Package ‘anomalize’

February 11, 2019

Type Package

Title Tidy Anomaly Detection

Version 0.1.1

Description
The 'anomalize' package enables a ``tidy'' workflow for detecting anomalies in data. The main functions are time_decompose(), anomalize(), and time_recompose(). When combined, it's quite simple to decompose time series, detect anomalies, and create bands separating the ``normal'' data from the anomalous data at scale (i.e. for multiple time series). Time series decomposition is used to remove trend and seasonal components via the time_decompose() function and methods include seasonal decomposition of time series by Loess (``stl'') and seasonal decomposition by piecewise medians (``twitter''). The anomalize() function implements two methods for anomaly detection of residuals including using an inner quartile range (``iqr'') and generalized extreme studentized deviation (``gesd''). These methods are based on those used in the 'forecast' package and the Twitter 'AnomalyDetection' package. Refer to the associated functions for specific references for these methods.

URL https://github.com/business-science/anomalize

BugReports https://github.com/business-science/anomalize/issues

License GPL (¿= 3)

Encoding UTF-8

LazyData true

Depends R (¿= 3.0.0)

Imports dplyr, glue, timetk, sweep, tibbletime, purrr, rlang, tibble, tidyr, ggplot2

RoxygenNote 6.0.1

Suggests tidyverse, tidyquant, testthat, covr, knitr, rmarkdown, devtools, roxygen2

VignetteBuilder knitr

NeedsCompilation no

Author Matt Dancho [aut, cre], Davis Vaughan [aut]

Maintainer Matt Dancho <mdancho@business-science.io>

Repository CRAN

Date/Publication 2018-04-17 11:51:22 UTC

R topics documented:

`anomalize` *Detect anomalies using the tidyverse*

Description

Detect anomalies using the tidyverse

Usage

```
anomalize(data, target, method = c("iqr", "gesd"), alpha = 0.05,
  max_anoms = 0.2, verbose = FALSE)
```

Arguments

`data`	A `tibble` or `tbl_time` object.
`target`	A column to apply the function to
`method`	The anomaly detection method. One of `"iqr"` or `"gesd"`. The IQR method is faster at the expense of possibly not being quite as accurate. The GESD method has the best properties for outlier detection, but is loop-based and therefore a bit slower.
`alpha`	Controls the width of the "normal" range. Lower values are more conservative while higher values are less prone to incorrectly classifying "normal" observations.
`max_anoms`	The maximum percent of anomalies permitted to be identified.
`verbose`	A boolean. If `TRUE`, will return a list containing useful information about the anomalies. If `FALSE`, just returns the data expanded with the anomalies and the lower (l1) and upper (l2) bounds.

Details

The `anomalize()` function is used to detect outliers in a distribution with no trend or seasonality present. The return has three columns: "remainder_l1" (lower limit for anomalies), "remainder_l2" (upper limit for anomalies), and "anomaly" (Yes/No).

Use `time_decompose()` to decompose a time series prior to performing anomaly detection with `anomalize()`. Typically, `anomalize()` is performed on the "remainder" of the time series decomposition.

For non-time series data (data without trend), the `anomalize()` function can be used without time series decomposition.

The `anomalize()` function uses two methods for outlier detection each with benefits.

IQR:

The IQR Method uses an innerquartile range of 25 the median. With the default `alpha = 0.05`, the limits are established by expanding the 25/75 baseline by an IQR Factor of 3 (3X). The IQR Factor = 0.15 / alpha (hense 3X with alpha = 0.05). To increase the IQR Factor controling the limits, decrease the alpha, which makes it more difficult to be an outlier. Increase alpha to make it easier to be an outlier.

The IQR method is used in forecast::tsoutliers().

GESD:

The GESD Method (Generlized Extreme Studentized Deviate Test) progressively eliminates outliers using a Student's T-Test comparing the test statistic to a critical value. Each time an outlier is removed, the test statistic is updated. Once test statistic drops below the critical value, all outliers are considered removed. Because this method involves continuous updating via a loop, it is slower than the IQR method. However, it tends to be the best performing method for outlier removal.

The GESD method is used in AnomalyDection::AnomalyDetectionTs().

Value

Returns a `tibble` / `tbl_time` object or list depending on the value of `verbose`.

References

1. How to correct outliers once detected for time series data forecasting? Cross Validated, https://stats.stackexchange.com
2. Cross Validated: Simple algorithm for online outlier detection of a generic time series. Cross Validated, https://stats.stackexchange.com
3. Owen S. Vallis, Jordan Hochenbaum and Arun Kejariwal (2014).A Novel Technique for Long-Term Anomaly Detection in the Cloud. Twitter Inc.
4. Owen S. Vallis, Jordan Hochenbaum and Arun Kejariwal (2014). AnomalyDetection: Anomaly Detection UsingSeasonal Hybrid Extreme Studentized Deviate Test. R package version 1.0.
5. Alex T.C. Lau (November/December 2015). GESD - A Robust and Effective Technique for Dealing with Multiple Outliers. ASTM Standardization News. www.astm.org/sn

See Also

Anomaly Detection Methods (Powers `anomalize`)

- `iqr()`

- gesd()

Time Series Anomaly Detection Functions (anomaly detection workflow):

- time_decompose()
- time_recompose()

Examples

```
library(dplyr)

# Needed to pass CRAN check / This is loaded by default
set_time_scale_template(time_scale_template())

data(tidyverse_cran_downloads)

tidyverse_cran_downloads %>%
    time_decompose(count, method = "stl") %>%
    anomalize(remainder, method = "iqr")
```

---

anomalize_methods *Methods that power anomalize()*

Description

Methods that power anomalize()

Usage

```
iqr(x, alpha = 0.05, max_anoms = 0.2, verbose = FALSE)

gesd(x, alpha = 0.05, max_anoms = 0.2, verbose = FALSE)
```

Arguments

x	A vector of numeric data.
alpha	Controls the width of the "normal" range. Lower values are more conservative while higher values are less prone to incorrectly classifying "normal" observations.
max_anoms	The maximum percent of anomalies permitted to be identified.
verbose	A boolean. If TRUE, will return a list containing useful information about the anomalies. If FALSE, just returns a vector of "Yes" / "No" values.

Value

Returns character vector or list depending on the value of verbose.

References

- The IQR method is used in forecast::tsoutliers()
- The GESD method is used in Twitter's AnomalyDetection package and is also available as a function in @raunakms's GESD method

See Also

anomalize()

Examples

```
set.seed(100)
x <- rnorm(100)
idx_outliers <- sample(100, size = 5)
x[idx_outliers] <- x[idx_outliers] + 10

iqr(x, alpha = 0.05, max_anoms = 0.2)
iqr(x, alpha = 0.05, max_anoms = 0.2, verbose = TRUE)

gesd(x, alpha = 0.05, max_anoms = 0.2)
gesd(x, alpha = 0.05, max_anoms = 0.2, verbose = TRUE)
```

---

anomalize_package   *anomalize: Tidy anomaly detection*

Description

anomalize: Tidy anomaly detection

Details

The 'anomalize' package enables a "tidy" workflow for detecting anomalies in data. The main functions are time_decompose(), anomalize(), and time_recompose(). When combined, it's quite simple to decompose time series, detect anomalies, and create bands separating the "normal" data from the anomalous data at scale (i.e. for multiple time series). Time series decomposition is used to remove trend and seasonal components via the time_decompose() function and methods include seasonal decomposition of time series by Loess and seasonal decomposition by piecewise medians. The anomalize() function implements two methods for anomaly detection of residuals including using an inner quartile range and generalized extreme studentized deviation. These methods are based on those used in the `forecast` package and the Twitter `AnomalyDetection` package. Refer to the associated functions for specific references for these methods.

To learn more about `anomalize`, start with the vignettes: `browseVignettes(package = "anomalize")` decompose_methods *Methods that power time_decompose()*

Description

Methods that power time_decompose()

Usage

```
decompose_twitter(data, target, frequency = "auto", trend = "auto",
  message = TRUE)

decompose_stl(data, target, frequency = "auto", trend = "auto",
  message = TRUE)
```

Arguments

data	A `tibble` or `tbl_time` object.
target	A column to apply the function to
frequency	Controls the seasonal adjustment (removal of seasonality). Input can be either "auto", a time-based definition (e.g. "2 weeks"), or a numeric number of observations per frequency (e.g. 10). Refer to `time_frequency()`.
trend	Controls the trend component For stl, the trend controls the sensitivity of the lowess smoother, which is used to remove the remainder. For twitter, the trend controls the period width of the median, which are used to remove the trend and center the remainder.
message	A boolean. If `TRUE`, will output information related to `tbl_time` conversions, frequencies, and trend / median spans (if applicable).

Value

A `tbl_time` object containing the time series decomposition.

References

- The "twitter" method is used in Twitter's AnomalyDetection package

See Also

`time_decompose()`

Examples

```
library(dplyr)

tidyverse_cran_downloads %>%
    ungroup() %>%
    filter(package == "tidyquant") %>%
    decompose_stl(count)
```

plot_anomalies *Visualize the anomalies in one or multiple time series*

Description

Visualize the anomalies in one or multiple time series

Usage

```
plot_anomalies(data, time_recomposed = FALSE, ncol = 1,
  color_no = "#2c3e50", color_yes = "#e31a1c", fill_ribbon = "grey70",
  alpha_dots = 1, alpha_circles = 1, alpha_ribbon = 1, size_dots = 1.5,
  size_circles = 4)
```

Arguments

`data`	A `tibble` or `tbl_time` object.
`time_recomposed`	A boolean. If `TRUE`, will use the `time_recompose()` bands to place bands as approximate limits around the "normal" data.
`ncol`	Number of columns to display. Set to 1 for single column by default.
`color_no`	Color for non-anomalous data.
`color_yes`	Color for anomalous data.
`fill_ribbon`	Fill color for the time_recomposed ribbon.
`alpha_dots`	Controls the transparency of the dots. Reduce when too many dots on the screen.
`alpha_circles`	Controls the transparency of the circles that identify anomalies.
`alpha_ribbon`	Controls the transparency of the time_recomposed ribbon.
`size_dots`	Controls the size of the dots.
`size_circles`	Controls the size of the circles that identify anomalies.

Details

Plotting function for visualizing anomalies on one or more time series. Multiple time series must be grouped using `dplyr::group_by()`.

Value

Returns a `ggplot` object.

See Also

`plot_anomaly_decomposition()`

Examples

```
library(dplyr)
library(ggplot2)

data(tidyverse_cran_downloads)

#### SINGLE TIME SERIES ####
tidyverse_cran_downloads %>%
    filter(package == "tidyquant") %>%
    ungroup() %>%
    time_decompose(count, method = "stl") %>%
    anomalize(remainder, method = "iqr") %>%
    time_recompose() %>%
    plot_anomalies(time_recomposed = TRUE)


#### MULTIPLE TIME SERIES ####
tidyverse_cran_downloads %>%
    time_decompose(count, method = "stl") %>%
    anomalize(remainder, method = "iqr") %>%
    time_recompose() %>%
    plot_anomalies(time_recomposed = TRUE, ncol = 3)
```

---

plot_anomaly_decomposition

Visualize the time series decomposition with anomalies shown

Description

Visualize the time series decomposition with anomalies shown

Usage

```
plot_anomaly_decomposition(data, ncol = 1, color_no = "#2c3e50",
  color_yes = "#e31a1c", alpha_dots = 1, alpha_circles = 1,
  size_dots = 1.5, size_circles = 4, strip.position = "right")
```

Arguments

data	A tibble or tbl_time object.
ncol	Number of columns to display. Set to 1 for single column by default.
color_no	Color for non-anomalous data.
color_yes	Color for anomalous data.
alpha_dots	Controls the transparency of the dots. Reduce when too many dots on the screen.
alpha_circles	Controls the transparency of the circles that identify anomalies.
size_dots	Controls the size of the dots.
size_circles	Controls the size of the circles that identify anomalies.
strip.position	Controls the placement of the strip that identifies the time series decomposition components.

Details

The first step in reviewing the anomaly detection process is to evaluate a single times series to observe how the algorithm is selecting anomalies. The `plot_anomaly_decomposition()` function is used to gain an understanding as to whether or not the method is detecting anomalies correctly and whether or not parameters such as decomposition method, anomalize method, alpha, frequency, and so on should be adjusted.

Value

Returns a `ggplot` object.

See Also

`plot_anomalies()`

Examples

```
library(dplyr)
library(ggplot2)

data(tidyverse_cran_downloads)

tidyverse_cran_downloads %>%
    filter(package == "tidyquant") %>%
    ungroup() %>%
    time_decompose(count, method = "stl") %>%
    anomalize(remainder, method = "iqr") %>%
    plot_anomaly_decomposition()
```

---

prep_tbl_time — *Automatically create tibbletime objects from tibbles*

Description

Automatically create tibbletime objects from tibbles

Usage

```
prep_tbl_time(data, message = FALSE)
```

Arguments

data	A `tibble`.
message	A boolean. If `TRUE`, returns a message indicating any conversion details important to know during the conversion to `tbl_time` class.

Details

Detects a date or datetime index column and automatically

Value

Returns a `tibbletime` object of class `tbl_time`.

Examples

```
library(dplyr)
library(tibbletime)

data_tbl <- tibble(
    date  = seq.Date(from = as.Date("2018-01-01"), by = "day", length.out = 10),
    value = rnorm(10)
    )

prep_tbl_time(data_tbl)
```

`set_time_scale_template` *Get and modify time scale template*

Description

Get and modify time scale template

Usage

```
set_time_scale_template(data)

get_time_scale_template()

time_scale_template()
```

Arguments

`data`	A `tibble` with a "time_scale", "frequency", and "trend" columns.

Details

Used to get and set the time scale template, which is used by `time_frequency()` and `time_trend()` when `period = "auto"`.

See Also

`time_frequency()`, `time_trend()`

Examples

```
get_time_scale_template()

set_time_scale_template(time_scale_template())
```

`tidyverse_cran_downloads` *Downloads of various "tidyverse" packages from CRAN*

Description

A dataset containing the daily download counts from 2017-01-01 to 2018-03-01 for the following tidyverse packages:

- `tidyr`
- `lubridate`
- `dplyr`
- `broom`
- `tidyquant`
- `tidytext`
- `ggplot2`
- `purrr`
- `stringr`
- `forcats`
- `knitr`
- `readr`
- `tibble`
- `tidyverse`

Usage

```
tidyverse_cran_downloads
```

Format

A `grouped_tbl_time` object with 6,375 rows and 3 variables:

date Date of the daily observation

count Number of downloads that day

package The package corresponding to the daily download number

Source

The package downloads come from CRAN by way of the `cranlogs` package.

time_apply — *Apply a function to a time series by period*

Description

Apply a function to a time series by period

Usage

```
time_apply(data, target, period, .fun, ..., start_date = NULL, side = "end",
  clean = FALSE, message = TRUE)
```

Arguments

`data`	A `tibble` with a date or datetime index.
`target`	A column to apply the function to
`period`	A time-based definition (e.g. "2 weeks"). or a numeric number of observations per frequency (e.g. 10). See `tibbletime::collapse_by()` for period notation.
`.fun`	A function to apply (e.g. `median`)
`...`	Additional parameters passed to the function, `.fun`
`start_date`	Optional argument used to specify the start date for the first group. The default is to start at the closest period boundary below the minimum date in the supplied index.
`side`	Whether to return the date at the beginning or the end of the new period. By default, the "end" of the period. Use "start" to change to the start of the period.
`clean`	Whether or not to round the collapsed index up / down to the next period boundary. The decision to round up / down is controlled by the side argument.
`message`	A boolean. If `message = TRUE`, the frequency used is output along with the units in the scale of the data.

Details

Uses a time-based period to apply functions to. This is useful in circumstances where you want to compare the observation values to aggregated values such as `mean()` or `median()` during a set time-based period. The returned output extends the length of the data frame so the differences can easily be computed.

Value

Returns a `tibbletime` object of class `tbl_time`.

Examples

```
library(dplyr)

data(tidyverse_cran_downloads)

# Basic Usage
tidyverse_cran_downloads %>%
    time_apply(count, period = "1 week", .fun = mean, na.rm = TRUE)
```

`time_decompose` *Decompose a time series in preparation for anomaly detection*

Description

Decompose a time series in preparation for anomaly detection

Usage

```
time_decompose(data, target, method = c("stl", "twitter"),
  frequency = "auto", trend = "auto", ..., merge = FALSE,
  message = TRUE)
```

Arguments

`data`	A `tibble` or `tbl_time` object.
`target`	A column to apply the function to
`method`	The time series decomposition method. One of `"stl"` or `"twitter"`. The STL method uses seasonal decomposition (see `decompose_stl()`). The Twitter method uses `trend` to remove the trend (see `decompose_twitter()`).
`frequency`	Controls the seasonal adjustment (removal of seasonality). Input can be either "auto", a time-based definition (e.g. "2 weeks"), or a numeric number of observations per frequency (e.g. 10). Refer to `time_frequency()`.
`trend`	Controls the trend component For stl, the trend controls the sensitivity of the lowess smoother, which is used to remove the remainder. For twitter, the trend controls the period width of the median, which are used to remove the trend and center the remainder.
`...`	Additional parameters passed to the underlying method functions.
`merge`	A boolean. `FALSE` by default. If `TRUE`, will append results to the original data.
`message`	A boolean. If `TRUE`, will output information related to `tbl_time` conversions, frequencies, and trend / median spans (if applicable).

Details

The `time_decompose()` function generates a time series decomposition on `tbl_time` objects. The function is "tidy" in the sense that it works on data frames. It is designed to work with time-based data, and as such must have a column that contains date or datetime information. The function also works with grouped data. The function implements several methods of time series decomposition, each with benefits.

STL:

The STL method (`method = "stl"`) implements time series decomposition using the underlying `decompose_stl()` function. If you are familiar with `stats::stl()`, the function is a "tidy" version that is designed to work with `tbl_time` objects. The decomposition separates the "season" and "trend" components from the "observed" values leaving the "remainder" for anomaly detection. The user can control two parameters: `frequency` and `trend`. The `frequency` parameter adjusts the "season" component that is removed from the "observed" values. The `trend` parameter adjusts the trend window (`t.window` parameter from `stl()`) that is used. The user may supply both `frequency` and `trend` as time-based durations (e.g. "6 weeks") or numeric values (e.g. 180) or "auto", which predetermines the frequency and/or trend based on the scale of the time series.

Twitter:

The Twitter method (`method = "twitter"`) implements time series decomposition using the methodology from the Twitter AnomalyDetection package. The decomposition separates the "seasonal" component and then removes the median data, which is a different approach than the STL method for removing the trend. This approach works very well for low-growth + high seasonality data. STL may be a better approach when trend is a large factor. The user can control two parameters: `frequency` and `trend`. The `frequency` parameter adjusts the "season" component that is removed from the "observed" values. The `trend` parameter adjusts the period width of the median spans that are used. The user may supply both `frequency` and `trend` as time-based durations (e.g. "6 weeks") or numeric values (e.g. 180) or "auto", which predetermines the frequency and/or median spans based on the scale of the time series.

Value

Returns a `tbl_time` object.

References

1. CLEVELAND, R. B., CLEVELAND, W. S., MCRAE, J. E., AND TERPENNING, I. STL: A Seasonal-Trend Decomposition Procedure Based on Loess. Journal of Official Statistics, Vol. 6, No. 1 (1990), pp. 3-73.
2. Owen S. Vallis, Jordan Hochenbaum and Arun Kejariwal (2014).A Novel Technique for Long-Term Anomaly Detection in the Cloud. Twitter Inc.
3. Owen S. Vallis, Jordan Hochenbaum and Arun Kejariwal (2014). AnomalyDetection: Anomaly Detection UsingSeasonal Hybrid Extreme Studentized Deviate Test. R package version 1.0.

See Also

Decomposition Methods (Powers `time_decompose`)

- `decompose_stl()`
- `decompose_twitter()`

Time Series Anomaly Detection Functions (anomaly detection workflow):

- anomalize()
- time_recompose()

Examples

```
library(dplyr)

data(tidyverse_cran_downloads)

# Basic Usage
tidyverse_cran_downloads %>%
    time_decompose(count, method = "stl")

# twitter
tidyverse_cran_downloads %>%
    time_decompose(count,
                   method       = "twitter",
                   frequency    = "1 week",
                   trend        = "2 months",
                   merge        = TRUE,
                   message      = FALSE)
```

`time_frequency` *Generate a time series frequency from a periodicity*

Description

Generate a time series frequency from a periodicity

Usage

```
time_frequency(data, period = "auto", message = TRUE)

time_trend(data, period = "auto", message = TRUE)
```

Arguments

`data`	A `tibble` with a date or datetime index.
`period`	Either "auto", a time-based definition (e.g. "2 weeks"), or a numeric number of observations per frequency (e.g. 10). See `tibbletime::collapse_by()` for period notation.
`message`	A boolean. If `message = TRUE`, the frequency used is output along with the units in the scale of the data.

Details

A frequency is loosely defined as the number of observations that comprise a cycle in a data set. The trend is loosely defined as time span that can be aggregated across to visualize the central tendency of the data. It's often easiest to think of frequency and trend in terms of the time-based units that the data is already in. **This is what `time_frequency()` and `time_trend()` enable: using time-based periods to define the frequency or trend.**

Frequency:

As an example, a weekly cycle is often 5-days (for working days) or 7-days (for calendar days). Rather than specify a frequency of 5 or 7, the user can specify `period = "1 week"`, and time_frequency()‘ will detect the scale of the time series and return 5 or 7 based on the actual data.

The `period` argument has three basic options for returning a frequency. Options include:

- `"auto"`: A target frequency is determined using a pre-defined template (see `template` below).
- `time-based duration`: (e.g. "1 week" or "2 quarters" per cycle)
- `numeric number of observations`: (e.g. 5 for 5 observations per cycle)

The `template` argument is only used when `period = "auto"`. The template is a tibble of three features: `time_scale`, `frequency`, and `trend`. The algorithm will inspect the scale of the time series and select the best frequency that matches the scale and number of observations per target frequency. A frequency is then chosen on be the best match. The predefined template is stored in a function `time_scale_template()`. However, the user can come up with his or her own template changing the values for frequency in the data frame and saving it to `anomalize_options$time_scale_template`.

Trend:

As an example, the trend of daily data is often best aggregated by evaluating the moving average over a quarter or a month span. Rather than specify the number of days in a quarter or month, the user can specify "1 quarter" or "1 month", and the `time_trend()` function will return the correct number of observations per trend cycle. In addition, there is an option, `period = "auto"`, to auto-detect an appropriate trend span depending on the data. The `template` is used to define the appropriate trend span.

Value

Returns a scalar numeric value indicating the number of observations in the frequency or trend span.

Examples

```
library(dplyr)

data(tidyverse_cran_downloads)

#### FREQUENCY DETECTION ####

# period = "auto"
tidyverse_cran_downloads %>%
    filter(package == "tidyquant") %>%
    ungroup() %>%
    time_frequency(period = "auto")
```

```
time_scale_template()

# period = "1 month"
tidyverse_cran_downloads %>%
    filter(package == "tidyquant") %>%
    ungroup() %>%
    time_frequency(period = "1 month")

#### TREND DETECTION ####

tidyverse_cran_downloads %>%
    filter(package == "tidyquant") %>%
    ungroup() %>%
    time_trend(period = "auto")
```

`time_recompose` *Recompose bands separating anomalies from "normal" observations*

Description

Recompose bands separating anomalies from "normal" observations

Usage

```
time_recompose(data)
```

Arguments

`data`	A `tibble` or `tbl_time` object that has been processed with `time_decompose()` and `anomalize()`.

Details

The `time_recompose()` function is used to generate bands around the "normal" levels of observed values. The function uses the remainder_l1 and remainder_l2 levels produced during the `anomalize()` step and the season and trend/median_spans values from the `time_decompose()` step to reconstruct bands around the normal values.

The following key names are required: observed:remainder from the `time_decompose()` step and remainder_l1 and remainder_l2 from the `anomalize()` step.

Value

Returns a `tbl_time` object.

See Also

Time Series Anomaly Detection Functions (anomaly detection workflow):

- `time_decompose()`
- `anomalize()`

Examples

```
library(dplyr)

data(tidyverse_cran_downloads)

# Basic Usage
tidyverse_cran_downloads %>%
    time_decompose(count, method = "stl") %>%
    anomalize(remainder, method = "iqr") %>%
    time_recompose()
```

Index