57, ]
# save lm models as additional columns
lm_jan <- jan %>%
  purrr::map(
    ~ mutate(., lm = slide_lst(., ~ lm(Count ~ Time, data = .), size = 48)
  ))
lm_jan[[1]][48:57, ]

## End(Not run)
```

`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)
```

`stretch` *Stretching window calculation*

Description

Fixing an initial window and expanding more observations:

- `stretch()` always returns a vector of numerics
- `stretch_lst()` returns a list
- `stretch_dfr()` return data frame using row-binding
- `stretcher()` splits the input `x` to a list according to the window size.

Usage

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

stretch_lst(x, .f, ..., size = 1, init = 1)

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

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

Arguments

`x`	A vector of numerics, or data frame. If a data frame, row-wise rolling window is performed.
`.f`	A function or one-sided formula using purrr-like syntax. If a formula, it is converted to a function.
`...`	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, which is passed to dplyr::bind_rows.

See Also

slide for sliding window with overlapping observations; tile for tiling window without overlapping observations.

Examples

```
x <- 1:10
stretch(x, mean, init = 3)
stretch(x, ~ mean(.), init = 3)
stretcher(x, init = 3)

# stretching a 2-day window for a data frame ----
sx <- pedestrian %>%
  filter(Sensor == "Southern Cross Station", Date <= as.Date("2015-01-10"))
sx %>%
  stretch_dfr(~ quantile(.$Count), init = 48)
```

---

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 vector of numerics
- `tile_lst()` returns a list
- `tile_dfr()` return data frame using row-binding
- `tiler()` splits the input `x` to a list according to the window size.

Usage

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

tile_lst(x, .f, ..., size = 1)

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

tiler(x, size = 1)
```

Arguments

`x`	A vector of numerics, or data frame. If a data frame, row-wise rolling window is performed.
`.f`	A function or one-sided formula using purrr-like syntax. If a formula, it is converted to a function.
`...`	Additional arguments passed on to `.f`.
`size`	An integer for 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, which is passed to dplyr::bind_rows.

See Also

slide for sliding window with overlapping observations; stretch for expanding more observations

Examples

```
# tiling over a vector ----
x <- 1:10
tile(x, sum, size = 3)
tile(x, ~ sum(.), size = 3)
tiler(x, size = 3)

# tiling over a 2-day window for hourly data ----
## Not run:
pedestrian %>%
  split_by(Sensor) %>%
  purrr::map_dfr(~ tile_dfr(., ~ quantile(.$Count), size = 48))

## End(Not run)
```

---

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(s) 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. It is extensible to work with custom index, for example trading days and weekly data.

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
```

`tsummarise` *Aggregate over calendar periods*

Description

It computes summary statistics for a tsibble over calendar periods, usually used in combination of `group_by`.

Usage

```
tsummarise(.data, ...)

tsummarise_all(.data, ..., .funs)

tsummarise_if(.data, ..., .predicate, .funs)

tsummarise_at(.data, ..., .vars, .funs)

tsummarize(.data, ...)

tsummarize_all(.data, ..., .funs)

tsummarize_if(.data, ..., .predicate, .funs)

tsummarize_at(.data, ..., .vars, .funs)
```

Arguments

`.data`	A data frame (of `tbl_ts` class).
`...`	Name-value pairs of expressions. The index variable must be present in the first name-value pair, with an index function. The remaining components work like `summarise()`. For the scoped variants like `_all()`, `_at()`, `_if()`, additional arguments for the function call in `.funs` will be ignored in `....` 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

`.funs` List of function calls generated by `funs()`, or a character vector of function names, or simply a function.

Bare formulas are passed to `rlang::as_function()` to create purrr-style lambda functions. Note that these lambda prevent hybrid evaluation from happening and it is thus more efficient to supply functions like `mean()` directly rather than in a lambda-formula.

`.predicate` A predicate function to be applied to the columns or a logical vector. The variables for which `.predicate` is or returns `TRUE` are selected. This argument is passed to `rlang::as_function()` and thus supports quosure-style lambda functions and strings representing function names.

`.vars` A list of columns generated by `vars()`, a character vector of column names, a numeric vector of column positions, or `NULL`.

Details

- For a grouped tsibble, the rightmost grouping variable will be dropped after the operation.
- The scoped variants only operate on the non-key and non-index variables.

See Also

dplyr::summarise_all

---

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

Description

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

Usage

```
yearweek(x)

yearmonth(x)

yearquarter(x)
```

Arguments

x	Other object.

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.

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
```

Index