

# Package ‘dateutils’

November 10, 2021

**Type** Package

**Title** Date Utils

**Version** 0.1.5

**Author** Seth Leonard [aut, cre],  
Jiancong Liu [ctb]

**Maintainer** Seth Leonard <seth@ottoquant.com>

**Description** Utilities for mixed frequency data. In particular, use to aggregate and normalize tabular mixed frequency data, index dates to end of period, and seasonally adjust tabular data.

**Depends** R (>= 3.5.0)

**License** MIT + file LICENSE

**Suggests** rmarkdown, knitr, testthat

**Imports** Rcpp (>= 0.12.13), data.table (>= 1.9.8), seasonal

**LinkingTo** Rcpp, RcppArmadillo

**RoxygenNote** 7.1.1

**LazyData** true

**Encoding** UTF-8

**VignetteBuilder** knitr, rmarkdown

**URL** <https://github.com/macroeconomicdata/dateutils>

**BugReports** <https://github.com/macroeconomicdata/dateutils/issues>

**NeedsCompilation** yes

**Repository** CRAN

**Date/Publication** 2021-11-10 15:50:10 UTC

## R topics documented:

add_forecast_dates . . . . .	3
agg_to_freq . . . . .	4
agg_to_freq_wide . . . . .	4
allNA . . . . .	5

all_finite . . . . .	6
any_finite . . . . .	6
can_seasonal . . . . .	7
col_to_list . . . . .	7
comp_form . . . . .	8
count_obs . . . . .	8
day . . . . .	9
Diff . . . . .	9
end_of_period . . . . .	10
end_of_year . . . . .	10
extract_basic_character . . . . .	11
extract_character . . . . .	11
extract_numeric . . . . .	12
fill_forward . . . . .	12
first_of_month . . . . .	13
first_of_quarter . . . . .	13
first_previous_quarter . . . . .	14
fred . . . . .	14
fredlib . . . . .	15
get_data_frq . . . . .	15
get_from_list . . . . .	16
index_by_friday . . . . .	16
is_in . . . . .	17
last_in_month . . . . .	17
last_in_quarter . . . . .	18
last_in_week . . . . .	18
last_in_year . . . . .	19
last_obs . . . . .	19
limit_character . . . . .	20
long_run_var . . . . .	21
match_index . . . . .	21
match_ts_dates . . . . .	22
mean_na . . . . .	22
month_days . . . . .	23
number_finite . . . . .	24
numdum . . . . .	24
pct_chng . . . . .	25
pct_response . . . . .	25
process . . . . .	26
process_MF . . . . .	27
process_wide . . . . .	29
rollmax . . . . .	30
rollmean . . . . .	31
rollmin . . . . .	31
row_to_list . . . . .	32
run_sa . . . . .	32
sd_na . . . . .	33
seas_df_long . . . . .	33

seas_df_wide . . . . .	34
spline_fill . . . . .	35
spline_fill_trend . . . . .	35
stack_obs . . . . .	36
sum_na . . . . .	36
total_response . . . . .	37
to_ts . . . . .	37
try_detrend . . . . .	38
try_sa . . . . .	39
try_trend . . . . .	39
ts_to_df . . . . .	40

## Index 41

---

add_forecast_dates	<i>Add NA values to the tail of a wide data.table</i>
--------------------	---

---

### Description

Add NA values to the tail of a wide data.table to be filled by forecasting routines

### Usage

```
add_forecast_dates(
  dt,
  horizon = 1,
  frq = c("month", "week", "quarter", "year"),
  date_name = "ref_date"
)
```

### Arguments

dt	data.table in wide format
horizon	number of periods to add at specified 'frq'
frq	frequency for aggregation, one of "month", "week", "quarter", or "year"
date_name	name of date column

### Value

NA-filled data.table in wide format

### Examples

```
add_forecast_dates(fred[series_name == "gdp constant prices"], frq="quarter")
```

---

**agg\_to\_freq***Aggregate long format data.table*

---

**Description**

Aggregate a data.table in long format to a specified frequency

**Usage**

```
agg_to_freq(  
  dt_long,  
  frq = c("month", "week", "quarter", "year"),  
  date_name = "ref_date",  
  id_name = "series_name",  
  value_name = "value"  
)
```

**Arguments**

dt_long	data.table in long format
frq	frequency for aggregation, one of "month", "week", "quarter", or "year"
date_name	name of date column
id_name	name of id column
value_name	name of value column

**Value**

Aggregated data at specified frequency in long format

**Examples**

```
out <- agg_to_freq(fred[series_name == "gdp constant prices"], frq = "year")
```

---

**agg\_to\_freq\_wide***Aggregate data.table and return wide format*

---

**Description**

Aggregate a data.table to a specified frequency and return wide format data

**Usage**

```
agg_to_freq_wide(
  dt,
  date_name = "ref_date",
  frq = c("month", "week", "quarter", "year"),
  id_name = "series_name",
  value_name = "value",
  dt_is_wide = FALSE
)
```

**Arguments**

dt	data.table in long format
date_name	name of date column
frq	frequency for aggregation, one of "month", "week", "quarter", or "year"
id_name	name of id column
value_name	name of value column
dt_is_wide	T/F, is input data 'dt' in wide format

**Value**

Aggregated data at specified frequency in wide format

**Examples**

```
out <- agg_to_freq_wide(fred, frq="quarter")
```

---

allNA	<i>Are all elements 'NA'?</i>
-------	-------------------------------

---

**Description**

Return a logical indicating if all elements are 'NA'

**Usage**

```
allNA(x)
```

**Arguments**

x	data vector
---	-------------

**Value**

A logical variable indicating all elements are 'NA'

**Examples**

```
allNA(c(NA, NA, 1, NA)) ## FALSE
```

---

all_finite	<i>Rows with only finite values</i>
------------	-------------------------------------

---

**Description**

Return indexes of rows with only finite values

**Usage**

```
all_finite(Y)
```

**Arguments**

Y                    matrix like data object

**Value**

Indexes of rows with with only finite values

**Examples**

```
X <- matrix(1,10,2)
X[3,1] <- NA
all_finite(X)
```

---

any_finite	<i>Rows with finite values</i>
------------	--------------------------------

---

**Description**

Return indexes of rows with at least one finite value

**Usage**

```
any_finite(Y)
```

**Arguments**

Y                    matrix like data object

**Value**

Indexes of rows with at least one finite value

**Examples**

```
X <- matrix(1,10,2)
X[3,] <- NA
any_finite(X)
```

---

can_seasonal	<i>Can data be seasonally adjusted?</i>
--------------	---

---

**Description**

Return a logical indicating whether data at given dates can be seasonally adjusted using seas()

**Usage**

```
can_seasonal(dates)
```

**Arguments**

dates	dates
-------	-------

**Value**

A logical variable indicating whether data can be seasonally adjusted

**Examples**

```
can_seasonal(fred$ref_date[1:20]) ## TRUE
```

---

col_to_list	<i>Convert columns to list</i>
-------------	--------------------------------

---

**Description**

Return 'Y' with each column as a list

**Usage**

```
col_to_list(Y)
```

**Arguments**

Y	matrix like data object
---	-------------------------

**Value**

Each column as a list

**Examples**

```
row_to_list(matrix(rnorm(20),10,2))
```

---

comp\_form                      *Companion Form*

---

**Description**

Put the transition matrix 'B' into companion form

**Usage**

```
comp_form(B)
```

**Arguments**

B                      Transition matrix from a VAR model

**Value**

Companion matrix of the input matrix

**Examples**

```
comp_form(matrix(c(1:4), nrow = 2, byrow = TRUE)) ## matrix(c(4,-2,-3,1), nrow = 2, byrow = TRUE)
```

---

count\_obs                      *Count observations*

---

**Description**

Return the number of finite observations in 'x'

**Usage**

```
count_obs(x)
```

**Arguments**

x                      data vector

**Value**

The Number of observations

**Examples**

```
count_obs(c(1,3,5,7,9,NA)) # 5
```



---

day	<i>Return the day of a Date value</i>
-----	---------------------------------------

---

**Description**

Return the day of a Date value as an integer

**Usage**

```
day(date)
```

**Arguments**

date	date value formatted as.Date()
------	--------------------------------

**Value**

the day of the date (integer)

**Examples**

```
day(as.Date("2019-09-15")) ## 15
```

---

Diff	<i>Difference data</i>
------	------------------------

---

**Description**

Wrapper for 'diff()' maintaining the same number of observations in 'x'

**Usage**

```
Diff(x, lag = 1)
```

**Arguments**

x	data
lag	number of lags to use

**Value**

Differenced data

**Examples**

```
Diff(c(100,50,100,20,100,110))
```

---

end_of_period	<i>End of period date</i>
---------------	---------------------------

---

**Description**

Return the date of the last day of the period (week, month, quarter, year). Weekly dates are indexed to Friday.

**Usage**

```
end_of_period(dates, period = c("month", "week", "quarter", "year"), shift = 0)
```

**Arguments**

dates	Date values formatted as.Date()
period	One of 'month', 'week', 'quarter', 'year'.
shift	Integer, shift date forward (positive values) or backwards (negative values) by the number of periods.

**Value**

Last day of period in as.Date() format

**Examples**

```
end_of_period(as.Date("2019-09-15")) ## 2019-09-30
```

---

end_of_year	<i>End of Year</i>
-------------	--------------------

---

**Description**

Find the end of year for a vector of dates

**Usage**

```
end_of_year(dates)
```

**Arguments**

dates	Transition matrix from a VAR model
-------	------------------------------------

**Value**

The last day of the year for the dates

**Examples**

```
end_of_year(as.Date("2019-09-15")) ## 2019-12-31
```

---

`extract_basic_character`

*Extract characters*

---

**Description**

Extract character values from x excluding space and underscore

**Usage**

```
extract_basic_character(x)
```

**Arguments**

x                    object containing character (and other) values

**Value**

Character values without space and underscore

**Examples**

```
extract_basic_character(c("this_one", "abc123")) ## c("thisone", "abc123")
```

---

`extract_character`

*Extract character values*

---

**Description**

Extract character values from x including space and underscore

**Usage**

```
extract_character(x)
```

**Arguments**

x                    object containing character values

**Value**

Character value from the object

**Examples**

```
extract_character(c("this_one", "abc123")) ## c("this_one", "abc")
```

---

extract_numeric	<i>Extract numeric values</i>
-----------------	-------------------------------

---

**Description**

Extract numeric values from x

**Usage**

```
extract_numeric(x)
```

**Arguments**

x                    object containing numeric (and other) values

**Value**

Numeric values from the object

**Examples**

```
extract_numeric(c("7+5", "abc123")) ## c(75, 123)
```

---

fill_forward	<i>Fill Forward</i>
--------------	---------------------

---

**Description**

Fill missing observations forward using the last finite observation

**Usage**

```
fill_forward(x)
```

**Arguments**

x                    Transition matrix from a VAR model

**Value**

x with missing obs filled by forward value

**Examples**

```
fill_forward(c(1,2,NA,NA,3,NA,5)) ## 1 2 2 2 3 3 5
```

---

first_of_month	<i>First of month</i>
----------------	-----------------------

---

**Description**

Return the first day of the month for each date in 'dates'

**Usage**

```
first_of_month(dates)
```

**Arguments**

dates            A sequence of dates in 'as.Date()' format

**Value**

First day of the month

**Examples**

```
dates <- seq.Date(from = as.Date("2020-09-11"),
                  by = "day", length.out = 10)
first_of_month(dates)
```

---

first_of_quarter	<i>First of Quarter</i>
------------------	-------------------------

---

**Description**

Find the first date in the quarter for a vector of dates

**Usage**

```
first_of_quarter(dates)
```

**Arguments**

dates            Transition matrix from a VAR model

**Value**

The first day of the quarter for the dates

**Examples**

```
first_of_quarter(as.Date("2019-9-15")) ## 2019-07-01
```

---

first\_previous\_quarter

*First of previous quarter date*

---

**Description**

Return the date of the first day of the previous quarter

**Usage**

```
first_previous_quarter(date)
```

**Arguments**

date                    date value formatted as.Date()

**Value**

The first day of the previous quarter of the date

**Examples**

```
first_previous_quarter(as.Date("2019-09-15")) ## 2019-04-01
```

---

fred

*Sample mixed frequency data from FRED*

---

**Description**

Sample mixed frequency data from FRED

**Author(s)**

Seth Leonard <seth@macroeconomicdata.com>

**References**

<https://fred.stlouisfed.org/>

---

fredlib	<i>Library of metadata for mixed frequency dataset 'fred'</i>
---------	---

---

**Description**

Library of metadata for mixed frequency dataset 'fred'

**Author(s)**

Seth Leonard <seth@macroeconomicdata.com>

**References**

<https://fred.stlouisfed.org/>

---

get_data_frq	<i>Get frequency of data based on missing observations</i>
--------------	--

---

**Description**

Guess the frequency of a data series based on the pattern of missing observations

**Usage**

```
get_data_frq(x = NULL, dates)
```

**Arguments**

x	data, potentially with missing observations
dates	corresponding dates in 'as.Date()' format

**Value**

The frequency of the data

**Examples**

```
dates <- as.Date(c("2020-1-1", "2020-1-15", "2020-2-1",  
                  "2020-2-15", "2020-3-1", "2020-3-15", "2020-4-1"))  
get_data_frq(c(1,NA,2,NA,3,NA,4), dates) ## "month"
```

get\_from\_list      *Get from list*

---

**Description**

Retrieve object 'what' from 'lst'

**Usage**

```
get_from_list(lst, what)
```

**Arguments**

lst	list
what	object to retrieve (by name or index)

**Value**

Element of the list indicated

**Examples**

```
get_from_list(list("a" = "alpha", "b" = c(1,2,3)), "a") # "alpha"
```

---

index\_by\_friday      *Find the Friday in a given week*

---

**Description**

Find the Friday in a given week from a sequence of Dates Vectors should be in as.Date() format

**Usage**

```
index_by_friday(dates)
```

**Arguments**

dates	vector of dates
-------	-----------------

**Value**

The date of the Friday in the week of the given date

**Examples**

```
dates <- seq.Date(from = as.Date("2020-09-21"),  
                 by = "week", length.out = 10)  
fridays <- index_by_friday(dates)  
weekdays(fridays)
```



---

is_in	<i>Find element of this_in that</i>
-------	-------------------------------------

---

**Description**

Find element of this\_in that, ie 'this\_in

**Usage**

```
is_in(that, this_in)
```

**Arguments**

that	first object
this_in	second object

**Value**

Logical variables indicating whether the element exists in both objects

**Examples**

```
that <- seq.Date(from = as.Date("2020-09-15"), by = "day", length.out = 10)
this_in <- seq.Date(from = as.Date("2020-09-11"), by = "day", length.out = 10)
is_in(that, this_in)
```

---

last_in_month	<i>Last date in the month</i>
---------------	-------------------------------

---

**Description**

Return the latest date in each month for the values in 'dates'

**Usage**

```
last_in_month(dates)
```

**Arguments**

dates	A sequence of dates in 'as.Date()' format
-------	---

**Value**

Last day of each month

**Examples**

```
dates <- seq.Date(from = as.Date("2020-09-11"),
                 by = "day", length.out = 10)
last_in_month(dates)
```

---

<code>last_in_quarter</code>	<i>Last date in the week</i>
------------------------------	------------------------------

---

**Description**

Return the latest date in the quarter for the values in 'dates'

**Usage**

```
last_in_quarter(dates)
```

**Arguments**

`dates`            A sequence of dates in 'as.Date()' format

**Value**

Last day of the quarter

**Examples**

```
dates <- seq.Date(from = as.Date("2020-09-11"),
                 by = "day", length.out = 10)
last_in_quarter(dates)
```

---

<code>last_in_week</code>	<i>Last date in the week</i>
---------------------------	------------------------------

---

**Description**

Return the latest date in each week for the values in 'dates'

**Usage**

```
last_in_week(dates)
```

**Arguments**

`dates`            A sequence of dates in 'as.Date()' format

**Value**

Last day of each week

**Examples**

```
dates <- seq.Date(from = as.Date("2020-09-21"),
                 by = "day", length.out = 10)
last_in_week(dates)
```

---

last_in_year	<i>Last date in the year</i>
--------------	------------------------------

---

**Description**

Return the latest date in each year for the values in 'dates'

**Usage**

```
last_in_year(dates)
```

**Arguments**

dates            A sequence of dates in 'as.Date()' format

**Value**

Last day of the year

**Examples**

```
dates <- seq.Date(from = as.Date("2020-09-11"),
                 by = "day", length.out = 10)
last_in_year(dates)
```

---

last_obs	<i>Last observation</i>
----------	-------------------------

---

**Description**

Return the last finite observation of 'x'

**Usage**

```
last_obs(x)
```

**Arguments**

x                    data potentially with non-finite values

**Value**

The last finite observation

**Examples**

```
last_obs(c(NA,1,2,3,NA,5,NA,7,NA,NA)) ## 7
```

---

limit\_character            *Limit Characters*

---

**Description**

limit the number of characters in a string and remove spacial characters (will not drop numbers)

**Usage**

```
limit_character(x, limit = 100)
```

**Arguments**

x                    object containing character values  
limit                maximum number of characters to return

**Value**

Character values within the limit

**Examples**

```
limit_character("a%b+&cd!efghij",limit = 3) ## "abc"
```

---

long_run_var	<i>Long Run Variance of a VAR</i>
--------------	-----------------------------------

---

**Description**

Find the long run variance of a VAR using the transition equation 'A' and shocks to observations 'Q'.

**Usage**

```
long_run_var(A, Q, m, p)
```

**Arguments**

A	Transition matrix from a VAR model in companion form
Q	Covariance of shocks
m	Number of series in the VAR
p	Number of lags in the VAR

**Value**

The variance matrix

**Examples**

```
long_run_var(comp_form(matrix(c(.2,.1,.1,.2,0,0,0,0), 2, 4)),
             matrix(c(1,0,0,0,0,1,0,0,0,0,0,0,0,0,0),4,4),2, 2)
```

---

match_index	<i>Match index values</i>
-------------	---------------------------

---

**Description**

Match index values of this to that

**Usage**

```
match_index(this, that)
```

**Arguments**

this	first object
that	second object

**Value**

A list of indexes indicating the elements that are matched to each other

**Examples**

```
match_index(c(1,2,3),c(2,3,4)) ## $that_idx: 1 2; $this_idx: 2 3
```

---

match_ts_dates	<i>Match dates between two timeseries</i>
----------------	---

---

**Description**

Find values in 'new\_ts' that correspond to dates in 'old\_ts'

**Usage**

```
match_ts_dates(old_ts, new_ts)
```

**Arguments**

old_ts	timeseries data
new_ts	timeseries data

**Value**

Timeseries data in which 'new\_ts' corresponds to 'old\_ts'

**Examples**

```
old_ts <- ts(c(1,2,3,4), start=c(2020,1), end=c(2020,4), frequency=4)
new_ts <- ts(c(5,6,3,4), start=c(2019,4), end=c(2020,3), frequency=4)
match_ts_dates(old_ts, new_ts)
```

---

mean_na	<i>Return the mean</i>
---------	------------------------

---

**Description**

Return the mean of 'x'. If no observations, return 'NA'. This is a workaround for the fact that in data.table, ':= mean(x, na.rm = TRUE)' will return 'NaN' where there are no observations

**Usage**

```
mean_na(x)
```

**Arguments**

x                      data potentially with non-finite values

**Value**

Mean of the input

**Examples**

```
mean_na(c(1,2,3,7,9,NA)) ## 4.4
```

---

month_days	<i>Number of days in a given month</i>
------------	--

---

**Description**

Get the number of days in a month given the year and month

**Usage**

```
month_days(year, month)
```

**Arguments**

year                      integer year value  
month                      integer month value

**Value**

The number of days in the month (integer)

**Examples**

```
month_days(2021,9) ## 30  
month_days(2020,2) ## 29
```

number\_finite                      *Number of finite values in a column*

---

**Description**

Return the number of finite values in a column of Y

**Usage**

```
number_finite(Y)
```

**Arguments**

Y                      matrix like data object

**Value**

The number of finite values per column

**Examples**

```
X <- matrix(1,10,2)
X[3,1] <- NA
number_finite(X)
```

---

numdum                      *Dummies for Numeric Data*

---

**Description**

Create dummy variables for unique numeric values in 'x'

**Usage**

```
numdum(x)
```

**Arguments**

x                      Numeric vector

**Value**

Dummy variables for each unique value in the data

**Examples**

```
numdum(c(3,3,5,3,4,3,5,4,4,5)) ## dummies for each of 3, 4, and 5
```



---

pct_chng	<i>Percent change</i>
----------	-----------------------

---

**Description**

Calculate the percent change in 'y' from one period to the next

**Usage**

```
pct_chng(y, lag = 1)
```

**Arguments**

y	data
lag	number of periods for percent change

**Value**

The percentage change among the lag period

**Examples**

```
pct_chng(c(100,50,100,20,100,110))
```

---

pct_response	<i>Percent of responses at a given frequency</i>
--------------	--

---

**Description**

Return the percent of responses to categorical answers at a specified frequency

**Usage**

```
pct_response(
  dt,
  col_name = NULL,
  by = c("month", "quarter", "week"),
  date_name = "ref_date"
)
```

**Arguments**

dt	data table of responses
col_name	name of column containing responses
by	frequency of response aggregation, one of "month", "quarter", "week"
date_name	name of column containing dates

**Value**

The percent of responses at the frequency

**Examples**

```
dt <- data.frame("ref_date" = seq.Date(as.Date("2000-01-01"), length.out = 100, by = "week"),
                "response" = c(rep("yes", 20), rep("no", 50), rep("yes", 30)))
out <- pct_response(dt, col_name = "response")
```

---

process

*Process Data*

---

**Description**

Process data to ensure stationarity in long format for time series modeling

**Usage**

```
process(
  dt,
  lib,
  detrend = TRUE,
  center = TRUE,
  scale = TRUE,
  as_of = NULL,
  date_name = "ref_date",
  id_name = "series_name",
  value_name = "value",
  pub_date_name = NULL,
  ignore_numeric_names = TRUE,
  silent = FALSE
)
```

**Arguments**

dt	Data in long format.
lib	Library with instructions regarding how to process data; see details.
detrend	T/F should data be detrended (see details)?
center	T/F should data be centered (i.e. de-meanned)?
scale	T/F should data be scaled (i.e. variance 1)?
as_of	"As of" date at which to censor observations for backesting. This requires 'pub_date_name' is specified.
date_name	Name of data column in the data.
id_name	Name of ID column in the data.
value_name	Name of value column in the data

pub_date_name	Name of publication date column in the data; required if 'as_of' specified.
ignore_numeric_names	T/F ignore numeric values in matching series names in 'dt' to series names in 'lib'. This is required for data aggregated using 'process_MF()', as lags of LHS and RHS data are tagged 0 for contemporaneous data, 1 for one lag, 2 for 2 lags, etc. Ignoring these tags insures processing from 'lib' is correctly identified.
silent	T/F, supress warnings?

## Details

Process data can be used to transform data to insure stationarity and to censor data for backtesting. Directions for processing each file come from the data.table 'lib'. This table must include the columns 'series\_name', 'take\_logs', and 'take\_diffs'. Unique series may also be identified by a combination of 'country' and 'series\_name'. Optional columns include 'needs\_SA' for series that need seasonal adjustment, 'detrend' for removing low frequency trends (nowcasting only; detrend should not be used for long horizon forecasts), 'center' to de-mean the data, and 'scale' to scale the data. If the argument to 'process\_wide()' of 'detrend', 'center', or 'scale' is 'FALSE', the operation will not be performed. If 'TRUE', the function will check for the column of the same name in 'lib'. If the column exists, T/F entries from this column are used to determine which series to transform. If the column does not exist, all series will be transformed.

## Value

data.table of processed values in long format.

## Examples

```
dt <- process(fred, fredlib)

LHS <- fred[series_name == "gdp constant prices"]
RHS <- fred[series_name != "gdp constant prices"]
dtQ <- process_MF(LHS, RHS)
dt_processed <- process(dtQ, fredlib)
```

---

process\_MF

*Process mixed frequency*

---

## Description

Process mixed frequency data for nowcasting applications by identifying the missing observations in the contemporaneous data and replicating this pattern of missing observations in the historical data prior to aggregation. This allows the incorporation of all available information into the model while still using uniform frequency models to actually generate predictions, and can thus be applied to a wide array of econometrics and machine learning applications.

**Usage**

```

process_MF(
  LHS,
  RHS,
  LHS_lags = 1,
  RHS_lags = 1,
  as_of = NULL,
  frq = c("auto", "week", "month", "quarter", "year"),
  date_name = "ref_date",
  id_name = "series_name",
  value_name = "value",
  pub_date_name = "pub_date",
  return_dt = TRUE
)

```

**Arguments**

LHS	Left hand side data in long format. May include multiple LHS variables, but LHS variance MUST have the same frequency.
RHS	Right hand side data in long format at any frequency.
LHS_lags	Number of lags of LHS variables to include in output.
RHS_lags	Number of lags of RHS variables to include in output (may be 0, indicating contemporaneous values only).
as_of	Backtesting the model "as of" this date; requires that 'pub_date' is specified in the data
frq	Frequency of LHS data, one of 'week', 'month', 'quarter', 'year'. If not specified, the function will attempt to automatically identify the frequency.
date_name	Name of date column in data.
id_name	Name of ID column in the data.
value_name	Name of value column in the data.
pub_date_name	Name of publication date in the data.
return_dt	T/F, should the function return a 'data.table'? IF FALSE the function will return matrix data.

**Details**

Right hand side data will always include observations contemporaneous with LHS data. Use 'RHS\_lags' to add lags of RHS data to the output, and 'LHS\_lags' to add lags of LHS data to the output. By default the function will return data in long format designed to be used with the 'dateutils' function 'process()'. Specifying 'return\_dt = FALSE' will return LHS variables in the matrix 'Y', RHS variables in the matrix 'X', and corresponding dates (by index) in the date vector 'dates'.

**Value**

data.table in long format (unless 'return\_dt = FALSE'). Variables ending in '0' are contemporaneous, ending in '1' are at one lag, '2' at two lags, etc.

**Examples**

```
LHS <- fred[series_name == "gdp constant prices"]
RHS <- fred[series_name != "gdp constant prices"]
dt <- process_MF(LHS, RHS)
```

process\_wide

*Process Wide Format Data***Description**

Process data in wide format for time series modeling

**Usage**

```
process_wide(
  dt_wide,
  lib,
  detrend = TRUE,
  center = TRUE,
  scale = TRUE,
  date_name = "ref_date",
  ignore_numeric_names = TRUE,
  silent = FALSE
)
```

**Arguments**

dt_wide	Data in wide format.
lib	Library with instructions regarding how to process data; see details.
detrend	T/F should data be detrended (see details)?
center	T/F should data be centered (i.e. de-meanned)?
scale	T/F should data be scaled (i.e. variance 1)?
date_name	Name of data column in the data.
ignore_numeric_names	T/F ignore numeric values in matching series names in 'dt' to series names in 'lib'. This is required for data aggregated using 'process_MF()', as lags of LHS and RHS data are tagged 0 for contemporaneous data, 1 for one lag, 2 for 2 lags, etc. Ignoring these tags insures processing from 'lib' is correctly identified.
silent	T/F, suppress warnings?

**Details**

'process\_wide()' can be used to transform wide data to insure stationarity. Censoring by pub\_date requires long format. Directions for processing each file come from the data.table 'lib'. This table must include the columns 'series\_name', 'take\_logs', and 'take\_diffs'. Unique series may also be identified by a combination of 'country' and 'series\_name'. Optional columns include 'needs\_SA' for series that need seasonal adjustment, 'detrend' for removing low frequency trends (nowcasting only; 'detrend' should not be used for long horizon forecasts), 'center' to de-mean the data, and 'scale' to scale the data. If the argument to 'process\_wide()' of 'detrend', 'center', or 'scale' is 'FALSE', the operation will not be performed. If 'TRUE', the function will check for the column of the same name in 'lib'. If the column exists, T/F entries from this column are used to determine which series to transform. If the column does not exist, all series will be transformed.

**Value**

data.table of processed data

**Examples**

```
LHS <- fred[series_name == "gdp constant prices"]
RHS <- fred[series_name != "gdp constant prices"]
dtQ <- process_MF(LHS, RHS)
dt_wide <- data.table::dcast(dtQ, ref_date ~ series_name, value.var = "value")
dt_processed <- process_wide(dt_wide, fredlib)
```

---

rollmax

*Rolling Max*

---

**Description**

Find the rolling maximum in 'x' with span 'n'

**Usage**

```
rollmax(x, n)
```

**Arguments**

x	Numeric vector
n	Integer span

**Value**

The maximum value of 'x' with span 'n'

**Examples**

```
rollmax(c(1,2,3), 2) ## c(2,3,3)
```

---

rollmean	<i>Rolling mean</i>
----------	---------------------

---

**Description**

Take the rolling mean of 'x' over 'n' elements

**Usage**

```
rollmean(x, n)
```

**Arguments**

x	data vector
n	span of rolling mean

**Value**

Rolling mean of the input

**Examples**

```
rollmean(c(1,2,3),2) ## NA, 1.5, 2.5
```

---

rollmin	<i>Rolling Min</i>
---------	--------------------

---

**Description**

Find the rolling minimum in 'x' with span 'n'

**Usage**

```
rollmin(x, n)
```

**Arguments**

x	Numeric vector
n	Integer span

**Value**

The minimum value of 'x' with span 'n'

**Examples**

```
rollmin(c(1,2,3),2) ## c(1,1,2)
```

---

row_to_list	<i>Convert rows to list</i>
-------------	-----------------------------

---

**Description**

Return 'Y' with each row as a list

**Usage**

```
row_to_list(Y)
```

**Arguments**

Y                    matrix like data object

**Value**

Each row as a list

**Examples**

```
row_to_list(matrix(rnorm(20),10,2))
```

---

run_sa	<i>Seasonally adjust data using seas()</i>
--------	--

---

**Description**

Seasonally adjust monthly or quarterly data using X-13 SEATS via seas()

**Usage**

```
run_sa(x, dates, x11 = FALSE, transfunc = c("none", "auto", "log"))
```

**Arguments**

x                    data  
 dates                dates corresponding to data 'x'  
 x11                  T/F, use x11 as opposed to X-13 SEATS  
 transfunc            Data transformation, one of 'none' for no transformation, 'auto' for automatic detection, or 'log' for log transformation

**Value**

A list with 'adj\_fact' containing seasonal factors and 'sa\_final' containing seasonally adjusted data.



**Examples**

```
x <- fred[series_name == "gdp constant prices", value]
dates <- fred[series_name == "gdp constant prices", ref_date ]
run_sa(x, dates, transfunc = "log")
```

---

sd_na	<i>Return the standard deviation</i>
-------	--------------------------------------

---

**Description**

Return the standard deviation of 'x'. If no observations, return 'NA'. This is a workaround for the fact that in data.table, ':= sd(x, na.rm = TRUE)' will return 'NaN' where there are no observations

**Usage**

```
sd_na(x)
```

**Arguments**

x                      data potentially with non-finite values

**Value**

Standard deviation of the input

**Examples**

```
sd_na(c(1,2,3,NA)) ## 1
```

---

seas_df_long	<i>Seasonally adjust long format data using seas()</i>
--------------	--

---

**Description**

Seasonally adjust multiple monthly or quarterly series in long format using X-13 SEATS via seas()

**Usage**

```
seas_df_long(
  df,
  sa_names,
  x11 = FALSE,
  transfunc = "none",
  series_names = "series_name",
  value_var = "value",
  date_var = "ref_date"
)
```

**Arguments**

df	long format dataframe
sa_names	names of series to seasonally adjust
x11	T/F, use x11 as opposed to X-13 SEATS
transfunc	Data transformation, one of 'none' for no transformation, 'auto' for automatic detection, or 'log' for log transformation
series_names	name of column containing series names
value_var	name of column containing values
date_var	name of column containing dates

**Value**

A list with data.frames 'sa\_factors' containing seasonal factors and 'values\_sa' containing seasonally adjusted data.

**Examples**

```
seas_df_long(fred[series_name == "gdp constant prices"], sa_names="value")
```

---

seas_df_wide	<i>Seasonally adjust wide format data using seas()</i>
--------------	--

---

**Description**

Seasonally adjust multiple monthly or quarterly series in wide format using X-13 SEATS via seas()

**Usage**

```
seas_df_wide(df, sa_cols, x11 = FALSE, transfunc = "none")
```

**Arguments**

df	wide format dataframe
sa_cols	names or column indexes of series to seasonally adjust
x11	T/F, use x11 as opposed to X-13 SEATS
transfunc	Data transformation, one of 'none' for no transformation, 'auto' for automatic detection, or 'log' for log transformation

**Value**

A list with data.frames 'sa\_factors' containing seasonal factors and 'values\_sa' containing seasonally adjusted data.

**Examples**

```
seas_df_wide(fred[series_name == "gdp constant prices"], sa_cols="value")
```

---

spline_fill	<i>Spline fill missing observations</i>
-------------	---

---

**Description**

Spline fill missing observations from the first observation to the last, leaving NA observations in the head and tail

**Usage**

```
spline_fill(x)
```

**Arguments**

x                    data with missing observations

**Value**

data with interpolated missing observations, except at head and tail, which remain NA

**Examples**

```
spline_fill_trend(c(NA,1,2,3,NA,5)) ## NA 1 2 3 4 5
```

---

spline_fill_trend	<i>Spline fill missing observations</i>
-------------------	---

---

**Description**

Spline fill missing observations, designed for filling low frequency trend estimates

**Usage**

```
spline_fill_trend(x)
```

**Arguments**

x                    data with missing observations

**Value**

data with interpolated missing observations

**Examples**

```
spline_fill_trend(c(1,2,3,NA,5)) ## 1 2 3 4 5
```

---

stack_obs	<i>Stack time series observations in VAR format</i>
-----------	---

---

**Description**

Stack time series observations in VAR format over series for p lags

**Usage**

```
stack_obs(Dat, p)
```

**Arguments**

Dat	Data in a format convertible to a matrix
p	number of lags, integer value

**Value**

stacked time series obs with p lags

**Examples**

```
mat <- matrix(rnorm(100),50,2)
Z <- stack_obs(mat, 2) ## stack the dataset `mat` with two lags
## Note: one "lag" will just return the original dataset.
```

---

sum_na	<i>Return the sum</i>
--------	-----------------------

---

**Description**

Return the sum of 'x'. If no observations, return 'NA'. This is a workaround for the fact that in data.table, ':= sum()' will return 'NaN' where there are no observations

**Usage**

```
sum_na(x)
```

**Arguments**

x	data potentially with non-finite values
---	---

**Value**

Sum of the input

**Examples**

```
sum_na(c(1,2,3,NA)) # 6
```

---

total_response	<i>Number of of responses at a given frequency</i>
----------------	--

---

**Description**

Return the total number of responses to categorical answers at a specified frequency

**Usage**

```
total_response(
  dt,
  col_name = NULL,
  by = c("month", "quarter", "week"),
  date_name = "ref_date"
)
```

**Arguments**

dt	data table of responses
col_name	name of column containing responses
by	frequency of response aggregation, one of "month", "quarter", "week"
date_name	name of column containing dates

**Value**

The number of responses at the frequency

**Examples**

```
dt <- data.frame("ref_date" = seq.Date(as.Date("2000-01-01"), length.out = 100, by = "week"),
  "response" = c(rep("yes", 20), rep("no", 50), rep("yes", 30)))
out <- total_response(dt, col_name = "response")
```

---

to_ts	<i>Tabular data to ts() format</i>
-------	------------------------------------

---

**Description**

transform data in 'x' corresponding to dates in 'dates' to ts() format

**Usage**

```
to_ts(x, dates)
```

**Arguments**

x	data
dates	dates

**Value**

data in ts() format

**Examples**

```
x <- c(1,2,3,4)
dates <- as.Date(c("2020-1-1", "2020-2-1", "2020-3-1", "2020-4-1"))
to_ts(x, dates)
```

---

try_detrend	<i>Remove low frequency trends from data</i>
-------------	--

---

**Description**

Estimate low frequency trends via loess regression and remove them. If the function errors, return x (i.e. no trend)

**Usage**

```
try_detrend(x, outlier_rm = TRUE, span = 0.6)
```

**Arguments**

x	data
outlier_rm	T/F, remove outliers to estimate trends?
span	span for the loess regression

**Value**

Data with trends removed

**Examples**

```
try_detrend(c(1,3,6,7,9,11,14,15,17,18))
```

---

try_sa	<i>Seasonally adjust data using seas()</i>
--------	--

---

**Description**

Seasonally adjust monthly or quarterly data using X-13 SEATS via seas()

**Usage**

```
try_sa(x, dates, x11 = FALSE, transfunc = "none", series_name = NULL)
```

**Arguments**

x	data
dates	dates corresponding to data 'x'
x11	T/F, use x11 as opposed to X-13 SEATS
transfunc	Data transformation, one of 'none' for no transformation, 'auto' for automatic detection, or 'log' for log transformation
series_name	Include series name to print out if failure (for lapply() applications)

**Value**

A list with 'adj\_fact' containing seasonal factors and 'sa\_final' containing seasonally adjusted data. If seasonal adjustment failed 'adj\_fact' will contain zeros and 'sa\_final' will contain the original data.

**Examples**

```
x <- fred[series_name == "gdp constant prices", value]
dates <- fred[series_name == "gdp constant prices", ref_date ]
try_sa(x, dates, transfunc = "log")
```

---

try_trend	<i>Estimate low frequency trends</i>
-----------	--------------------------------------

---

**Description**

Estimate low frequency trends via loess regression. If the function errors, return zeros (i.e. no trend)

**Usage**

```
try_trend(x, outlier_rm = TRUE, span = 0.6)
```

**Arguments**

x                    data  
 outlier\_rm        T/F, remove outliers to estimate trends?  
 span                span for the loess regression

**Value**

Estimated trend in the data

**Examples**

```
try_trend(c(1,3,6,7,9,11,14,15,17,18))
```

---

ts_to_df	<i>ts() data to a dataframe</i>
----------	---------------------------------

---

**Description**

Transform monthly or quarterly ts() data to a dataframe

**Usage**

```
ts_to_df(x, end_period = TRUE)
```

**Arguments**

x                    ts() format data which is either monthly or quarterly  
 end\_period        T/F, for monthly or quarterly data, should dates be indexed to the end of the period?

**Value**

Data in dataframe format

**Examples**

```
x <- ts(c(1,2,3,4), start=c(2020,1), end=c(2020,4), frequency=4)
ts_to_df(x)
```



# Index

- \* **data**
  - fred, 14
  - fredlib, 15
- add\_forecast\_dates, 3
- agg\_to\_freq, 4
- agg\_to\_freq\_wide, 4
- all\_finite, 6
- allNA, 5
- any\_finite, 6
- can\_seasonal, 7
- col\_to\_list, 7
- comp\_form, 8
- count\_obs, 8
- day, 9
- Diff, 9
- end\_of\_period, 10
- end\_of\_year, 10
- extract\_basic\_character, 11
- extract\_character, 11
- extract\_numeric, 12
- fill\_forward, 12
- first\_of\_month, 13
- first\_of\_quarter, 13
- first\_previous\_quarter, 14
- fred, 14
- fredlib, 15
- get\_data\_frq, 15
- get\_from\_list, 16
- index\_by\_friday, 16
- is\_in, 17
- last\_in\_month, 17
- last\_in\_quarter, 18
- last\_in\_week, 18
- last\_in\_year, 19
- last\_obs, 19
- limit\_character, 20
- long\_run\_var, 21
- match\_index, 21
- match\_ts\_dates, 22
- mean\_na, 22
- month\_days, 23
- number\_finite, 24
- numdum, 24
- pct\_chng, 25
- pct\_response, 25
- process, 26
- process\_MF, 27
- process\_wide, 29
- rollmax, 30
- rollmean, 31
- rollmin, 31
- row\_to\_list, 32
- run\_sa, 32
- sd\_na, 33
- seas\_df\_long, 33
- seas\_df\_wide, 34
- spline\_fill, 35
- spline\_fill\_trend, 35
- stack\_obs, 36
- sum\_na, 36
- to\_ts, 37
- total\_response, 37
- try\_detrend, 38
- try\_sa, 39
- try\_trend, 39
- ts\_to\_df, 40