Package: tdata (via r-universe)

September 12, 2024

Title Prepare Your Time-Series Data for Further Analysis

Version 0.3.0

Description Provides a set of tools for managing time-series data, with a particular emphasis on defining various frequency types such as daily and weekly. It also includes functionality for converting data between different frequencies.

License GPL (>= 3)

URL https://github.com/rmojab63/LDT

VignetteBuilder knitr

Encoding UTF-8

SystemRequirements C++17

RoxygenNote 7.2.3

Depends R (>= 3.5.0)

Imports Rcpp

Suggests knitr, testthat, rmarkdown

LinkingTo BH, Rcpp

Config/testthat/edition 3

LazyData true

NeedsCompilation yes

Author Ramin Mojab [aut, cre]

Maintainer Ramin Mojab <rmojab63@gmail.com>

Date/Publication 2023-11-07 15:30:02 UTC

Repository https://rmojab63.r-universe.dev

RemoteUrl https://github.com/cran/tdata

RemoteRef HEAD

RemoteSha dce9befc22c743bbd868fd2967b70934884d4577

Contents

as.character.ldtf *Convert Frequency to Character*

Description

This function converts a frequency to its string representation. The format is explained in the `f.?` functions.

Usage

```
## S3 method for class 'ldtf'
as.character(x, ...)
```

Arguments

`x`	The value of the frequency, which must be an `ldtf` object returned from the `f.?` functions.
`...`	Additional arguments.

Value

A string representation of the value of the frequency.

as.character.ldtv *Convert a Variable to Character String*

Description

Use this function to convert a variable to a compact form.

Usage

```
## S3 method for class 'ldtv'
as.character(x, ...)
```

Arguments

`x`	An object of class `ldtv`.
`...`	Additional arguments.

Details

The returned character will have just one line, with items separated by `tab` or `semi-colon`.

Value

A character that represents the variable.

Examples

```
# define the variable:
data <- c(1,2,3,2,3,4,5)
start_f <- f.monthly(2022,12)
fields <- list(c("key1","value1"), c("key2", "value2"))
v1 = variable(data,start_f, "V1", fields)

#string representation:
v1_str <- as.character(v1)
```

---

as.data.frame.ldtv *Convert Variable to Data Frame*

Description

Use this function to convert a variable to a data frame. You can use the result for plotting.

Usage

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

Arguments

x	An ldtv object.
...	Additional arguments.

Value

A data frame in which row names are set from the frequency of the variable.

Examples

```
# Define the variable:
data <- c(1,2,3,2,3,4,5)
start_f <- f.monthly(2022,12)
fields <- list(c("key1","value1"), c("key2", "value2"))
v1 = variable(data,start_f,"V1", fields)

# convert it to data.frame
df1 <- as.data.frame(v1)
```

as.frequency *Convert Character String to Frequency*

Description

Use this function to convert a character string back to a frequency. You need the class id information.

Usage

```
as.frequency(str, classId)
```

Arguments

str	The value of the frequency as a valid character, which you might have obtained from the as.character.ldtf function.
classId	The class id of the frequency. These are explained in f.? functions.

Value

A frequency, which is an object of class 'ldtf'. See the f.? functions.

as.numeric.ldtv *Coerce Variable to 'numeric'*

Description

Coerce Variable to 'numeric'

Usage

```
## S3 method for class 'ldtv'
as.numeric(x, ...)
```

Arguments

x	Variable with data field.
...	Other arguments.

Value

data in x.

`bind.variables` *Bind Variables and Create a Data.frame*

Description

Use this function to bind variables with the same class of frequency together.

Usage

```
bind.variables(
  varList,
  interpolate = FALSE,
  adjustLeadLags = FALSE,
  numExo = 0,
  horizon = 0
)
```

Arguments

`varList`	A list of variables (i.e., `ldtv` objects) with similar frequency class.
`interpolate`	If `TRUE`, missing observations are interpolated.
`adjustLeadLags`	If `TRUE`, leads and lags are adjusted with respect to the first variable.
`numExo`	An integer representing the number of exogenous variables.
`horizon`	An integer representing the required length of out-of-sample data if `adjustLeadLags` is `TRUE` and there are exogenous variables. It creates lags of exogenous variables or omits `NaN`s to make data available.

Value

A list with the following members:

`data`	A numeric matrix representing the final data after the requested fixes. It is a matrix with variables in the columns and frequencies as the row names.
`info`	An integer matrix containing information about the columns of the final data, such as range of data, missing data, lags/leads, etc.

Examples

```
v1 = variable(c(1,2,3,2,3,4,5),f.monthly(2022,12),"V1")
v2 = variable(c(10,20,30,20,30,40,50),f.monthly(2022,8),"V2")
L = bind.variables(list(v1,v2))
```

convert.to.daily *Convert Data to Daily Frequency*

Description

Use this function to convert a time-series data (currently implemented: `Date-List`, `Daily-In-Week`) to a time-series data with daily frequency.

Usage

```
convert.to.daily(variable, aggregateFun = NULL)
```

Arguments

`variable`	A variable.
`aggregateFun`	Function to aggregate data within each interval (see details).

Details

In some cases, conversion sorts the dates and fills any gaps between them with `NA`. However, in other cases, conversion requires aggregation. For example, when aggregating hourly data over a period of `k` hours to generate daily data, we expect `k` numbers in each interval. The aggregate function can be set to calculate the mean, variance, median, etc., or any function that takes the vector of `k` values and returns a number.

Value

A variable with daily frequency, with data sorted from the original variable and missing dates filled with `NA`.

Examples

```
startFreq <- f.list.date(c("20220904","20220901"), "20220901")
v <- variable(c(4,1), startFreq)
w <- convert.to.daily(v)
```

convert.to.multidaily *Convert Data to Multi-Day Frequency*

Description

Use this function to convert a time-series data (currently implemented: daily) to a time-series data with multi-day frequency.

Usage

```
convert.to.multidaily(variable, k, aggregateFun, fromEnd = TRUE)
```

Arguments

variable	A variable.
k	Number of days in multi-day frequency, must be positive.
aggregateFun	Function to aggregate data within each interval.
fromEnd	If the number of observations is not divisible by k, this argument matters. If TRUE, the last observation is the combination of k observations. Otherwise, the last observation may be created from fewer observations.

Details

See the details section of the convert.to.daily function.

Value

A variable with multi-day frequency.

Examples

```
startFreq <- f.daily(c(2022, 9, 1))
v <- variable(c(1,2,3,4,5,6,7,8), startFreq)
w <- convert.to.multidaily(v, 3, function(x)mean(x, na.rm=TRUE))
```

---

convert.to.weekly *Convert Data to Weekly Frequency*

Description

Use this function to convert time-series data (currently implemented: daily) to time-series data with weekly frequency.

Usage

```
convert.to.weekly(variable, weekStart, aggregateFun)
```

Arguments

variable	A variable.
weekStart	Determines the start day of the week, can be sun, mon, tue, wed, thu, fri, or sat.
aggregateFun	Function to aggregate data within each interval.

Details

See the details section of the convert.to.daily function.

Value

A variable with weekly frequency.

Examples

```
startFreq <- f.daily(c(2022, 9, 1))
v <- variable(c(1,2,3,4,5,6,7,8), startFreq)
w <- convert.to.weekly(v, "mon", function(x)mean(x, na.rm=TRUE))
```

`convert.to.XxYear` *Convert Data to Year-Based Frequency*

Description

Use this function to convert time-series data (currently implemented: daily) to time-series data with year-based frequency such as monthly, quarterly, yearly, etc.

Usage

```
convert.to.XxYear(variable, x, aggregateFun)
```

Arguments

`variable`	A variable.
`x`	Determines the number of partitions in each year, for example, use 12 for monthly data.
`aggregateFun`	Function to aggregate data within each interval.

Details

See the details section of the `convert.to.daily` function.

Value

A variable with year-based frequency.

Examples

```
startFreq <- f.daily(c(2023,1,1))
v <- variable(c(1:(365*2)), startFreq)
w <- convert.to.XxYear(v,12,function(x)mean(x))
```

`f.cross.section` *Create a Cross-Section Frequency*

Description

This frequency is typically used for indexed data. It is represented by an integer that indicates the position of the observation.

Usage

```
f.cross.section(position)
```

Arguments

`position`	An integer representing the position of the observation.

Details

In order to use the `as.frequency` function for this type of frequency, you need the following information:

- **Character Format** `"#"` (the number is the `position`)
- **Class Id** `"cs"`

Value

An object of class `ldtf` which is also a list with the following members:

`class`	Determines the class of this frequency.
`position`	Determines the `position`.

Examples

```
cs0 <- f.cross.section(10) # this initializes a cross-section frequency

cs0_value_str <-  as.character(cs0) # this will be '10'.
cs0_class_str <- get.class.id(cs0) # this will be 'cs'.

cs_new <- as.frequency("20", "cs")
#      this is a cross-section frequency. It points to position 20.
```

`f.daily` *Create a Daily Frequency*

Description

Use this function to create a frequency for time-series data that occurs daily.

Usage

```
f.daily(date)
```

Arguments

`date`	The date, which can be a list with `year`, `month`, and `day` elements. It can also be an integer array with 3 elements for year, month, and day respectively, or an object that can be used as an argument for the `base::as.Date` function.

Details

To use the `as.frequency` function for this type of frequency, you need the following information:

- **Character Format** `"YYYYMMDD"` (similar to `Weekly`)
- **Class Id** `"d"`

Value

An object of class `ldtf`, which is also a list with the following members:

`class`	Determines the class of this frequency.
`year`	Determines the `year`.
`month`	Determines the `month`.
`day`	Determines the `day`.

Examples

```
d0 <- f.daily(c(2023, 1, 2)) # This is 2/1/2023. Next observation belongs to 3/1/2023.

d0_value_str <-  as.character(d0) # this will be '20230102'.
d0_class_str <- get.class.id(d0) # this will be 'd'.

d_new <- as.frequency("20230109", "d") # This is 9/1/2023.

# Don't use invalid or unsupported dates:

# d_invalid <- try(as.frequency("1399109", "d")) # this is a too old date and unsupported
# d_invalid <- try(as.frequency("20230132", "d")) # invalid day in month
d_invalid <- try(as.frequency("20231331", "d")) # invalid month
```

`f.daily.in.week` *Create a* `Daily-In-Week` *Frequency*

Description

Use this function to create a frequency for time-series data that occurs daily within a subset of a week. The first day of the interval is used as the reference.

Usage

```
f.daily.in.week(date, weekStart = "mon", weekEnd = "fri", forward = TRUE)
```

Arguments

`date`	The date, which can be a list with `year`, `month`, and `day` elements. It can also be an integer array with 3 elements for year, month, and day respectively, or an object that can be used as an argument for the `base::as.Date` function.
`weekStart`	The first day of the week, which can be `sun`, `mon`, `tue`, `wed`, `thu`, `fri`, or `sat`.
`weekEnd`	The last day of the week, which can be one of the values listed for `weekStart`. Together, they define the week.
`forward`	If the current date is not in the week and this value is true, it moves forward to the first day of the week. If this value is false, it moves backward to the last day of the week.

Details

In order to use the `as.frequency` function for this type of frequency, you need the following information:

- **Character Format**: The first day of the interval in `"YYYYMMDD"` format.
- **Class Id**: `"i:...-..."` (where the first '...' represents `weekStart` and the second '...' represents `weekEnd`; e.g., `i:mon-fri` means a week from Monday to Friday)

Value

An object of class `ldtf`. It is also a list with the following members:

`class`	Determines the class of this frequency.
`year`	Determines the `year`.
`month`	Determines the `month`.
`day`	Determines the `day`.
`weekStart`	Determines the `weekStart`.
`weekEnd`	Determines the `weekEnd`.

Examples

```
dw0 <- f.daily.in.week(c(2023, 5, 16), "mon", "fri") # This is 16/5/2023.
dw0_value_str <-  as.character(dw0) # this will be '20230516'.
dw0_class_str <- get.class.id(dw0) # this will be 'i:mon-fri'.

# Let's use the same date with another week definition:
dw1 <- f.daily.in.week(c(2023, 5, 16), "wed", "sat")
#     This is NOT 16/5/2023. It is 17/5/2023.
#     Since it was outside the week, we moved it forward.
dw2 <- f.daily.in.week(c(2023, 5, 16), "wed", "sat", FALSE)
#     This is 13/5/2023. The original day was outside the
#     week, but we moved backward too the end of
#     the previous week (which is Saturday).

dw_new <- as.frequency("20230519", "i:sat-wed")
#     This is 20/1/2023 (by default, it moves forward).

# Don't use invalid or unsupported dates:

dw_invalid <- try(as.frequency("1399109", "d3")) # this is a too old date and unsupported
dw_invalid <- try(as.frequency("20230132", "d4")) # invalid day in month
dw_invalid <- try(as.frequency("20231331", "d5")) # invalid month

# don't use invalid week definitions:
dw_invalid <- try(f.daily.in.week(c(2023, 5, 16), "Wednesday", "sat"))
```

`f.hourly` *Create an 'Hourly' Frequency*

Description

Use this function to create a frequency for time-series data that occurs hourly in a day or a subset of a week.

Usage

```
f.hourly(day, hour)
```

Arguments

`day`	A 'Day-based' object of class `ldtf`, such as `Daily` or `Daily-In-Week`.
`hour`	The index of the hour in the day, which should be between 1 and 24.

Details

In order to use the `as.frequency` function for this type of frequency, you need the following information:

- **Character Format**: `"YYYYMMDD:#"` (where # represents the value of `hour`)
- **Class Id**: `ho|...` (where '...' represents the 'class id' of `day`)

Value

An object of class `ldtf`. It is also a list with the following members:

`class`	Determines the class of this frequency.
`day`	Determines the `day`.
`hour`	Determines the `hour`.

Examples

```
ho0 <- f.hourly(f.daily(c(2023,5,16)),4)

ho0_value_str <-  as.character(ho0) # this will be '20230516:4'.
ho0_class_str <- get.class.id(ho0)
#    this will be 'ho|d'. The second part (i.e., 'd')
#    shows that this frequency is defined in a 'Daily' frequency.

ho_new <- as.frequency("20231101:3", "ho|i:wed-sat")

# Don't make the following mistakes:

ho_invalid <- try(as.frequency("20231101:3", "ho|j:wed-sat"))
#  invalid format in day-based frequency
ho_invalid <- try(f.hourly(f.daily(c(2023,5,16)),25)) # invalid hour
```

`f.list.date` *Create a* `List-Date` *Frequency*

Description

Use this frequency for data with date labels. It is generally a list of dates, but it can also be used to label observations outside this list.

Usage

```
f.list.date(items, value = NULL, reformat = TRUE)
```

Arguments

items	The items in the list in YYYYMMDD format.
value	The current value in YYYYMMDD format. If null, the first value in items is used.
reformat	If the elements of items are not in YYYYMMDD format, set this to be TRUE.

Details

In order to use the as.frequency function for this type of frequency, you need the following information:

- **Character Format**: "YYYYMMDD" (i.e., the item)
- **Class Id**: Ld or Ld:... (where '...' represents the semi-colon-separated items)

Value

An object of class ldtf. It is also a list with the following members:

class	Determines the class of this frequency.
items	Determines the items.
value	Determines the value.

Examples

```
Ld0 <- f.list.date(c("20231101","20220903","20200823","20230303"), "20200823")

Ld0_value_str <-  as.character(Ld0) # this will be '20200823'.
Ld0_class_str <- get.class.id(Ld0)
#      this will be 'Ld:20231101;20220903;20200823;20230303'.

Ld_new <- as.frequency("20231101", "Ld:20231101;20220903;20200823;20230303")
Ld_new0 <- as.frequency("20231101", "Ld")
#     compared to the previous one, its items will be empty

# Don't make the following mistakes:

Ld_invalid <- try(as.frequency("20231102", "Ld:20231101;20220903;20200823;20230303"))
  # 'E' is not a member of the list
Ld_invalid <- try(f.list.date(c("20231101","20220903","20200823","20230303"), "20231102"))
```

f.list.string *Create a* List-String *Frequency*

Description

This frequency is typically used for labeled data. It is generally a list, but it can also be used to label observations outside this list.

Usage

```
f.list.string(items, value)
```

Arguments

items	The items in the list.
value	The current item.

Details

In order to use the as.frequency function for this type of frequency, you need the following information:

- **Character Format**: "..." (where '...' represents the value)
- **Class Id**: Ls or Ls:... (where '...' represents the semi-colon-separated items)

Value

An object of class ldtf, which is also a list with the following members:

class	Determines the class of this frequency.
items	Determines the items.
value	Determines the value.

Examples

```
L0 <- f.list.string(c("A","B","C","D"), "C")

L0_value_str <-  as.character(L0) # this will be 'C'.
L0_class_str <- get.class.id(L0) # this will be 'Ls:A;B;C;D'.

L_new <- as.frequency("A", "Ls:A;B;C;D")
L_new0 <- as.frequency("A", "Ls") # compared to the previous one, its items will be empty

# Don't make the following mistakes:

L_invalid <- try(as.frequency("E", "Ls:A;B;C;D")) # 'E' is not a member of the list
L_invalid <- try(f.list.string(c("A","B","C","D"), "E"))
```

f.minutely *Create a* Minute-ly *Frequency*

Description

Use this function to create a frequency for time-series data that occurs every minute in a day or a subset of a week.

Usage

```
f.minutely(day, minute)
```

Arguments

day	A 'Day-based' object of class ldtf, such as Daily or Daily-In-Week.
minute	The index of the minute in the day, which should be between 1 and 1440.

Details

In order to use the as.frequency function for this type of frequency, you need the following information:

- **Character Format**: "YYYYMMDD:#" (where # represents the value of minute)
- **Class Id**: mi|... (where '...' represents the 'class id' of day)

Value

An object of class ldtf. It is also a list with the following members:

class	Determines the class of this frequency.
day	Determines the day.
minute	Determines the minute.

Examples

```
mi0 <- f.minutely(f.daily(c(2023,5,16)),1200)

mi0_value_str <-  as.character(mi0) # this will be '20230516:1200'.
mi0_class_str <- get.class.id(mi0)
#     this will be 'mi|d'. The second part (i.e., 'd')
#     shows that this frequency is defined in a 'Daily' frequency.

mi_new <- as.frequency("20231101:3", "mi|i:wed-sat")

# Don't make the following mistakes:

mi_invalid <- try(as.frequency("20231101:3", "mi|j:wed-sat"))
#   invalid format in day-based frequency
```

```
mi_invalid <- try(f.minutely(f.daily(c(2023,5,16)),2000)) # invalid minute
```

`f.monthly` *Create a Monthly Frequency*

Description

Use this function to create a frequency for time-series data that occurs monthly.

Usage

```
f.monthly(year, month)
```

Arguments

`year`	An integer representing the year of the observation.
`month`	An integer representing the month of the observation (It should be between 1 to 12).

Details

In order to use the `as.frequency` function for this type of frequency, you need the following information:

- **Character Format** `"#m#"` (first # is the `year`, second # is the `month` (1 to 12); e.g., 2010m8 or 2010m12. Note that 2000m0 or 2000m13 are invalid.
- **Class Id** `"m"`

Value

An object of class `ldtf` which is also a list with the following members:

`class`	Determines the class of this frequency.
`year`	Determines the `year`.
`month`	Determines the `month`.

Examples

```
m0 <- f.monthly(2020, 2)
#     this is a monthly frequency that refers to the second month of the year 2020.

m0_value_str <-  as.character(m0) # this will be '2020M2'.
m0_class_str <- get.class.id(m0) # this will be 'm'.

m_new <- as.frequency("2021m3", "m")
#     this is a monthly frequency that refers to the third month of the year 2021.
```

```
# Don't make the following mistakes:

m_invalid <- try(f.monthly(2020, 0))
m_invalid <- try(f.monthly(2020, 5))
m_invalid <- try(as.frequency("2021m0", "m"))
m_invalid <- try(as.frequency("2021m13", "m"))
m_invalid <- try(as.frequency("2021", "m"))
```

`f.multi.daily` *Create a Multi-Day Frequency*

Description

Use this function to create a frequency for time-series data that occurs every `k` days. The first day of the interval is used as the reference.

Usage

```
f.multi.daily(date, k)
```

Arguments

`date`	The date, which can be a list with `year`, `month`, and `day` elements. It can also be an integer array with 3 elements for year, month, and day respectively, or an object that can be used as an argument for the `base::as.Date` function.
`k`	The number of days in the interval.

Details

In order to use the `as.frequency` function for this type of frequency, you need the following information:

- **Character Format**: The first day of the interval in `"YYYYMMDD"` format.
- **Class Id**: `"d#"` (where # is the value of `k`; e.g., d3 means every 3 days)

Value

An object of class `ldtf`. It is also a list with the following members:

`class`	Determines the class of this frequency.
`year`	Determines the `year`.
`month`	Determines the `month`.
`day`	Determines the `day`.
`k`	Determines the value: `k`.

Examples

```
md0 <- f.multi.daily(c(2023, 1, 2), 4) # This is 2/1/2023. Next observation belongs to 6/1/2023.

md0_value_str <-  as.character(md0) # this will be '20230102'.
md0_class_str <- get.class.id(md0) # this will be 'd4'.

md_new <- as.frequency("20230109", "d") # This is 9/1/2023.

# Don't use invalid or unsupported dates:

md_invalid <- try(as.frequency("1399109", "d3")) # this is a too old date and unsupported
md_invalid <- try(as.frequency("20230132", "d4")) # invalid day in month
md_invalid <- try(as.frequency("20231331", "d5")) # invalid month
```

`f.multi.weekly` *Create a Multi-Week Frequency*

Description

Use this function to create a frequency for time-series data that occurs every 'k' weeks. The first day of the first week is used as the reference.

Usage

```
f.multi.weekly(date, k)
```

Arguments

`date`	The date, which can be a list with `year`, `month`, and `day` elements. It can also be an integer array with 3 elements for year, month, and day respectively, or an object that can be used as an argument for the `base::as.Date` function.
`k`	The number of weeks.

Details

To use the `as.frequency` function for this type of frequency, you need the following information:

- **Character Format** The first day of the first week in `"YYYYMMDD"` format.
- **Class Id** `"w#"` (the number is the value of `k`; e.g., w3 means every 3 weeks)

Value

An object of class `ldtf`, which is also a list with the following members:

`class`	The class of this frequency.
`year`	The `year`.

month	The month.
day	The day.
k	The value of k.

Examples

```
mw0 <- f.multi.weekly(c(2023, 1, 2), 3)
# This is 2/1/2023, which is Monday. The next observation belongs to 23/1/2023.

mw0_value_str <- as.character(mw0) # This will be '20230102'.
mw0_class_str <- get.class.id(mw0) # This will be 'w3'.

mw_new <- as.frequency("20230109", "w4") # This is 9/1/2023.

# Don't use invalid or unsupported dates:

mw_invalid <- try(as.frequency("1399109", "w4")) # this is a too old date and unsupported
mw_invalid <- try(as.frequency("20230132", "w5")) # invalid day in month
mw_invalid <- try(as.frequency("20231331", "w2")) # invalid month
mw_invalid <- try(as.frequency("20231012", "w0"))
```

---

f.multi.yearly *Create a Multi-Year Frequency*

Description

Use this function to create a frequency for time-series data that occurs every z years.

Usage

```
f.multi.yearly(year, z)
```

Arguments

year	An integer representing the year of the observation.
z	An integer representing the number of years. It should be larger than zero.

Details

In order to use the as.frequency function for this type of frequency, you need the following information:

- **Character Format** "#" (the number is the year, which means the string representation is the first year of the interval)
- **Class Id** "z#" ('#' represents the value: z; e.g., z3 means every 3 years)

Value

An object of class `ldtf` which is also a list with the following members:

`class`	Determines the class of this frequency.
`year`	Determines the `year`.
`z`	Determines the value: `z`.

Examples

```
my0 <- f.multi.yearly(2020, 2)
#      this is a multi-year frequency that refers to the year 2020.
#      The next observation is expected in 2022 (not the next year).

my0_value_str <-  as.character(my0) # this will be '2020'.
my0_class_str <- get.class.id(my0) # this will be 'z2'.

my_new <- as.frequency("2020", "z3")
#      this is a multi-year frequency that refers to the year 2020.
#      However, the next observation is expected in 2023.

# Don't make the following mistakes:

my_invalid <- try(f.multi.yearly(2020, 0))
my_invalid <- try(f.multi.yearly(2020, -5))
my_invalid <- try(as.frequency("2021", "z"))
```

`f.quarterly` *Create a Quarterly Frequency*

Description

Use this function to create a frequency for time-series data that occurs quarterly.

Usage

```
f.quarterly(year, quarter)
```

Arguments

`year`	An integer representing the year of the observation.
`quarter`	An integer representing the quarter of the observation (It should be between 1 and 4).

Details

In order to use the `as.frequency` function for this type of frequency, you need the following information:

- **Character Format** `"#q#"` (first '#' is the `year`, second '#' is the `quarter`; e.g., 2010q3 or 2010q4. Note that 2000q0 or 2000q5 are invalid.
- **Class Id** `"q"`

Value

An object of class `ldtf` which is also a list with the following members:

`class`	Determines the class of this frequency.
`year`	Determines the `year`.
`quarter`	Determines the `quarter`.

Examples

```
q0 <- f.quarterly(2020, 2)
#     this is a quarterly frequency that refers to the second quarter of the year 2021.

q0_value_str <-  as.character(q0) # this will be '2020Q2'.
q0_class_str <- get.class.id(q0) # this will be 'q'.

q_new <- as.frequency("2021q3", "q")
#      this is a quarterly frequency that refers to the third quarter of the year 2021.

# Don't make the following mistakes:

q_invalid <- try(f.quarterly(2020, 0))
q_invalid <- try(f.quarterly(2020, 5))
q_invalid <- try(as.frequency("2021q0", "q"))
q_invalid <- try(as.frequency("2021q5", "q"))
q_invalid <- try(as.frequency("2021", "q"))
```

`f.secondly` *Create a* `Second-ly` *Frequency*

Description

Use this function to create a frequency for time-series data that occurs every second in a day or a subset of a week.

Usage

```
f.secondly(day, second)
```

Arguments

day	A 'Day-based' object of class ldtf, such as Daily or Daily-In-Week.
second	The index of the second in the day, which should be between 1 and 86400.

Details

In order to use the as.frequency function for this type of frequency, you need the following information:

- **Character Format**: "YYYYMMDD:#" (where # represents the value of second)
- **Class Id**: se|... (where '...' represents the 'class id' of day)

Value

An object of class ldtf. It is also a list with the following members:

class	Determines the class of this frequency.
day	Determines the day.
second	Determines the second.

Examples

```
se0 <- f.secondly(f.daily(c(2023,5,16)),40032)

se0_value_str <-  as.character(se0) # this will be '20230516:40032'.
se0_class_str <- get.class.id(se0)
#     this will be 'se|d'. The second part (i.e., 'd') shows
#     that this frequency is defined in a 'Daily' frequency.

se_new <- as.frequency("20231101:3", "se|i:wed-sat")

# Don't make the following mistakes:

mi_invalid <- try(as.frequency("20231101:3", "se|j:wed-sat"))
#  invalid format in day-based frequency
mi_invalid <- try(f.secondly(f.daily(c(2023,5,16)),100000)) # invalid second
```

---

f.weekly *Create a Weekly Frequency*

Description

Use this function to create a frequency for time-series data that occurs weekly. The first day of the week is used as the reference.

Usage

```
f.weekly(date)
```

Arguments

date	The date, which can be a list with year, month, and day elements. It can also be an integer array with 3 elements for year, month, and day respectively, or an object that can be used as an argument for the base::as.Date function. This date determines the start of the week.

Details

To use the as.frequency function for this type of frequency, you need the following information:

- **Character Format** The first day of the week in "YYYYMMDD" format.
- **Class Id** "w"

Value

An object of class ldtf, which is also a list with the following members:

class	The class of this frequency.
year	The year.
month	The month.
day	The day.

Examples

```
w0 <- f.weekly(c(2023, 1, 2)) # This is 2/1/2023, which is Monday.
#    The next observation belongs to 9/1/2023.

w0_value_str <-  as.character(w0) # this will be '20230102'.
w0_class_str <- get.class.id(w0) # this will be 'w'.

w_new <- as.frequency("20230109", "w") # This is 9/1/2023.

# Don't use invalid or unsupported dates:

w_invalid <- try(as.frequency("1399109", "w")) # this is a too old date and unsupported
w_invalid <- try(as.frequency("20230132", "w")) # invalid day in month
w_invalid <- try(as.frequency("20231331", "w")) # invalid month
```

`f.x.times.a.day` *Create an* `X-Times-A-Day` *Frequency*

Description

Use this function to create a frequency for time-series data that occurs `x` times in a day or a subset of a week.

Usage

```
f.x.times.a.day(day, x, position)
```

Arguments

`day`	A 'Day-based' object of class `ldtf`, such as `Daily` or `Daily-In-Week`.
`x`	The number of observations in each day.
`position`	The position of the current observation, which should be a positive integer and cannot be larger than `x`.

Details

In order to use the `as.frequency` function for this type of frequency, you need the following information:

- **Character Format**: `"#"` (where '#' represents the value of `position`)
- **Class Id**: `"da#|...."` (where '#' represents the value of `x` and '...' represents the 'class id' of `day`)

Value

An object of class `ldtf`. It is also a list with the following members:

`class`	Determines the class of this frequency.
`day`	Determines the day.
`second`	Determines the second.

Examples

```
xd0 <- f.x.times.a.day(f.daily(c(2023,5,16)),13, 12)

xd0_value_str <-  as.character(xd0) # this will be '20230516:12'.
xd0_class_str <- get.class.id(xd0)
#      this will be 'da13|d'. The second part (i.e., 'd')
#      shows that this frequency is defined in a 'Daily' frequency.

xd_new <- as.frequency("20231101:3", "da3|i:wed-sat")

# Don't make the following mistakes:
```

```
xd_invalid <- try(as.frequency("20231101:3", "da|i:wed-sat"))
#  invalid format in day-based frequency
xd_invalid <- try(f.x.times.a.day(f.daily(c(2023,5,16)),4,0)) # invalid position
```

`f.x.times.a.year` *Create an* `X-Times-A-Year` *Frequency*

Description

Use this function to create a frequency for time-series data that occurs `x` times every year.

Usage

```
f.x.times.a.year(year, x, position)
```

Arguments

`year`	An integer representing the year of the observation.
`x`	An integer representing the number of observations in each year. It should be a positive integer.
`position`	An integer representing the position of the current observation. It should be a positive integer and cannot be larger than `x`.

Details

In order to use the `as.frequency` function for this type of frequency, you need the following information:

- **Character Format** `"#:#"` (first # is the `year` and the second # is the `position`; e.g., 2010:8/12 or 2010:10/10. Note that 2000:0/2 or 2000:13/12 are invalid.
- **Class Id** `"y#"` (the number is the value: `x`)

Value

An object of class `ldtf` which is also a list with the following members:

`class`	Determines the class of this frequency.
`year`	Determines the `year`.
`x`	Determines the value: `x`.
`position`	Determines the `position`.

Examples

```
xty0 <- f.x.times.a.year(2020, 3, 1)
#      this frequency divides the year 2020 into 3 partitions
#      and refers to the first partition.

xty_value_str <-  as.character(xty0) # this will be '2020:1'.
xty_class_str <- get.class.id(xty0) # this will be 'y3'.

xty_new <- as.frequency("2021:24", "z24")
#     this frequency divides the year 2021 into 24 partitions
#     and refers to the last partition.

# Don't make the following mistakes:

xty_invalid <- try(f.x.times.a.year(2020, 3, 0))
xty_invalid <- try(f.x.times.a.year(2020, 24, 25))
xty_invalid <- try(as.frequency("2021:13", "y12"))
xty_invalid <- try(as.frequency("2021:0", "y1"))
xty_invalid <- try(as.frequency("2021", "y1"))
```

---

f.x.times.z.years *Create an* X-Times-Z-Years *Frequency*

Description

Use this function to create a frequency for time-series data that occurs x times every z years.

Usage

```
f.x.times.z.years(year, x, z, position)
```

Arguments

year	An integer representing the year of the observation.
x	An integer representing the number of partitions in each z years. It should be a positive integer.
z	An integer representing the number of years. It should be a positive integer.
position	An integer representing the position of the current observation. It should be a positive integer and cannot be larger than x.

Details

In order to use the as.frequency function for this type of frequency, you need the following information:

- **Character Format** "#:#" (Similar to X-Times-A-Year. Note that the string representation refers to the first year of the interval.)
- **Class Id** "x#z#" (first '#' is the value: x, second '#' is the value: z; e.g., x23z4 means 23 times every 4 years)

Value

An object of class ldtf, which is also a list with the following members:

class	The class of this frequency.
year	The year.
z	The value: z.
x	The value: x.
position	The position.

Examples

```
xtzy0 <- f.x.times.z.years(2020, 3, 2, 3)
# This frequency divides the year 2020 into 3 partitions and
# refers to the last partition. The next observation
# belongs to 2022 (not the next year).

xtzy_value_str <- as.character(xtzy0) # This will be '2020:3'.
xtzy_class_str <- get.class.id(xtzy0) # This will be 'x3z2'.

xtzy_new <- as.frequency("2021:3", "x3z4")
# This frequency divides the year 2021 into 3 partitions
# and refers to the last partition. The next observation occurs after 4 years.

# Don't make the following mistakes:

xtzy_invalid <- try(f.x.times.z.years(2020, 3, 5, 0))
xtzy_invalid <- try(f.x.times.z.years(2020, 3, 0, 1))
xtzy_invalid <- try(as.frequency("2021:25", "x24y2"))
```

---

f.yearly *Create an Annual Frequency*

Description

Use this function to create a frequency for time-series data that occurs annually.

Usage

```
f.yearly(year)
```

Arguments

year	An integer representing the year of the observation.

Details

In order to use the as.frequency function for this type of frequency, you need the following information:

- **Character Format** "#" (the number is the year)
- **Class Id** "y"

Value

An object of class ldtf which is also a list with the following members:

class	Determines the class of this frequency.
year	Determines the year.

Examples

```
y0 <- f.yearly(2020) # this initializes a 'yearly' frequency

y0_value_str <-  as.character(y0) # this will be '2020'.
y0_class_str <- get.class.id(y0) # this will be 'y'.

y_new <- as.frequency("2021", "y") # this is a yearly frequency. It points to year 2021.
```

---

get.class.id *Get the Class Id of a Frequency*

Description

Use this function to get the 'id' of a frequency class.

Usage

```
get.class.id(frequency)
```

Arguments

frequency	The frequency, which must be an ldtf object returned from the f.? functions.

Details

You need this 'id' to convert the character back to the object. Some frequencies have a constant class id, such as 'm' for 'monthly' data. Some class 'ids' have parameters in them. Note that the format is explained in the f.? functions.

Value

A character string that represents the class id of this frequency.

Examples

```
freq <- f.x.times.a.day(f.daily(c(2023,5,16)),13, 12)
freq_class_id <- get.class.id(freq) # this will be 'da13|d'.
```

---

get.class.id0 *Convert Frequency to Character and Class Id*

Description

This function returns the output of the as.character.ldtf and get.class.id functions.

Usage

```
get.class.id0(frequency)
```

Arguments

frequency	The value of the frequency, which must be an `ldtf` object returned from the `f.?` functions.

Value

A list with the following items:

- **value**: The string representation of the frequency. If you only want this, use the `as.character()` function.
- **day**: The class Id of this frequency. If you only want this, use the `get.class.id` function.
- **classType**: The type of the class.

See Also

`get.class.id`

Examples

```
freq <- f.x.times.a.day(f.daily(c(2023,5,16)),13, 12)
freq_class_id <- get.class.id0(freq)

freq1 <- f.monthly(2020,3)
freq1_class_id <- get.class.id0(freq1)
```

`get.longrun.growth` *Calculate Long-run Growth*

Description

Use this function to calculate the long-run growth of a time-series data.

Usage

```
get.longrun.growth(
  data,
  continuous = FALSE,
  isPercentage = FALSE,
  trimStart = 0,
  trimEnd = 0,
  skipZero = TRUE
)
```

Arguments

`data`	A numeric vector that represents the data of the series.
`continuous`	A logical value indicating whether to use the continuous formula.
`isPercentage`	A logical value indicating whether the unit of measurement in `data` is a percentage (e.g., growth rate). If `TRUE`, the long-run growth rate is calculated by the arithmetic mean for the continuous case and the geometric mean otherwise. If missing data exists, it returns `NA`.
`trimStart`	If the number of leading `NA`s is larger than this number, the function returns `NA`. Otherwise, it finds the first non-NA value and continues the calculations.
`trimEnd`	Similar to `trimStart`, but for the end of the series.
`skipZero`	If `TRUE`, leading and trailing zeros are skipped, similar to `NA`.

Details

A variable can have discrete growth ($y(t) = y(0)(1+g_1)(1+g_2)\dots(1+g_t)$) or continuous growth ($y(t) = y(0)e^{g_1}e^{g_2}\dots e^{g_t}$) over t periods. $y(0)$ is the first value and $y(n)$ is the last value. By long-run growth rate, we mean a number such as g such that if we start from $y(0)$ and the variable growth is g every period, we reach $y(t)$ after t periods. This number summarizes all g_is, however, it is not generally the average of these rates.

Value

The long-run growth rate (percentage).

Examples

```
y <- c(60, 70, 80, 95)
g <- get.longrun.growth(y, isPercentage = TRUE, continuous = FALSE)
# Note that 'g' is different from 'mean(y)'.
```

---

get.seq *Generate a Sequence from a Range of Frequencies*

Description

Use this function to generate a list of character strings, where each element is a string representation of a frequency within the specified range.

Usage

```
get.seq(from, to, by = 1)
```

Arguments

from	The first frequency of the sequence.
to	The last frequency of the sequence.
by	An integer that determines the increment of the sequence.

Details

The two arguments from and to should be valid frequencies (see the f.? functions). They should also be consistent; you cannot create a sequence in which one is, for example, monthly and the other is yearly.

Value

A list of character strings that represents the sequence.

See Also

get.seq0

Examples

```
from <- f.monthly(2020,1)
to <- f.monthly(2021,12)
sequence1 <- get.seq(from, to, 1) # this will be '2020M1', '2020M2', ..., '2021M12'
sequence2 <- get.seq(from, to, 2) # this will be '2020M1', '2020M3', ..., '2021M11'
sequence3 <- get.seq(from, to, 3) # this will be '2020M1', '2020M4', ..., '2021M10'

# backward:
sequence4 <- get.seq(to, from, -1) # this will be '2021M12', '2021M11', ..., '2020M1'
```

`get.seq0` *Generate a Sequence from a Range of Frequencies*

Description

Use this function to generate a list of character strings, where each element is a string representation of a frequency within the specified range.

Usage

```
get.seq0(start, length, by = 1)
```

Arguments

`start`	The first frequency of the sequence.
`length`	The length of the sequence.
`by`	An integer that determines the increment of the sequence.

Value

A list of character strings that represents the sequence.

See Also

`get.seq`

Examples

```
start <- f.monthly(2020,1)
sequence1 <- get.seq0(start, 24, 1) # this will be '2020M1', '2020M2', ..., '2021M12'
sequence2 <- get.seq0(start, 24, 2) # this will be '2020M1', '2020M3', ..., '2023M11'
sequence3 <- get.seq0(start, 24, 3) # this will be '2020M1', '2020M4', ..., '2025M10'

# backward:
sequence4 <- get.seq0(start, 24, -1) # this will be '2020M1', '2019M12', ..., '2018M2'

# Lists are a little different:
start_l <- f.list.string(c("A","B","C","D"), "C")
sequence5 <- get.seq0(start_l, 5, 1) # this will be 'C', 'D', 'out_item:1', ..., 'out_item:3'
```

length.ldtv *Get Length of Data in a Variable*

Description

Get Length of Data in a Variable

Usage

```
## S3 method for class 'ldtv'
length(x)
```

Arguments

x	Variable with data field.

Value

Length of data in x.

minus.freqs *Get Interval between two frequencies*

Description

Use this function to get the number of intervals between two frequencies.

Usage

```
minus.freqs(freq1, freq2)
```

Arguments

freq1	The first frequency.
freq2	The second frequency.

Value

The number of intervals between the two frequencies (freq1 - freq2).

Examples

```
f1 <- f.yearly(2000)
f2 <- f.yearly(2010)
count <- minus.freqs(f1, f2) # this is -10
count <- minus.freqs(f2, f1) # this is 10
```

next.freq *Get Next Frequency*

Description

Use this function to get the next frequency.

Usage

```
next.freq(freq, count)
```

Arguments

freq	A frequency.
count	Determines the number of steps. If negative, it returns the previous frequency.

Value

The next frequency after the given frequency.

Examples

```
f <- f.yearly(2000)
fn <- next.freq(f, 10) # this is 2010
```

oil_price *Data for Vignette*

Description

This is oil price data from 2010 retrieved by using the following code: `oil_price <- Quandl::Quandl("OPEC/ORB", start_date="2010-01-01")` It is saved due to the fact that CRAN checks may fail if the vignette relies on an external API call.

Usage

```
oil_price
```

Format

A `data.frame` with 2 columns: `Date` and `Value`

print.ldtf *Print a Frequency*

Description

Print a Frequency

Usage

```
## S3 method for class 'ldtf'
print(x, ...)
```

Arguments

x	A frequency which is the output of f.? functions in this package.
...	Additional arguments

Value

NULL

print.ldtv *Print a Variable*

Description

Use this to print a variable.

Usage

```
## S3 method for class 'ldtv'
print(x, ...)
```

Arguments

x	A variable which is an object of class ldtv.
...	Additional arguments

Value

NULL

`remove.na.strategies` *Scenarios for Removing* NA*s*

Description

Use this function to remove NA values from a matrix. This helps you to optimize the size of the information.

Usage

```
remove.na.strategies(
  data,
  countFun = function(nRows, nCols) nRows * nCols,
  rowIndices = NULL,
  colIndices = NULL,
  printMsg = FALSE
)
```

Arguments

data	A matrix that contains NA values.
countFun	A function to determine how strategies are sorted. The default function counts the number of observations. You might want to give columns a higher level of importance, for example, by using nRows*nCols^1.5.
rowIndices	The indices of the sorted rows to search. Use this to create jumps for a large number of rows (e.g., if the first sorted strategies suggest a small number of columns and you are looking for other strategies). Use NULL to disable it.
colIndices	Similar to rowIndices, but for columns.
printMsg	If TRUE, it prints the progress.

Details

When a matrix has NA values, one can omit columns with NA, rows with NA, or a combination of these two. The total number of observations is a function of the order. This function tries all combinations and returns the results.

Value

A list of lists, each with the following elements:

nRows	The number of rows in the matrix.
nCols	The number of columns in the matrix.
colRemove	The indices of the columns to be removed.
rowRemove	The indices of the rows to be removed.

Examples

```
data <- matrix(c(NA, 2, 3, 4, NA, 5, NA, 6, 7, NA, 9, 10, 11, 12, 13, 14, 15, NA, 16, 17), 4, 5)
res <- remove.na.strategies(data)
```

`row.names.ldtv` *Get Row Names of a Variable*

Description

Get Row Names of a Variable

Usage

```
## S3 method for class 'ldtv'
row.names(x)
```

Arguments

`x`	Variable with `startFrequency` field

Value

A character string vector with frequencies of the observations as the row names.

`variable` *Create a Variable*

Description

Use this function to create a variable, which is a data array with frequencies. It can have a name and other named fields.

Usage

```
variable(data, startFrequency = NULL, name = NULL, fields = NULL)
```

Arguments

`data`	The data of the variable.
`startFrequency`	The frequency of the first element.
`name`	The name of the variable.
`fields`	A list that contains named fields.

Value

An object of class `ldtv`, which is also a list with the following members:

- **data**: Determines the `data`.
- **name**: Determines the `name`.
- **startFrequency**: Determines the `startFrequency`.
- **fields**: Determines the `fields`.

Examples

```
data <- c(1,2,3,2,3,4,5)
start_f <- f.monthly(2022,12)
fields <- list(c("key1","value1"), c("key2", "value2"))
v1 = variable(data, start_f, "V1", fields)
```

Index