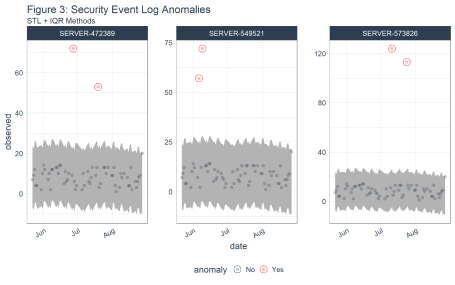
[Information Security (InfoSec)](https://en.wikipedia.org/wiki/Information_security) is critical to a business. For those new to **InfoSec, it is the state of being protected against the unauthorized use of information, especially electronic data. A single malicious threat can cause massive damage to a firm, large or small.** **The data I routinely deals with is massive in scale**: I processes [security event telemetry](https://en.wikipedia.org/wiki/Telemetry) of all types (operating systems, network, applications, service layer) for all of Windows, Xbox, the Universal Store (transactions/purchases), and a few others. Billions of events in short order.

**Learning Trajectory**

This is a great article from **master in information security, Russ McCree**. You’ll learn how Russ is using our new package for time series anomaly detection, anomalize, within his Blue Team (internal thread-defending team) work at Microsoft. He provides real-world examples of “threat hunting”, or the act of identifying malicious attacks on servers and how anomalize can help to *algorithmically* identify threats. Specifically, Russ shows you how to detect anomalies in security event logs as shown below.



**Threat Hunting With Anomalize**

By Russ McCree, Group Program Manager of Microsoft’s Windows and Devices Group

When, in redefined *DFIR* under the premise of **Deeper Functionality for Investigators in R**, I discovered a “tip of the iceberg” scenario. To that end, I’d like to revisit the DFIR concept with an additional discovery and opportunity. In reality, this is really a case of Deeper Functionality for Investigators in R within the original and paramount activity of [**D**igital **F**orensics/**I**ncident **R**esponse (DFIR)](https://en.wikipedia.org/wiki/Digital_forensics).

**Massive Data Requires Algorithmic Methods**

Those of us in the DFIR practice, and Blue Teaming at large, are overwhelmed by data and scale. **Success truly requires algorithmic methods**. If you’re not already invested here I have an **immediately applicable case study for you in tidy anomaly detection with anomalize**.

First, let me give credit where entirely due for the work that follows. Everything I discuss and provide is immediately derivative. When a client asked *Business Science* to build an open source anomaly detection algorithm that suited their needs:

“*a tidy anomaly detection algorithm that’s time-based (built on top of tibbletime) and scalable from one to many time series*,”

I’d say he responded beautifully.

**Anomalizing in InfoSec: Threat Hunting At Scale**

I’ll quote specifically before shifting context:

“*Our client had a challenging problem: detecting anomalies in time series on daily or weekly data at scale. Anomalies indicate exceptional events, which could be increased web traffic in the marketing domain or a malfunctioning server in the IT domain. Regardless, it’s important to flag these unusual occurrences to ensure the business is running smoothly. One of the challenges was that the client deals with not one time series but thousands that need to be analyzed for these extreme events.*”

**Key takeaway: Detecting anomalies in time series on daily or weekly data at scale. Anomalies indicate exceptional events.**

Now shift context with me to security-specific events and incidents, as they pertain to security monitoring, incident response, and threat hunting. I discussed **Time Series Regression (TSR) with the Holt-Winters method and a focus on seasonality and trends**. Unfortunately, I couldn’t share the code for how we applied TSR, but pointed out alternate methods, including **Seasonal and Trend Decomposition using Loess (STL)**, which:

* Handles any type of seasonality (which can change over time)
* Automatically smooths the trend-cycle (this can also be controlled by the user)
* Is robust to outliers (high leverage points that can shift the mean)

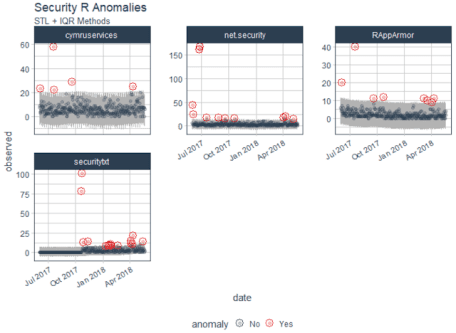
The anomalize package includes the following main functions:

* time\_decompose(): Separates the time series into seasonal, trend, and remainder components. The methods used including **STL** and **Twitter**.
* anomalize(): Applies anomaly detection methods to the remainder component. The methods used including **IQR** and **GESD** are described.
* time\_recompose(): Calculates limits that separate the “normal” data from the anomalies.

I initially toyed with tweaking Matt’s demo to model downloads for security-specific R packages (yes, there are such things) from CRAN, including:

* RAppArmor
* net.security
* securitytxt
* cymruservices

Alas, this was a mere rip and replace, and really didn’t exhibit the use of anomalize in a deserving, varied, truly security-specific context. That said, I was able to generate immediate results doing so, as seen in **Figure 1: Security R Anomalies**.



**Code Tutorial: Anomalizing A Real Security Scenario**

I wanted to run anomalize against **a real security data scenario**, so I went back to the dataset from the original DFIR articles where I’d utilized counts of 4624 Event IDs per day, per user, on a given set of servers. As utilized originally, I’d represented results specific to only one device and user, but herein is the beauty of anomalize. We can achieve quick results across multiple times series (multiple systems/users). This premise is but one of many where time series analysis and seasonality can be applied to security data.

For those that would like to follow along, load the following libraries.

library(tidyverse)

library(tibbletime)

library(anomalize)

I originally tried to write log data from log.csv straight to an anomalize.R script with logs = read\_csv("log.csv") into a tibble (ready your troubles with tibbles jokes), which was not being parsed accurately, particularly time attributes. To correct this, from Matt’s GitHub I grabbed tidyverse\_cran\_downloads.R, and modified it as follows:

# Path to security log data

logs\_path <- "https://raw.githubusercontent.com/holisticinfosec/toolsmith\_R/master/anomalize/log.csv"

# Group by server, Convert to tibbletime

security\_access\_logs <- read\_csv(logs\_path) %>%

group\_by(server) %>%

as\_tbl\_time(date)

security\_access\_logs

## # A time tibble: 198 x 3

## # Index: date

## # Groups: server [3]

## date count server

##

## 1 2017-05-22 7 SERVER-549521

## 2 2017-05-23 9 SERVER-549521

## 3 2017-05-24 12 SERVER-549521

## 4 2017-05-25 4 SERVER-549521

## 5 2017-05-26 4 SERVER-549521

## 6 2017-05-30 2 SERVER-549521

## 7 2017-05-31 10 SERVER-549521

## 8 2017-06-01 14 SERVER-549521

## 9 2017-06-02 12 SERVER-549521

## 10 2017-06-05 7 SERVER-549521

## # ... with 188 more rows

This helped greatly thanks to the tibbletime package, which **“is an extension that allows for the creation of time aware tibbles”**. Some immediate advantages of this include: the ability to perform time-based subsetting on tibbles, quickly summarising and aggregating results by time periods. Guess what, Matt’s colleague, Davis Vaughan, is the one who wrote tibbletime too. 

I then followed Matt’s sequence as he posted on *Business Science*, but with my logs defined as a function in Security\_Access\_Logs\_Function.R. Following, I’ll give you the code snippets, as revised from Matt’s examples, followed by their respective results specific to processing my Event ID 4624 daily count log.

First, let’s summarize daily logon counts across three servers over four months.  
The result is evident in **Figure 2: Server Logon Counts**.

# plot login counts across 3 servers

security\_access\_logs %>%

ggplot(aes(date, count)) +

geom\_point(color = "#2c3e50", alpha = 0.5) +

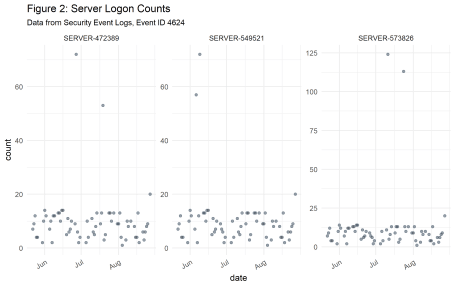
facet\_wrap(~ server, scale = "free\_y", ncol = 3) +

theme\_minimal() +

theme(axis.text.x = element\_text(angle = 30, hjust = 1)) +

labs(title = "Figure 2: Server Logon Counts",

subtitle = "Data from Security Event Logs, Event ID 4624")



Next, let’s determine which daily download logons are anomalous with Matt’s three main functions, time\_decompose(), anomalize(), and time\_recompose(), along with the visualization function, plot\_anomalies(), across the same three servers over four months. The result is revealed in **Figure 3: Security Event Log Anomalies**.

# Detect and plot security event log anomalies

security\_access\_logs %>%

# Data Manipulation / Anomaly Detection

time\_decompose(count, method = "stl") %>%

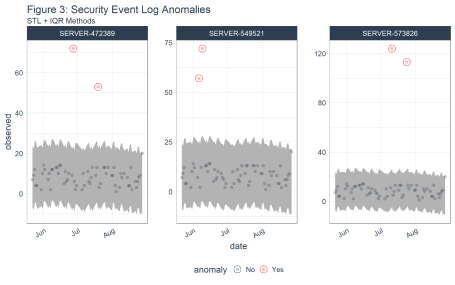
anomalize(remainder, method = "iqr") %>%

time\_recompose() %>%

# Anomaly Visualization

plot\_anomalies(time\_recomposed = TRUE, ncol = 3, alpha\_dots = 0.25) +

labs(title = "Figure 3: Security Event Log Anomalies", subtitle = "STL + IQR Methods")



Next, we can compare method combinations for the time\_decompose() and anomalize() methods:

* The Twitter (time\_decompose()) combined with the GESD (anomalize()) method
* The STL (time\_decompose()) and IQR (anomalize()) arguments

These are two different decomposition and anomaly detection approaches.

**Twitter + GESD**:

Following Matt’s method using Twitter’s AnomalyDetection package, combining time\_decompose(method = "twitter") with anomalize(method = "gesd"), while adjusting the trend = "4 months" to adjust median spans, we’ll focus only on SERVER-549521. In **Figure 4: SERVER-549521 Anomalies, Twitter + GESD**, you’ll note that there are anomalous logon counts on SERVER-549521 in June.

# Get only SERVER549521 access

SERVER549521 <- security\_access\_logs %>%

filter(server == "SERVER-549521") %>%

ungroup()

# Anomalize using Twitter + GESD

SERVER549521 %>%

# Twitter + GESD

time\_decompose(count, method = "twitter", trend = "4 months") %>%

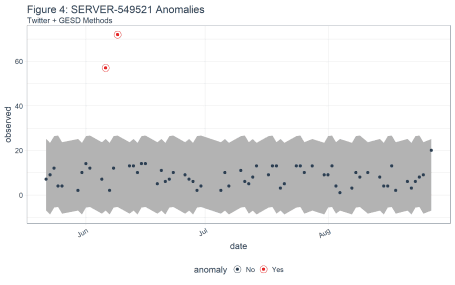
anomalize(remainder, method = "gesd") %>%

time\_recompose() %>%

# Anomaly Visualziation

plot\_anomalies(time\_recomposed = TRUE) +

labs(title = "Figure 4: SERVER-549521 Anomalies", subtitle = "Twitter + GESD Methods")



**STL + IQR**:

Again, we note anomalies in June, as seen in **Figure 5: STL + IQR Methods**. Obviously, the results are quite similar, as one would hope.

# STL + IQR

SERVER549521 %>%

# STL + IQR Anomaly Detection

time\_decompose(count, method = "stl", trend = "4 months") %>%

anomalize(remainder, method = "iqr") %>%

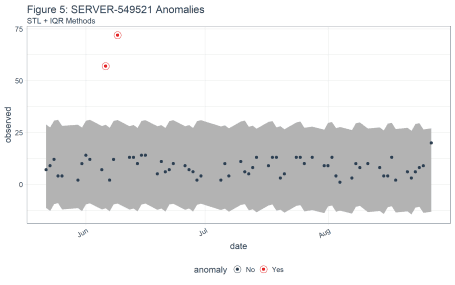
time\_recompose() %>%

# Anomaly Visualization

plot\_anomalies(time\_recomposed = TRUE) +

labs(title = "Figure 5: SERVER-549521 Anomalies",

subtitle = "STL + IQR Methods")



Finally, let use Matt’s plot\_anomaly\_decomposition() for visualizing the inner workings of how algorithm detects anomalies in the remainder for SERVER-549521. The result is a four part visualization, including observed, season, trend, and remainder as seen in **Figure 6**.

# Created from Anomalize project, Matt Dancho

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

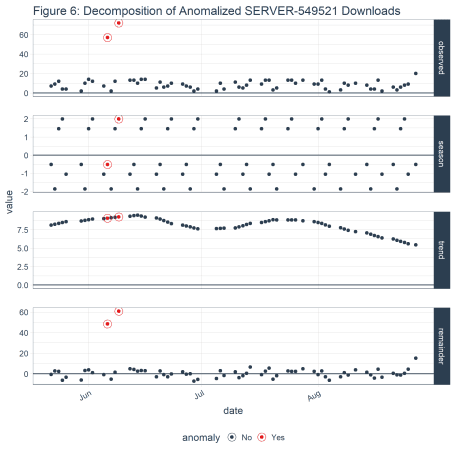
SERVER549521 %>%

time\_decompose(count) %>%

anomalize(remainder) %>%

plot\_anomaly\_decomposition() +

labs(title = "Figure 6: Decomposition of Anomalized SERVER-549521 Downloads")



**Future Work In InfoSec: Anomalize At A Larger Scale**

I’m really looking forward to putting these methods to use at a **much larger scale**, across a far broader event log dataset. I firmly assert that Blue Teams are already way behind in combating automated adversary tactics and problems of sheer scale, so…much…data. It’s only with tactics such as Matt’s anomalize package, and others of its ilk, that defenders can hope to succeed.

Code Chunks – R Scripts of Anomalize Package – Global Variables

|  |  |
| --- | --- |
| globalVariables(c( | |
|  | | "n", |
|  | | ".", |
|  | | ".period\_groups", |
|  | | "data", |
|  | | "abs\_diff\_lower", |
|  | | "abs\_diff\_upper", |
|  | | "below\_max\_anoms", |
|  | | "centerline", |
|  | | "critical\_value", |
|  | | "direction", |
|  | | "index", |
|  | | "limit\_lower", |
|  | | "limit\_upper", |
|  | | "max\_abs\_diff", |
|  | | "outlier", |
|  | | "outlier\_reported", |
|  | | "sorting", |
|  | | "test\_statistic", |
|  | | "value", |
|  | | "observed", |
|  | | "random", |
|  | | "remainder", |
|  | | "seasadj", |
|  | | "season", |
|  | | "trend", |
|  | | "target", |
|  | | "anomaly", |
|  | | "key", |
|  | | "median\_spans", |
|  | | "recomposed\_l1", |
|  | | "recomposed\_l2", |
|  | | "data\_names", |
|  | | "nested.col" |
|  | | )) |
|  | |  |
| Anomalize.R - #' Detect anomalies using the tidyverse  anomalize <- function(data, target, method = c("iqr", "gesd"), |
|  | alpha = 0.05, max\_anoms = 0.20, verbose = FALSE) { | |
|  | UseMethod("anomalize", data) | |
|  | } | |
|  |  | |
|  | #' @export | |
|  | anomalize.default <- function(data, target, method = c("iqr", "gesd"), | |
|  | alpha = 0.05, max\_anoms = 0.20, verbose = FALSE) { | |
|  | stop("Error anomalize(): Object is not of class `tbl\_df` or `tbl\_time`.", call. = FALSE) | |
|  | } | |
|  |  | |
|  | #' @export | |
|  | anomalize.tbl\_df <- function(data, target, method = c("iqr", "gesd"), | |
|  | alpha = 0.05, max\_anoms = 0.20, verbose = FALSE) { | |
|  |  | |
|  | # Checks | |
|  | if (missing(target)) stop('Error in anomalize(): argument "target" is missing, with no default', call. = FALSE) | |
|  |  | |
|  | # Setup | |
|  | target\_expr <- rlang::enquo(target) | |
|  |  | |
|  | method <- tolower(method[[1]]) | |
|  | x <- data %>% dplyr::pull(!! target\_expr) | |
|  |  | |
|  | # Detect Anomalies | |
|  | # method <- tolower(method[[1]]) | |
|  | # args <- list(x = data %>% dplyr::pull(!! target\_expr), | |
|  | # alpha = alpha, | |
|  | # max\_anoms = max\_anoms, | |
|  | # verbose = TRUE) | |
|  | # | |
|  | # outlier\_list <- do.call(method, args) | |
|  |  | |
|  | # Explicitly call functions | |
|  | if (method == "iqr") { | |
|  | outlier\_list <- anomalize::iqr(x = x, | |
|  | alpha = alpha, | |
|  | max\_anoms = max\_anoms, | |
|  | verbose = TRUE) | |
|  | } else if (method == "gesd") { | |
|  | outlier\_list <- anomalize::gesd(x = x, | |
|  | alpha = alpha, | |
|  | max\_anoms = max\_anoms, | |
|  | verbose = TRUE) | |
|  |  | |
|  | } else { | |
|  | stop("The `method` selected is invalid.", call. = FALSE) | |
|  | } | |
|  |  | |
|  | outlier <- outlier\_list$outlier | |
|  | limit\_lower <- outlier\_list$critical\_limits[[1]] | |
|  | limit\_upper <- outlier\_list$critical\_limits[[2]] | |
|  |  | |
|  | # Returns | |
|  | ret <- data %>% | |
|  | dplyr::mutate(!! paste0(dplyr::quo\_name(target\_expr), "\_l1") := limit\_lower, | |
|  | !! paste0(dplyr::quo\_name(target\_expr), "\_l2") := limit\_upper) %>% | |
|  | tibble::add\_column(anomaly = outlier) | |
|  |  | |
|  | if (verbose) { | |
|  | ret <- list( | |
|  | anomalized\_tbl = ret, | |
|  | anomaly\_details = outlier\_list | |
|  | ) | |
|  |  | |
|  | return(ret) | |
|  |  | |
|  | } else { | |
|  | return(ret) | |
|  | } | |
|  |  | |
|  | } | |
|  |  | |
|  | #' @export | |
|  | anomalize.grouped\_df <- function(data, target, method = c("iqr", "gesd"), | |
|  | alpha = 0.05, max\_anoms = 0.20, verbose = FALSE, ...) { | |
|  |  | |
|  | # Checks | |
|  | if (missing(target)) stop('Error in anomalize(): argument "target" is missing, with no default', call. = FALSE) | |
|  | if (verbose) warning(glue::glue("Cannot use 'verbose = TRUE' with grouped data.")) | |
|  |  | |
|  | # Setup | |
|  | target\_expr <- dplyr::enquo(target) | |
|  |  | |
|  | ret <- data %>% | |
|  | grouped\_mapper( | |
|  | .f = anomalize, | |
|  | target = !! target\_expr, | |
|  | method = method[[1]], | |
|  | alpha = alpha, | |
|  | max\_anoms = max\_anoms, | |
|  | verbose = F, | |
|  | ...) | |
|  |  | |
|  | return(ret) | |
|  |  | |
|  | } | |

Anomalize\_Clean.R - Clean anomalies from anomalized data

|  |
| --- |
| clean\_anomalies <- function(data) { |
|  | UseMethod("clean\_anomalies", data) |
|  | } |
|  |  |
|  | #' @export |
|  | clean\_anomalies.default <- function(data) { |
|  | stop("Error clean\_anomalies(): Object is not of class `tbl\_df` or `tbl\_time`.", call. = FALSE) |
|  | } |
|  |  |
|  | #' @export |
|  | clean\_anomalies.tbl\_df <- function(data) { |
|  |  |
|  | # Checks |
|  | check\_clean\_anomalies\_input(data) |
|  |  |
|  | # Get method col |
|  | method\_col <- get\_method\_col(data) |
|  |  |
|  | if (method\_col == "trend") { |
|  | data %>% |
|  | dplyr::mutate(observed\_cleaned = ifelse(anomaly == "Yes", season + trend, observed)) |
|  | } else { |
|  | data %>% |
|  | dplyr::mutate(observed\_cleaned = ifelse(anomaly == "Yes", season + median\_spans, observed)) |
|  | } |
|  |  |
|  | } |
|  |  |
|  | check\_clean\_anomalies\_input <- function(data) { |
|  |  |
|  | data\_names <- names(data) |
|  |  |
|  | # Detect method - STL or Twitter |
|  | method\_names <- c("trend", "median\_spans") |
|  | method\_name\_in\_data <- any(method\_names %in% data\_names) |
|  |  |
|  | # Check - No method name in data |
|  | if (!method\_name\_in\_data) stop("Error clean\_anomalies(): Output does not contain a column named trend or median\_spans. This may occur if the output was not detrended with time\_decompose().", call. = FALSE) |
|  |  |
|  | # Check - Required names from time\_decompose() |
|  | required\_names <- c("observed", "season") |
|  | required\_names\_in\_data <- all(required\_names %in% data\_names) |
|  | if (!required\_names\_in\_data) stop("Error clean\_anomalies(): Output does not contain columns named observed and season. This may occur if the output was not detrended with time\_decompose().", call. = FALSE) |
|  |  |
|  | # Check - Required names from time\_decompose() |
|  | required\_names <- c("anomaly") |
|  | required\_names\_in\_data <- all(required\_names %in% data\_names) |
|  | if (!required\_names\_in\_data) stop("Error clean\_anomalies(): Output does not contain columns named anomaly. This may occur if the output was not anomalized with anomalize().", call. = FALSE) |
|  |  |
|  |  |
|  | } |
|  |  |
|  |  |
|  | get\_method\_col <- function(data) { |
|  |  |
|  | data\_names <- names(data) |
|  |  |
|  | # Detect method - STL or Twitter |
|  | method\_names <- c("trend", "median\_spans") |
|  | method\_name\_in\_data <- method\_names %in% data\_names |
|  |  |
|  | method\_names[method\_name\_in\_data] |
|  |  |
|  | } |

Anomalize\_Methods.R - Methods that power anomalize()

|  |
| --- |
| iqr <- function(x, alpha = 0.05, max\_anoms = 0.2, verbose = FALSE) { |
|  | quantile\_x <- stats::quantile(x, prob = c(0.25, 0.75), na.rm = TRUE) |
|  | iq\_range <- quantile\_x[[2]] - quantile\_x[[1]] |
|  | limits <- quantile\_x + (0.15 / alpha) \* iq\_range \* c(-1, 1) |
|  |  |
|  | outlier\_idx <- ((x < limits[1]) | (x > limits[2])) |
|  | outlier\_vals <- x[outlier\_idx] |
|  | outlier\_response <- ifelse(outlier\_idx == TRUE, "Yes", "No") |
|  |  |
|  | vals\_tbl <- tibble::tibble(value = x) %>% |
|  | tibble::rownames\_to\_column(var = "index") %>% |
|  | # Establish limits and assess if outside of limits |
|  | dplyr::mutate( |
|  | limit\_lower = limits[1], |
|  | limit\_upper = limits[2], |
|  | abs\_diff\_lower = ifelse(value <= limit\_lower, abs(value - limit\_lower), 0), |
|  | abs\_diff\_upper = ifelse(value >= limit\_upper, abs(value - limit\_upper), 0), |
|  | max\_abs\_diff = ifelse(abs\_diff\_lower > abs\_diff\_upper, abs\_diff\_lower, abs\_diff\_upper) |
|  | ) %>% |
|  | dplyr::select(index, dplyr::everything()) %>% |
|  | dplyr::select(-c(abs\_diff\_lower, abs\_diff\_upper)) %>% |
|  | # Sort by absolute distance from centerline of limits |
|  | dplyr::mutate( |
|  | centerline = (limit\_upper + limit\_lower) / 2, |
|  | sorting = abs(value - centerline) |
|  | ) %>% |
|  | dplyr::arrange(dplyr::desc(sorting)) %>% |
|  | dplyr::select(-c(centerline, sorting)) %>% |
|  | tibble::rownames\_to\_column(var = "rank") %>% |
|  | dplyr::mutate( |
|  | rank = as.numeric(rank), |
|  | index = as.numeric(index) |
|  | ) %>% |
|  | # Identify outliers |
|  | dplyr::arrange(dplyr::desc(max\_abs\_diff)) %>% |
|  | dplyr::mutate( |
|  | outlier = ifelse(max\_abs\_diff > 0, "Yes", "No"), |
|  | below\_max\_anoms = ifelse(dplyr::row\_number() / dplyr::n() > max\_anoms, |
|  | "No", "Yes" |
|  | ), |
|  | outlier\_reported = ifelse(outlier == "Yes" & below\_max\_anoms == "Yes", |
|  | "Yes", "No" |
|  | ), |
|  | direction = dplyr::case\_when( |
|  | (outlier\_reported == "Yes") & (value > limit\_upper) ~ "Up", |
|  | (outlier\_reported == "Yes") & (value < limit\_lower) ~ "Down", |
|  | TRUE ~ "NA" |
|  | ), |
|  | direction = ifelse(direction == "NA", NA, direction) |
|  | ) |
|  |  |
|  | vals\_tbl\_filtered <- vals\_tbl %>% |
|  | dplyr::filter(below\_max\_anoms == "Yes") %>% |
|  | dplyr::select(-c(max\_abs\_diff:below\_max\_anoms)) %>% |
|  | dplyr::rename(outlier = outlier\_reported) |
|  |  |
|  | # Critical Limits |
|  | if (any(vals\_tbl$outlier == "No")) { |
|  | # Non outliers identified, pick first limit |
|  | limit\_tbl <- vals\_tbl %>% |
|  | dplyr::filter(outlier == "No") %>% |
|  | dplyr::slice(1) |
|  | limits\_vec <- c( |
|  | limit\_lower = limit\_tbl$limit\_lower, |
|  | limit\_upper = limit\_tbl$limit\_upper |
|  | ) |
|  | } else { |
|  | # All outliers, pick last limits |
|  | limit\_tbl <- vals\_tbl %>% |
|  | dplyr::slice(n()) |
|  | limits\_vec <- c( |
|  | limit\_lower = limit\_tbl$limit\_lower, |
|  | limit\_upper = limit\_tbl$limit\_upper |
|  | ) |
|  | } |
|  |  |
|  | # Return results |
|  | if (verbose) { |
|  | outlier\_list <- list( |
|  | outlier = vals\_tbl %>% dplyr::arrange(index) %>% dplyr::pull(outlier\_reported), |
|  | outlier\_idx = vals\_tbl %>% dplyr::filter(outlier\_reported == "Yes") %>% dplyr::pull(index), |
|  | outlier\_vals = vals\_tbl %>% dplyr::filter(outlier\_reported == "Yes") %>% dplyr::pull(value), |
|  | outlier\_direction = vals\_tbl %>% dplyr::filter(outlier\_reported == "Yes") %>% dplyr::pull(direction), |
|  | critical\_limits = limits\_vec, |
|  | outlier\_report = vals\_tbl\_filtered |
|  | ) |
|  | return(outlier\_list) |
|  | } else { |
|  | return(vals\_tbl %>% dplyr::arrange(index) %>% dplyr::pull(outlier\_reported)) |
|  | } |
|  | } |
|  |  |
|  |  |
|  |  |
|  | # 1B. GESD: Generalized Extreme Studentized Deviate Test ---- |
|  |  |
|  | #' @export |
|  | #' @rdname anomalize\_methods |
|  | gesd <- function(x, alpha = 0.05, max\_anoms = 0.2, verbose = FALSE) { |
|  |  |
|  | # Variables |
|  | n <- length(x) |
|  | r <- trunc(n \* max\_anoms) # use max anoms to limit loop |
|  | R <- numeric(length = r) # test statistics for 'r' outliers |
|  |  |
|  | lambda <- numeric(length = r) # critical values for 'r' outliers |
|  | outlier\_ind <- numeric(length = r) # removed outlier observation values |
|  | outlier\_val <- numeric(length = r) # removed outlier observation values |
|  | m <- 0 # number of outliers |
|  | x\_new <- x # temporary observation values |
|  | median\_new <- numeric(length = r) |
|  | mad\_new <- numeric(length = r) |
|  |  |
|  | # Outlier detection |
|  | for (i in seq\_len(r)) { |
|  |  |
|  | # Compute test statistic |
|  | median\_new[i] <- median(x\_new) |
|  | mad\_new[i] <- mad(x\_new) |
|  |  |
|  | z <- abs(x\_new - median(x\_new)) / (mad(x\_new) + .Machine$double.eps) # Z-scores |
|  |  |
|  | max\_ind <- which(z == max(z), arr.ind = T)[1] # in case of ties, return first one |
|  | R[i] <- z[max\_ind] # max Z-score |
|  | outlier\_val[i] <- x\_new[max\_ind] # removed outlier observation values |
|  | outlier\_ind[i] <- which(x\_new[max\_ind] == x, arr.ind = T)[1] # index of removed outlier observation values |
|  | x\_new <- x\_new[-max\_ind] # remove observation that maximizes |x\_i - x\_mean| |
|  |  |
|  | # Compute critical values |
|  | p <- 1 - alpha / (2 \* (n - i + 1)) # probability |
|  | t\_pv <- qt(p, df = (n - i - 1)) # Critical value from Student's t distribution |
|  | lambda[i] <- ((n - i) \* t\_pv) / (sqrt((n - i - 1 + t\_pv^2) \* (n - i + 1))) |
|  |  |
|  | # Find exact number of outliers |
|  | # largest 'i' such that R\_i > lambda\_i |
|  | if (!is.na(R[i]) & !is.na(lambda[i])) { # qt can produce NaNs |
|  | if (R[i] > lambda[i]) { |
|  | m <- i |
|  | } |
|  | } |
|  | } |
|  |  |
|  | vals\_tbl <- tibble::tibble( |
|  | rank = as.numeric(1:r), |
|  | index = outlier\_ind, |
|  | value = outlier\_val, |
|  | test\_statistic = R, |
|  | critical\_value = lambda, |
|  | median = median\_new, |
|  | mad = mad\_new, |
|  | limit\_lower = median - critical\_value \* mad, |
|  | limit\_upper = critical\_value \* mad + median |
|  | ) %>% |
|  | dplyr::mutate( |
|  | outlier = ifelse(test\_statistic > critical\_value, "Yes", "No"), |
|  | direction = dplyr::case\_when( |
|  | (outlier == "Yes") & (value > limit\_upper) ~ "Up", |
|  | (outlier == "Yes") & (value < limit\_lower) ~ "Down", |
|  | TRUE ~ "NA" |
|  | ), |
|  | direction = ifelse(direction == "NA", NA, direction) |
|  | ) %>% |
|  | dplyr::select(-c(test\_statistic:mad)) |
|  |  |
|  | outlier\_index <- vals\_tbl %>% dplyr::filter(outlier == "Yes") %>% dplyr::pull(index) |
|  | outlier\_idx <- seq\_along(x) %in% outlier\_index |
|  | outlier\_response <- ifelse(outlier\_idx == TRUE, "Yes", "No") |
|  |  |
|  | # Critical Limits |
|  | if (any(vals\_tbl$outlier == "No")) { |
|  | # Non outliers identified, pick first limit |
|  | limit\_tbl <- vals\_tbl %>% |
|  | dplyr::filter(outlier == "No") %>% |
|  | dplyr::slice(1) |
|  | limits\_vec <- c( |
|  | limit\_lower = limit\_tbl$limit\_lower, |
|  | limit\_upper = limit\_tbl$limit\_upper |
|  | ) |
|  | } else { |
|  | # All outliers, pick last limits |
|  | limit\_tbl <- vals\_tbl %>% |
|  | dplyr::slice(n()) |
|  | limits\_vec <- c( |
|  | limit\_lower = limit\_tbl$limit\_lower, |
|  | limit\_upper = limit\_tbl$limit\_upper |
|  | ) |
|  | } |
|  |  |
|  | # Return results |
|  | if (verbose) { |
|  | outlier\_list <- list( |
|  | outlier = outlier\_response, |
|  | outlier\_idx = outlier\_index, |
|  | outlier\_vals = vals\_tbl %>% dplyr::filter(outlier == "Yes") %>% dplyr::pull(value), |
|  | outlier\_direction = vals\_tbl %>% dplyr::filter(outlier == "Yes") %>% dplyr::pull(direction), |
|  | critical\_limits = limits\_vec, |
|  | outlier\_report = vals\_tbl |
|  | ) |
|  | return(outlier\_list) |
|  | } else { |
|  | return(outlier\_response) |
|  | } |
|  | } |

Plot\_Anomalies.R - Visualize the anomalies in one or multiple time series

|  |
| --- |
| plot\_anomalies <- function(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) { |
|  |  |
|  | UseMethod("plot\_anomalies", data) |
|  | } |
|  |  |
|  | #' @export |
|  | plot\_anomalies.default <- function(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) { |
|  | stop("Object is not of class `tbl\_time`.", call. = FALSE) |
|  | } |
|  |  |
|  | #' @export |
|  | plot\_anomalies.tbl\_time <- function(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) { |
|  |  |
|  | # Checks |
|  | column\_names <- names(data) |
|  | check\_names <- c("observed", "anomaly") %in% column\_names |
|  | if (!all(check\_names)) stop('Error in plot\_anomalies(): key names are missing. Make sure observed:remainder, anomaly, recomposed\_l1, and recomposed\_l2 are present', call. = FALSE) |
|  |  |
|  | # Setup |
|  | date\_expr <- tibbletime::get\_index\_quo(data) |
|  | date\_col <- tibbletime::get\_index\_char(data) |
|  |  |
|  | g <- data %>% |
|  | ggplot2::ggplot(ggplot2::aes\_string(x = date\_col, y = "observed")) |
|  |  |
|  |  |
|  | if (time\_recomposed) { |
|  | check\_names <- c("recomposed\_l1", "recomposed\_l2") %in% column\_names |
|  | if (!all(check\_names)) stop('Error in plot\_anomalies(): key names are missing. Make sure recomposed\_l1 and recomposed\_l2 are present', call. = FALSE) |
|  |  |
|  | g <- g + |
|  | ggplot2::geom\_ribbon(ggplot2::aes(ymin = recomposed\_l1, ymax = recomposed\_l2), |
|  | fill = fill\_ribbon) |
|  |  |
|  | } |
|  |  |
|  | g <- g + |
|  | ggplot2::geom\_point(ggplot2::aes\_string(color = "anomaly"), size = size\_dots, alpha = alpha\_dots) + |
|  | ggplot2::geom\_point(ggplot2::aes\_string(x = date\_col, y = "observed", color = "anomaly"), |
|  | size = size\_circles, shape = 1, alpha = alpha\_circles, |
|  | data = data %>% dplyr::filter(anomaly == "Yes"), |
|  | inherit.aes = FALSE) + |
|  | theme\_tq() + |
|  | ggplot2::scale\_color\_manual(values = c("No" = color\_no, "Yes" = color\_yes)) + |
|  | ggplot2::theme(axis.text.x = ggplot2::element\_text(angle = 30, hjust = 1)) |
|  |  |
|  |  |
|  |  |
|  |  |
|  | if (dplyr::is.grouped\_df(data)) { |
|  |  |
|  | facet\_group <- dplyr::groups(data) %>% |
|  | purrr::map(quo\_name) %>% |
|  | unlist() %>% |
|  | paste0(collapse = " + ") |
|  |  |
|  | g <- g + |
|  | ggplot2::facet\_wrap(as.formula(paste0(" ~ ", facet\_group)), |
|  | scales = "free\_y", ncol = ncol) |
|  | } |
|  |  |
|  | return(g) |
|  |  |
|  | } |

Plot\_Anomaly\_Decomposition.R - Visualize the time series decomposition with anomalies shown

|  |
| --- |
| plot\_anomaly\_decomposition <- function(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") { |
|  | UseMethod("plot\_anomaly\_decomposition", data) |
|  |  |
|  | } |
|  |  |
|  | #' @export |
|  | plot\_anomaly\_decomposition.default <- function(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") { |
|  | stop("Object is not of class `tbl\_time`.", call. = FALSE) |
|  |  |
|  |  |
|  | } |
|  |  |
|  | #' @export |
|  | plot\_anomaly\_decomposition.grouped\_tbl\_time <- function(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") { |
|  | stop("Object cannot be grouped. Select a single time series for evaluation, and use `dplyr::ungroup()`.", call. = FALSE) |
|  |  |
|  |  |
|  | } |
|  |  |
|  | #' @export |
|  | plot\_anomaly\_decomposition.tbl\_time <- function(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") { |
|  |  |
|  | # Checks |
|  | column\_names <- names(data) |
|  | check\_names <- c("observed", "remainder", "anomaly", "remainder\_l1", "remainder\_l2") %in% column\_names |
|  | if (!all(check\_names)) stop('Error in plot\_anomaly\_decomposition(): key names are missing. Make sure observed:remainder, remainder\_l1, and remainder\_l2 are present', call. = FALSE) |
|  |  |
|  |  |
|  | # Setup |
|  | date\_expr <- tibbletime::get\_index\_quo(data) |
|  | date\_col <- tibbletime::get\_index\_char(data) |
|  |  |
|  | data\_anomaly\_tbl <- data %>% |
|  | dplyr::select(!! date\_expr, observed:remainder, anomaly) %>% |
|  | tidyr::gather(key = key, value = value, -dplyr::one\_of(c(!! date\_col, 'anomaly')), factor\_key = T) |
|  |  |
|  | g <- data\_anomaly\_tbl %>% |
|  | ggplot2::ggplot(ggplot2::aes\_string(x = date\_col, y = "value", color = "anomaly")) + |
|  | # Points |
|  | ggplot2::geom\_point(size = size\_dots, alpha = alpha\_dots) + |
|  | # Circles |
|  | ggplot2::geom\_point(size = size\_circles, shape = 1, alpha = alpha\_circles, |
|  | data = data\_anomaly\_tbl %>% dplyr::filter(anomaly == "Yes")) + |
|  | # Horizontal Line at Y = 0 |
|  | ggplot2::geom\_hline(yintercept = 0, color = palette\_light()[[1]]) + |
|  | theme\_tq() + |
|  | ggplot2::facet\_wrap(~ key, ncol = ncol, scales = "free\_y", strip.position = strip.position) + |
|  | ggplot2::scale\_color\_manual(values = c("No" = color\_no, "Yes" = color\_yes)) + |
|  | ggplot2::theme(axis.text.x = ggplot2::element\_text(angle = 30, hjust = 1)) |
|  |  |
|  |  |
|  | return(g) |
|  |  |
|  | } |

Prep\_tbl\_Time.R - Automatically create tibbletime objects from tibbles

|  |
| --- |
| prep\_tbl\_time <- function(data, message = FALSE) { |
|  | UseMethod("prep\_tbl\_time", data) |
|  | } |
|  |  |
|  | #' @export |
|  | prep\_tbl\_time.default <- function(data, message = FALSE) { |
|  | stop("Object is not of class `data.frame`.", call. = FALSE) |
|  | } |
|  |  |
|  |  |
|  | #' @export |
|  | prep\_tbl\_time.data.frame <- function(data, message = FALSE) { |
|  |  |
|  | cl <- class(data)[[1]] |
|  |  |
|  | idx <- tryCatch(timetk::tk\_get\_timeseries\_variables(data)[[1]], error = function(e) stop("Error in prep\_tbl\_time(): No date or datetime column found.")) |
|  |  |
|  | data <- data %>% |
|  | tibbletime::as\_tbl\_time(index = !! rlang::sym(idx)) |
|  |  |
|  | if (message) message(glue::glue("Converting from {cl} to {class(data)[[1]]}. |
|  | Auto-index message: index = {idx}")) |
|  |  |
|  | return(data) |
|  | } |
|  |  |
|  | #' @export |
|  | prep\_tbl\_time.tbl\_time <- function(data, message = FALSE) { |
|  | return(data) |
|  | } |

TidyQuant\_theme\_Compat.R- tidyquant functions copied to remove dependency on tidyquant

|  |
| --- |
| theme\_tq <- function(base\_size = 11, base\_family = "") { |
|  |  |
|  | # Tidyquant colors |
|  | blue <- "#2c3e50" |
|  | green <- "#18BC9C" |
|  | white <- "#FFFFFF" |
|  | grey <- "grey80" |
|  |  |
|  | # Starts with theme\_grey and then modify some parts |
|  | ggplot2::theme\_grey(base\_size = base\_size, base\_family = base\_family) %+replace% |
|  | ggplot2::theme( |
|  |  |
|  | # Base Inherited Elements |
|  | line = ggplot2::element\_line(colour = blue, size = 0.5, linetype = 1, |
|  | lineend = "butt"), |
|  | rect = ggplot2::element\_rect(fill = white, colour = blue, |
|  | size = 0.5, linetype = 1), |
|  | text = ggplot2::element\_text(family = base\_family, face = "plain", |
|  | colour = blue, size = base\_size, |
|  | lineheight = 0.9, hjust = 0.5, vjust = 0.5, angle = 0, |
|  | margin = ggplot2::margin(), debug = FALSE), |
|  |  |
|  | # Axes |
|  | axis.line = ggplot2::element\_blank(), |
|  | axis.text = ggplot2::element\_text(size = ggplot2::rel(0.8)), |
|  | axis.ticks = ggplot2::element\_line(color = grey, size = ggplot2::rel(1/3)), |
|  | axis.title = ggplot2::element\_text(size = ggplot2::rel(1.0)), |
|  |  |
|  | # Panel |
|  | panel.background = ggplot2::element\_rect(fill = white, color = NA), |
|  | panel.border = ggplot2::element\_rect(fill = NA, size = ggplot2::rel(1/2), color = blue), |
|  | panel.grid.major = ggplot2::element\_line(color = grey, size = ggplot2::rel(1/3)), |
|  | panel.grid.minor = ggplot2::element\_line(color = grey, size = ggplot2::rel(1/3)), |
|  | panel.grid.minor.x = ggplot2::element\_blank(), |
|  | panel.spacing = ggplot2::unit(.75, "cm"), |
|  |  |
|  | # Legend |
|  | legend.key = ggplot2::element\_rect(fill = white, color = NA), |
|  | legend.position = "bottom", |
|  |  |
|  | # Strip (Used with multiple panels) |
|  | strip.background = ggplot2::element\_rect(fill = blue, color = blue), |
|  | strip.text = ggplot2::element\_text(color = white, size = ggplot2::rel(0.8), margin = ggplot2::margin(t = 5, b = 5)), |
|  |  |
|  | # Plot |
|  | plot.title = ggplot2::element\_text(size = ggplot2::rel(1.2), hjust = 0, |
|  | margin = ggplot2::margin(t = 0, r = 0, b = 4, l = 0, unit = "pt")), |
|  | plot.subtitle = ggplot2::element\_text(size = ggplot2::rel(0.9), hjust = 0, |
|  | margin = ggplot2::margin(t = 0, r = 0, b = 3, l = 0, unit = "pt")), |
|  |  |
|  | # Complete theme |
|  | complete = TRUE |
|  | ) |
|  | } |
|  |  |
|  | theme\_tq\_dark <- function(base\_size = 11, base\_family = "") { |
|  |  |
|  | # Tidyquant colors |
|  | blue <- "#2c3e50" |
|  | green <- "#18BC9C" |
|  | white <- "#FFFFFF" |
|  | grey <- "grey50" |
|  |  |
|  | # Starts with theme\_tq and then invert some colors |
|  | theme\_tq(base\_size = base\_size, base\_family = base\_family) %+replace% |
|  | ggplot2::theme( |
|  |  |
|  | # Axes |
|  | axis.ticks = ggplot2::element\_line(color = blue, size = ggplot2::rel(1/3)), |
|  |  |
|  | # Panel |
|  | panel.background = ggplot2::element\_rect(fill = grey, color = NA), |
|  | panel.grid.major = ggplot2::element\_line(color = white, size = ggplot2::rel(1/3)), |
|  | panel.grid.minor = ggplot2::element\_line(color = white, size = ggplot2::rel(1/3)), |
|  |  |
|  | # Complete theme |
|  | complete = TRUE |
|  | ) |
|  | } |
|  |  |
|  | theme\_tq\_green <- function(base\_size = 11, base\_family = "") { |
|  |  |
|  | # Tidyquant colors |
|  | blue <- "#2c3e50" |
|  | green <- "#18BC9C" |
|  | white <- "#FFFFFF" |
|  | grey <- "grey80" |
|  |  |
|  | # Starts with theme\_tq and then invert some colors |
|  | theme\_tq(base\_size = base\_size, base\_family = base\_family) %+replace% |
|  | ggplot2::theme( |
|  |  |
|  | # Axes |
|  | axis.ticks = ggplot2::element\_line(color = blue, size = ggplot2::rel(1/3)), |
|  |  |
|  | # Panel |
|  | panel.background = ggplot2::element\_rect(fill = green, color = NA), |
|  | panel.grid.major = ggplot2::element\_line(color = white, size = ggplot2::rel(1/3)), |
|  | panel.grid.minor = ggplot2::element\_line(color = white, size = ggplot2::rel(1/3)), |
|  |  |
|  | # Complete theme |
|  | complete = TRUE |
|  | ) |
|  | } |
|  |  |
|  | scale\_color\_tq <- function(..., theme = "light") { |
|  |  |
|  | pal <- switch(theme, |
|  | "light" = unname(palette\_light()) %>% rep(100), |
|  | "dark" = unname(palette\_dark()) %>% rep(100), |
|  | "green" = unname(palette\_green() %>% rep(100)) |
|  | ) |
|  |  |
|  | ggplot2::scale\_color\_manual(values = pal) |
|  | } |
|  |  |
|  | palette\_light <- function() { |
|  | c( |
|  | blue = "#2c3e50", # blue |
|  | red = "#e31a1c", # red |
|  | green = "#18BC9C", # green |
|  | yellow = "#CCBE93", # yellow |
|  | steel\_blue = "#a6cee3", # steel\_blue |
|  | navy\_blue = "#1f78b4", # navy\_blue |
|  | light\_green = "#b2df8a", # light\_green |
|  | pink = "#fb9a99", # pink |
|  | light\_orange = "#fdbf6f", # light\_orange |
|  | orange = "#ff7f00", # orange |
|  | light\_purple = "#cab2d6", # light\_purple |
|  | purple = "#6a3d9a" # purple |
|  | ) %>% toupper() |
|  | } |
|  |  |
|  | palette\_dark <- function() { |
|  | # Brighter version of palette\_light |
|  | c( |
|  | blue = "#0055AA", # blue |
|  | red = "#C40003", # red |
|  | green = "#00C19B", # green |
|  | yellow = "#EAC862", # yellow |
|  | steel\_blue = "#7FD2FF", # steel\_blue |
|  | navy\_blue = "#007ED3", # navy\_blue |
|  | light\_green = "#b2df8a", # light\_green |
|  | pink = "#FFACAA", # pink |
|  | light\_orange = "#FF9D1E", # light\_orange |
|  | lime\_green = "#C3EF00", # lime\_green |
|  | light\_purple = "#cab2d6", # light\_purple |
|  | purple = "#894FC6" # purple |
|  | ) %>% toupper() |
|  | } |
|  |  |
|  | palette\_green <- function() { |
|  | # Green compatible version of palette\_light |
|  | c( |
|  | blue = "#0055AA", # blue |
|  | red = "#C40003", # red |
|  | yellow = "#EAC862", # yellow |
|  | steel\_blue = "#7FD2FF", # steel\_blue |
|  | navy\_blue = "#007ED3", # navy\_blue |
|  | creme = "#F6F4F3", # creme |
|  | pink = "#FFACAA", # pink |
|  | light\_orange = "#FF9D1E", # light\_orange |
|  | lime\_green = "#C3EF00", # lime\_green |
|  | light\_purple = "#cab2d6", # light\_purple |
|  | purple = "#894FC6", # purple |
|  | brown = "#592E2E" # brown |
|  | ) %>% toupper() |
|  | } |
|  |  |
|  | palette\_light <- function() { |
|  | c( |
|  | blue = "#2c3e50", # blue |
|  | red = "#e31a1c", # red |
|  | green = "#18BC9C", # green |
|  | yellow = "#CCBE93", # yellow |
|  | steel\_blue = "#a6cee3", # steel\_blue |
|  | navy\_blue = "#1f78b4", # navy\_blue |
|  | light\_green = "#b2df8a", # light\_green |
|  | pink = "#fb9a99", # pink |
|  | light\_orange = "#fdbf6f", # light\_orange |
|  | orange = "#ff7f00", # orange |
|  | light\_purple = "#cab2d6", # light\_purple |
|  | purple = "#6a3d9a" # purple |
|  | ) %>% toupper() |
|  | } |

Time\_Apply.R - Apply a function to a time series by period

|  |
| --- |
| time\_apply <- function(data, target, period, .fun, ..., |
|  | start\_date = NULL, side = "end", clean = FALSE, message = TRUE) { |
|  |  |
|  | UseMethod("time\_apply", data) |
|  |  |
|  | } |
|  |  |
|  | #' @export |
|  | time\_apply.default <- function(data, target, period, .fun, ..., |
|  | start\_date = NULL, side = "end", clean = FALSE, message = TRUE) { |
|  | stop("Object is not of class `tbl\_df` or `tbl\_time`.", call. = FALSE) |
|  | } |
|  |  |
|  |  |
|  | #' @export |
|  | time\_apply.data.frame <- function(data, target, period, .fun, ..., |
|  | start\_date = NULL, side = "end", clean = FALSE, message = TRUE) { |
|  |  |
|  | # Checks |
|  | if (missing(target)) stop('Error in time\_apply(): argument "target" is missing, with no default', call. = FALSE) |
|  | if (missing(period)) stop('Error in time\_apply(): argument "period" is missing, with no default', call. = FALSE) |
|  | if (missing(.fun)) stop('Error in time\_apply(): argument ".fun" is missing, with no default', call. = FALSE) |
|  |  |
|  |  |
|  | # Setup inputs |
|  | data <- prep\_tbl\_time(data, message = F) |
|  |  |
|  | date\_col\_expr <- tibbletime::get\_index\_quo(data) |
|  | date\_col\_name <- dplyr::quo\_name(date\_col\_expr) |
|  |  |
|  | target\_expr <- dplyr::enquo(target) |
|  |  |
|  | # Function apply logic |
|  | if (is.character(period)) { |
|  | # See collapse\_by for valid character sequences (e.g. "1 Y") |
|  | ret <- data %>% |
|  | tibbletime::collapse\_by(period = period, clean = clean, start\_date = start\_date, side = side) %>% |
|  | dplyr::group\_by(!! tibbletime::get\_index\_quo(.)) %>% |
|  | dplyr::mutate(time\_apply = .fun(!! target\_expr, ...)) %>% |
|  | dplyr::ungroup() %>% |
|  | dplyr::mutate(!! date\_col\_name := data %>% dplyr::pull(!! date\_col\_expr)) |
|  |  |
|  | } else { |
|  | # Numeric (e.g. every 15 data points) |
|  | ret <- data %>% |
|  | dplyr::mutate( |
|  | .period\_groups = c(0, (1:(nrow(.) - 1) %/% period)) |
|  | ) %>% |
|  | dplyr::group\_by(.period\_groups) %>% |
|  | dplyr::mutate( |
|  | time\_apply = .fun(!! target\_expr, ...) |
|  | ) %>% |
|  | dplyr::ungroup() %>% |
|  | dplyr::select(-.period\_groups) |
|  | } |
|  |  |
|  | return(ret) |
|  |  |
|  | } |
|  |  |
|  | #' @export |
|  | time\_apply.grouped\_df <- function(data, target, period, .fun, ..., |
|  | start\_date = NULL, side = "end", clean = FALSE, message = TRUE) { |
|  |  |
|  | # Checks |
|  | if (missing(target)) stop('Error in time\_apply(): argument "target" is missing, with no default', call. = FALSE) |
|  | if (missing(period)) stop('Error in time\_apply(): argument "period" is missing, with no default', call. = FALSE) |
|  | if (missing(.fun)) stop('Error in time\_apply(): argument ".fun" is missing, with no default', call. = FALSE) |
|  |  |
|  |  |
|  | # Setup |
|  | data <- prep\_tbl\_time(data, message = F) |
|  |  |
|  | target\_expr <- dplyr::enquo(target) |
|  |  |
|  | # Map time\_apply.data.frame |
|  | ret <- data %>% |
|  | grouped\_mapper( |
|  | .f = time\_apply, |
|  | target = !! target\_expr, |
|  | period = period, |
|  | .fun = .fun, |
|  | ... = ..., |
|  | start\_date = start\_date, |
|  | side = side, |
|  | clean = clean, |
|  | message = message) |
|  |  |
|  | return(ret) |
|  |  |
|  | } |

Time\_Decompose.R - Decompose a time series in preparation for anomaly detection

|  |
| --- |
| time\_decompose <- function(data, target, method = c("stl", "twitter"), |
|  | frequency = "auto", trend = "auto", ..., merge = FALSE, message = TRUE) { |
|  | UseMethod("time\_decompose", data) |
|  | } |
|  |  |
|  | #' @export |
|  | time\_decompose.default <- function(data, target, method = c("stl", "twitter"), |
|  | frequency = "auto", trend = "auto", ..., merge = FALSE, message = TRUE) { |
|  | stop("Error time\_decompose(): Object is not of class `tbl\_df` or `tbl\_time`.", call. = FALSE) |
|  | } |
|  |  |
|  | #' @export |
|  | time\_decompose.tbl\_time <- function(data, target, method = c("stl", "twitter"), |
|  | frequency = "auto", trend = "auto", ..., merge = FALSE, message = TRUE) { |
|  |  |
|  | # Checks |
|  | if (missing(target)) stop('Error in time\_decompose(): argument "target" is missing, with no default', call. = FALSE) |
|  |  |
|  | # Setup |
|  | target\_expr <- dplyr::enquo(target) |
|  | method <- tolower(method[[1]]) |
|  |  |
|  | # Set method |
|  | if (method == "twitter") { |
|  | decomp\_tbl <- data %>% |
|  | decompose\_twitter(!! target\_expr, frequency = frequency, trend = trend, message = message, ...) |
|  | } else if (method == "stl") { |
|  | decomp\_tbl <- data %>% |
|  | decompose\_stl(!! target\_expr, frequency = frequency, trend = trend, message = message, ...) |
|  | # } else if (method == "multiplicative") { |
|  | # decomp\_tbl <- data %>% |
|  | # decompose\_multiplicative(!! target\_expr, frequency = frequency, message = message, ...) |
|  | } else { |
|  | stop(paste0("method = '", method[[1]], "' is not a valid option.")) |
|  | } |
|  |  |
|  | # Merge if desired |
|  | if (merge) { |
|  | ret <- merge\_two\_tibbles(data, decomp\_tbl, .f = time\_decompose) |
|  | } else { |
|  | ret <- decomp\_tbl |
|  | } |
|  |  |
|  | return(ret) |
|  |  |
|  | } |
|  |  |
|  | #' @export |
|  | time\_decompose.tbl\_df <- function(data, target, method = c("stl", "twitter"), |
|  | frequency = "auto", trend = "auto", ..., merge = FALSE, message = TRUE) { |
|  |  |
|  | # Checks |
|  | if (missing(target)) stop('Error in time\_decompose(): argument "target" is missing, with no default', call. = FALSE) |
|  |  |
|  | # Prep |
|  | data <- prep\_tbl\_time(data, message = message) |
|  |  |
|  | # Send to time\_decompose.tbl\_time |
|  | time\_decompose(data = data, |
|  | target = !! dplyr::enquo(target), |
|  | method = method[[1]], |
|  | frequency = frequency, |
|  | trend = trend, |
|  | ... = ..., |
|  | merge = merge, |
|  | message = message) |
|  |  |
|  | } |
|  |  |
|  |  |
|  |  |
|  |  |
|  | #' @export |
|  | time\_decompose.grouped\_tbl\_time <- function(data, target, method = c("stl", "twitter"), |
|  | frequency = "auto", trend = "auto", ..., merge = FALSE, message = FALSE) { |
|  |  |
|  | # Checks |
|  | if (missing(target)) stop('Error in time\_decompose(): argument "target" is missing, with no default', call. = FALSE) |
|  |  |
|  | # Setup |
|  | target\_expr <- dplyr::enquo(target) |
|  |  |
|  | # Mapping |
|  | ret <- data %>% |
|  | grouped\_mapper( |
|  | .f = time\_decompose, |
|  | target = !! target\_expr, |
|  | method = method[[1]], |
|  | frequency = frequency, |
|  | trend = trend, |
|  | ... = ..., |
|  | merge = merge, |
|  | message = message) |
|  |  |
|  | return(ret) |
|  |  |
|  | } |
|  |  |
|  | #' @export |
|  | time\_decompose.grouped\_df <- function(data, target, method = c("stl", "twitter"), |
|  | frequency = "auto", trend = "auto", ..., merge = FALSE, message = FALSE) { |
|  |  |
|  | data <- prep\_tbl\_time(data, message = message) |
|  |  |
|  | # Send to grouped\_tbl\_time |
|  | time\_decompose(data = data, |
|  | target = !! dplyr::enquo(target), |
|  | method = method[[1]], |
|  | frequency = frequency, |
|  | trend = trend, |
|  | ... = ..., |
|  | merge = merge, |
|  | message = message) |
|  |  |
|  | } |

Time\_Decompose\_methods.R - Methods that power time\_decompose()

|  |
| --- |
|  |
| decompose\_twitter <- function(data, target, frequency = "auto", trend = "auto", message = TRUE) { |
|  |  |
|  | # Checks |
|  | if (missing(target)) stop('Error in decompose\_twitter(): argument "target" is missing, with no default', call. = FALSE) |
|  | # if (!is.null(median\_spans)) |
|  | # if (!is.numeric(median\_spans)) stop('Error in decompse\_twitter(): argument "median\_spans" must be numeric.', call. = FALSE) |
|  |  |
|  | data <- prep\_tbl\_time(data) |
|  | date\_col\_vals <- tibbletime::get\_index\_col(data) |
|  |  |
|  | target\_expr <- dplyr::enquo(target) |
|  |  |
|  | date\_col\_name <- timetk::tk\_get\_timeseries\_variables(data)[[1]] |
|  | date\_col\_expr <- rlang::sym(date\_col\_name) |
|  |  |
|  | freq <- time\_frequency(data, period = frequency, message = message) |
|  | # trnd <- time\_trend(data, period = trend) |
|  |  |
|  | # Time Series Decomposition |
|  | decomp\_tbl <- data %>% |
|  | dplyr::pull(!! target\_expr) %>% |
|  | stats::ts(frequency = freq) %>% |
|  | stats::stl(s.window = "periodic", robust = TRUE) %>% |
|  | sweep::sw\_tidy\_decomp() %>% |
|  | dplyr::select(-c(index, seasadj)) %>% |
|  | # forecast::mstl() %>% |
|  | # as.tibble() %>% |
|  | tibble::add\_column(!! date\_col\_name := date\_col\_vals, .after = 0) %>% |
|  | purrr::set\_names(c(date\_col\_name, "observed", "season", "trend", "remainder")) %>% |
|  | dplyr::mutate(seasadj = observed - season) %>% |
|  | dplyr::select(!! date\_col\_expr, observed, season, seasadj, trend, remainder) |
|  |  |
|  | # Median Span Logic |
|  | trnd <- time\_trend(data, period = trend, message = FALSE) |
|  | median\_spans\_needed <- round(nrow(data) / trnd) |
|  |  |
|  | decomp\_tbl <- decomp\_tbl %>% |
|  | dplyr::mutate( |
|  | .period\_groups = rep(1:median\_spans\_needed, length.out = nrow(.)) %>% sort() |
|  | ) %>% |
|  | dplyr::group\_by(.period\_groups) %>% |
|  | dplyr::mutate(median\_spans = median(observed, na.rm = T)) %>% |
|  | dplyr::ungroup() %>% |
|  | dplyr::select(-.period\_groups) |
|  |  |
|  | if (message) { |
|  | med\_span <- decomp\_tbl %>% |
|  | dplyr::count(median\_spans) %>% |
|  | dplyr::pull(n) %>% |
|  | median(na.rm = TRUE) |
|  |  |
|  | med\_scale <- decomp\_tbl %>% |
|  | timetk::tk\_index() %>% |
|  | timetk::tk\_get\_timeseries\_summary() %>% |
|  | dplyr::pull(scale) |
|  |  |
|  | message(glue::glue("median\_span = {med\_span} {med\_scale}s")) |
|  | } |
|  |  |
|  | # Remainder calculation |
|  | decomp\_tbl <- decomp\_tbl %>% |
|  | dplyr::mutate( |
|  | remainder = observed - season - median\_spans |
|  | ) %>% |
|  | dplyr::select(!! date\_col\_expr, observed, season, median\_spans, remainder) |
|  |  |
|  | decomp\_tbl <- anomalize::prep\_tbl\_time(decomp\_tbl) |
|  |  |
|  | return(decomp\_tbl) |
|  |  |
|  | } |
|  |  |

|  |
| --- |
|  |
| decompose\_stl <- function(data, target, frequency = "auto", trend = "auto", message = TRUE) { |
|  |  |
|  | # Checks |
|  | if (missing(target)) stop('Error in decompose\_stl(): argument "target" is missing, with no default', call. = FALSE) |
|  |  |
|  |  |
|  | data <- prep\_tbl\_time(data) |
|  | date\_col\_vals <- tibbletime::get\_index\_col(data) |
|  |  |
|  | target\_expr <- dplyr::enquo(target) |
|  |  |
|  | date\_col\_name <- timetk::tk\_get\_timeseries\_variables(data)[[1]] |
|  | date\_col\_expr <- rlang::sym(date\_col\_name) |
|  |  |
|  | freq <- time\_frequency(data, period = frequency, message = message) |
|  | trnd <- time\_trend(data, period = trend, message = message) |
|  |  |
|  | # Time Series Decomposition |
|  | decomp\_tbl <- data %>% |
|  | dplyr::pull(!! target\_expr) %>% |
|  | stats::ts(frequency = freq) %>% |
|  | stats::stl(s.window = "periodic", t.window = trnd, robust = TRUE) %>% |
|  | sweep::sw\_tidy\_decomp() %>% |
|  | # forecast::mstl() %>% |
|  | # as.tibble() %>% |
|  | tibble::add\_column(!! date\_col\_name := date\_col\_vals, .after = 0) %>% |
|  | dplyr::select(!! date\_col\_expr, observed, season, trend, remainder) |
|  |  |
|  | decomp\_tbl <- anomalize::prep\_tbl\_time(decomp\_tbl) |
|  |  |
|  | return(decomp\_tbl) |
|  |  |
|  | } |
|  |  |

Time\_Frequency.R - Generate a time series frequency from a periodicity

|  |
| --- |
| time\_frequency <- function(data, period = "auto", message = TRUE) { |
|  |  |
|  | # Checks |
|  | if (!is.data.frame(data)) stop("Error time\_frequency(): Object must inherit class `data.frame`, `tbl\_df` or `tbl\_time`.") |
|  |  |
|  | if (dplyr::is.grouped\_df(data)) |
|  | stop(glue::glue("Error time\_frequency(): Cannot use on a grouped data frame. |
|  | Frequency should be performed on a single time series.")) |
|  |  |
|  | # Setup inputs |
|  | template <- get\_time\_scale\_template() |
|  | data <- prep\_tbl\_time(data, message = F) |
|  |  |
|  | index\_expr <- data %>% tibbletime::get\_index\_quo() |
|  | index\_name <- dplyr::quo\_name(index\_expr) |
|  |  |
|  | # Get timeseries summary attributes |
|  | ts\_summary <- data %>% |
|  | tibbletime::get\_index\_col() %>% |
|  | timetk::tk\_get\_timeseries\_summary() |
|  |  |
|  | ts\_nobs <- ts\_summary$n.obs |
|  | ts\_scale <- ts\_summary$scale |
|  |  |
|  |  |
|  | if (is.numeric(period)) { |
|  | # 1. Numeric Periods |
|  | freq <- period |
|  |  |
|  | } else if (period != "auto") { |
|  | # 2. Text (e.g. period = "2 Weeks") |
|  | freq <- data %>% |
|  | tibbletime::collapse\_by(period = period) %>% |
|  | dplyr::count(!! index\_expr) %>% |
|  | dplyr::pull(n) %>% |
|  | stats::median(na.rm = T) |
|  |  |
|  | } else { |
|  | # 3. period = "auto" |
|  |  |
|  | periodicity\_target <- template %>% |
|  | target\_time\_decomposition\_scale(time\_scale = ts\_scale, target = "frequency", index\_shift = 0) |
|  |  |
|  | freq <- data %>% |
|  | tibbletime::collapse\_by(period = periodicity\_target) %>% |
|  | dplyr::count(!! index\_expr) %>% |
|  | dplyr::pull(n) %>% |
|  | stats::median(na.rm = T) |
|  |  |
|  | # Insufficient observations: nobs-to-freq should be at least 3-1 |
|  | if (ts\_nobs < 3\*freq) { |
|  | periodicity\_target <- template %>% |
|  | target\_time\_decomposition\_scale(time\_scale = ts\_scale, target = "frequency", index\_shift = 1) |
|  |  |
|  | freq <- data %>% |
|  | tibbletime::collapse\_by(period = periodicity\_target) %>% |
|  | dplyr::count(!! index\_expr) %>% |
|  | dplyr::pull(n) %>% |
|  | stats::median(na.rm = T) |
|  | } |
|  |  |
|  | if (ts\_nobs < 3\*freq) { |
|  | freq <- 1 |
|  | } |
|  | } |
|  |  |
|  | if (message) { |
|  | freq\_string <- glue::glue("frequency = {freq} {ts\_scale}s") |
|  | message(freq\_string) |
|  | } |
|  |  |
|  | return(freq) |
|  | } |
|  |  |
|  | #' @export |
|  | #' @rdname time\_frequency |
|  | time\_trend <- function(data, period = "auto", message = TRUE) { |
|  |  |
|  | # Checks |
|  | if (!is.data.frame(data)) stop("Error time\_trend(): Object must inherit class `data.frame`, `tbl\_df` or `tbl\_time`.") |
|  |  |
|  | if (dplyr::is.grouped\_df(data)) |
|  | stop(glue::glue("Cannot use on a grouped data frame. |
|  | Frequency should be performed on a single time series.")) |
|  |  |
|  | # Setup inputs |
|  | template <- get\_time\_scale\_template() |
|  | data <- prep\_tbl\_time(data, message = F) |
|  |  |
|  | index\_expr <- data %>% tibbletime::get\_index\_quo() |
|  | index\_name <- dplyr::quo\_name(index\_expr) |
|  |  |
|  | # Get timeseries summary attributes |
|  | ts\_summary <- data %>% |
|  | tibbletime::get\_index\_col() %>% |
|  | timetk::tk\_get\_timeseries\_summary() |
|  |  |
|  | ts\_nobs <- ts\_summary$n.obs |
|  | ts\_scale <- ts\_summary$scale |
|  |  |
|  |  |
|  | if (is.numeric(period)) { |
|  | # 1. Numeric Periods |
|  | trend <- period |
|  |  |
|  | } else if (period != "auto") { |
|  | # 2. Text (e.g. period = "2 Weeks") |
|  | trend <- data %>% |
|  | tibbletime::collapse\_by(period = period) %>% |
|  | dplyr::count(!! index\_expr) %>% |
|  | dplyr::pull(n) %>% |
|  | stats::median(na.rm = T) |
|  |  |
|  | } else { |
|  | # 3. period = "auto" |
|  |  |
|  | periodicity\_target <- template %>% |
|  | target\_time\_decomposition\_scale(time\_scale = ts\_scale, target = "trend", index\_shift = 0) |
|  |  |
|  | trend <- data %>% |
|  | tibbletime::collapse\_by(period = periodicity\_target) %>% |
|  | dplyr::count(!! index\_expr) %>% |
|  | dplyr::pull(n) %>% |
|  | stats::median(na.rm = T) |
|  |  |
|  | # Insufficient observations: nobs-to-trend should be at least 2-1 |
|  | if (ts\_nobs / trend < 2) { |
|  | periodicity\_target <- template %>% |
|  | target\_time\_decomposition\_scale(time\_scale = ts\_scale, target = "trend", index\_shift = 1) |
|  |  |
|  | trend <- data %>% |
|  | tibbletime::collapse\_by(period = periodicity\_target) %>% |
|  | dplyr::count(!! index\_expr) %>% |
|  | dplyr::pull(n) %>% |
|  | stats::median(na.rm = T) |
|  |  |
|  | trend <- ceiling(trend) |
|  |  |
|  | } |
|  |  |
|  | if (ts\_nobs / trend < 2) { |
|  | trend <- ts\_nobs |
|  | } |
|  | } |
|  |  |
|  | if (message) { |
|  | trend\_string <- glue::glue("trend = {trend} {ts\_scale}s") |
|  | message(trend\_string) |
|  | } |
|  |  |
|  | return(trend) |
|  | } |
|  |  |
|  | # Helper function to get the time decomposition scale |
|  | target\_time\_decomposition\_scale <- function(template, time\_scale, target = c("frequency", "trend"), index\_shift = 0) { |
|  |  |
|  | target\_expr <- rlang::sym(target[[1]]) |
|  |  |
|  | idx <- which(template$time\_scale == time\_scale) - index\_shift |
|  | key\_value <- template$time\_scale[idx] |
|  |  |
|  | template %>% |
|  | dplyr::filter(time\_scale == key\_value) %>% |
|  | dplyr::pull(!! target\_expr) |
|  | } |

Time\_recompose.R- Recompose bands separating anomalies from "normal" observations

|  |
| --- |
| time\_recompose <- function(data) { |
|  | UseMethod("time\_recompose", data) |
|  | } |
|  |  |
|  | #' @export |
|  | time\_recompose.default <- function(data) { |
|  | stop("Error time\_recompose(): Object is not of class `tbl\_df` or `tbl\_time`.", call. = FALSE) |
|  | } |
|  |  |
|  | #' @export |
|  | time\_recompose.tbl\_time <- function(data) { |
|  |  |
|  | # Checks |
|  | column\_names <- names(data) |
|  | check\_names <- c("observed", "remainder", "remainder\_l1", "remainder\_l2") %in% column\_names |
|  | if (!all(check\_names)) stop('Error in time\_recompose(): key names are missing. Make sure observed:remainder, remainder\_l1, and remainder\_l2 are present', call. = FALSE) |
|  |  |
|  | # Setup |
|  | # target\_expr <- dplyr::enquo(target) |
|  | # method <- tolower(method[[1]]) |
|  |  |
|  | l1 <- data %>% |
|  | dplyr::select(observed:remainder, contains("\_l1")) %>% |
|  | dplyr::select(-c(observed, remainder)) %>% |
|  | apply(MARGIN = 1, FUN = sum) |
|  |  |
|  | l2 <- data %>% |
|  | dplyr::select(observed:remainder, contains("\_l2")) %>% |
|  | dplyr::select(-c(observed, remainder)) %>% |
|  | apply(MARGIN = 1, FUN = sum) |
|  |  |
|  | ret <- data %>% |
|  | # add\_column(!! paste0(quo\_name(target\_expr), "\_l1") := l1) |
|  | tibble::add\_column( |
|  | recomposed\_l1 = l1, |
|  | recomposed\_l2 = l2 |
|  | ) |
|  |  |
|  | return(ret) |
|  |  |
|  | } |
|  |  |
|  | #' @export |
|  | time\_recompose.tbl\_df <- function(data) { |
|  |  |
|  | # Prep |
|  | data <- prep\_tbl\_time(data, message = FALSE) |
|  |  |
|  | # Send to time\_recompose.tbl\_time |
|  | time\_recompose(data = data) |
|  |  |
|  | } |
|  |  |
|  |  |
|  | #' @export |
|  | time\_recompose.grouped\_tbl\_time <- function(data) { |
|  |  |
|  | # Checks |
|  | column\_names <- names(data) |
|  | check\_names <- c("observed", "remainder", "remainder\_l1", "remainder\_l2") %in% column\_names |
|  | if (!all(check\_names)) stop('Error in time\_recompose(): key names are missing. Make sure observed:remainder, remainder\_l1, and remainder\_l2 are present', call. = FALSE) |
|  |  |
|  | # Setup |
|  | group\_names <- dplyr::groups(data) |
|  | group\_vars\_expr <- rlang::syms(group\_names) |
|  |  |
|  | # Recompose l1 and l2 bands |
|  | l1 <- data %>% |
|  | dplyr::ungroup() %>% |
|  | dplyr::select(observed:remainder, contains("\_l1")) %>% |
|  | dplyr::select(-c(observed, remainder)) %>% |
|  | apply(MARGIN = 1, FUN = sum) |
|  |  |
|  | l2 <- data %>% |
|  | dplyr::ungroup() %>% |
|  | dplyr::select(observed:remainder, contains("\_l2")) %>% |
|  | dplyr::select(-c(observed, remainder)) %>% |
|  | apply(MARGIN = 1, FUN = sum) |
|  |  |
|  | ret <- data %>% |
|  | dplyr::ungroup() %>% |
|  | tibble::add\_column( |
|  | recomposed\_l1 = l1, |
|  | recomposed\_l2 = l2 |
|  | ) %>% |
|  | dplyr::group\_by(!!! group\_vars\_expr) |
|  |  |
|  | return(ret) |
|  |  |
|  | } |
|  |  |
|  | #' @export |
|  | time\_recompose.grouped\_df <- function(data) { |
|  |  |
|  | data <- prep\_tbl\_time(data, message = message) |
|  |  |
|  | # Send to grouped\_tbl\_time |
|  | time\_recompose(data = data) |
|  |  |
|  | } |

Time\_Scale\_Template.R - Get and modify time scale template

|  |
| --- |
| set\_time\_scale\_template <- function(data) { |
|  | if (!missing(data)) { |
|  | options(time\_scale\_template = data) |
|  | } |
|  | #getOption('time\_scale\_template') |
|  | } |
|  |  |
|  | #' @export |
|  | #' @rdname time\_scale\_template |
|  | get\_time\_scale\_template <- function() { |
|  | getOption('time\_scale\_template') |
|  | } |
|  |  |
|  | #' @export |
|  | #' @rdname time\_scale\_template |
|  | time\_scale\_template <- function() { |
|  |  |
|  | tibble::tribble( |
|  | ~ "time\_scale", ~ "frequency", ~ "trend", |
|  | "second", "1 hour", "12 hours", |
|  | "minute", "1 day", "14 days", |
|  | "hour", "1 day", "1 month", |
|  | "day", "1 week", "3 months", |
|  | "week", "1 quarter", "1 year", |
|  | "month", "1 year", "5 years", |
|  | "quarter", "1 year", "10 years", |
|  | "year", "5 years", "30 years" |
|  | ) |
|  |  |
|  | } |

Utils.R - UTILITY FUNCTIONS ----

|  |
| --- |
| 1. Mapping Functions ----- |
|  |  |
|  | grouped\_mapper <- function(data, target, .f, ...) { |
|  |  |
|  | data <- prep\_tbl\_time(data, message = F) |
|  |  |
|  | target\_expr <- dplyr::enquo(target) |
|  |  |
|  | group\_names <- dplyr::group\_vars(data) |
|  |  |
|  | ret <- data %>% |
|  | dplyr::group\_nest() %>% |
|  | dplyr::mutate(nested.col = purrr::map( |
|  | .x = data, |
|  | .f = .f, |
|  | target = !! target\_expr, |
|  | ...) |
|  | ) %>% |
|  | dplyr::select(-data) %>% |
|  | tidyr::unnest(cols = nested.col) %>% |
|  | dplyr::group\_by\_at(.vars = group\_names) |
|  |  |
|  | # if (merge) { |
|  | # ret <- merge\_two\_tibbles(tib1 = data, tib2 = ret, .f = .f) |
|  | # } |
|  |  |
|  | return(ret) |
|  |  |
|  | } |
|  |  |
|  | # 2. Merging Time-Based Tibbles ----- |
|  |  |
|  | merge\_two\_tibbles <- function(tib1, tib2, .f) { |
|  |  |
|  | # Merge results |
|  | if (identical(nrow(tib1), nrow(tib2))) { |
|  |  |
|  | # Arrange dates - Possibility of issue if dates not decending in tib1 |
|  | tib1 <- arrange\_by\_date(tib1) |
|  |  |
|  | # Drop date column and groups |
|  | tib2 <- drop\_date\_and\_group\_cols(tib2) |
|  |  |
|  | # Replace bad names |
|  | tib2 <- replace\_bad\_names(tib2, .f) |
|  |  |
|  | # Replace duplicate names |
|  | tib2 <- replace\_duplicate\_colnames(tib1, tib2) |
|  |  |
|  | ret <- dplyr::bind\_cols(tib1, tib2) |
|  |  |
|  | } else { |
|  |  |
|  | stop("Could not join. Incompatible structures.") |
|  | } |
|  |  |
|  | return(ret) |
|  | } |
|  |  |
|  | replace\_duplicate\_colnames <- function(tib1, tib2) { |
|  |  |
|  | # Collect column names |
|  | name\_list\_tib1 <- colnames(tib1) |
|  | name\_list\_tib2 <- colnames(tib2) |
|  | name\_list <- c(name\_list\_tib1, name\_list\_tib2) |
|  |  |
|  | duplicates\_exist <- detect\_duplicates(name\_list) |
|  |  |
|  | # Iteratively add .1, .2, .3 ... onto end of column names |
|  | if (duplicates\_exist) { |
|  |  |
|  | i <- 1 |
|  |  |
|  | while (duplicates\_exist) { |
|  |  |
|  | dup\_names\_stripped <- |
|  | strsplit(name\_list[duplicated(name\_list)], |
|  | split = "\\.\\.") %>% |
|  | sapply(function(x) x[[1]]) |
|  |  |
|  | name\_list[duplicated(name\_list)] <- |
|  | paste0(dup\_names\_stripped, "..", i) |
|  |  |
|  | i <- i + 1 |
|  |  |
|  | duplicates\_exist <- detect\_duplicates(name\_list) |
|  |  |
|  | } |
|  |  |
|  | name\_list\_tib2 <- name\_list[(ncol(tib1) + 1):length(name\_list)] |
|  |  |
|  | colnames(tib2) <- name\_list\_tib2 |
|  | } |
|  |  |
|  | return(tib2) |
|  | } |
|  |  |
|  | detect\_duplicates <- function(name\_list) { |
|  |  |
|  | name\_list %>% |
|  | duplicated() %>% |
|  | any() |
|  | } |
|  |  |
|  | # bad / restricted names are names that get selected unintetionally by OHLC functions |
|  | replace\_bad\_names <- function(tib, fun\_name) { |
|  |  |
|  | bad\_names\_regex <- "open|high|low|close|volume|adjusted|price" |
|  |  |
|  | name\_list\_tib <- colnames(tib) |
|  | name\_list\_tib\_lower <- tolower(name\_list\_tib) |
|  |  |
|  | detect\_bad\_names <- grepl(pattern = bad\_names\_regex, |
|  | x = name\_list\_tib\_lower) |
|  |  |
|  | if (any(detect\_bad\_names)) { |
|  |  |
|  | len <- length(name\_list\_tib\_lower[detect\_bad\_names]) |
|  | name\_list\_tib[detect\_bad\_names] <- rep(fun\_name, length.out = len) |
|  |  |
|  | } |
|  |  |
|  | colnames(tib) <- name\_list\_tib |
|  |  |
|  | return(tib) |
|  | } |
|  |  |
|  | arrange\_by\_date <- function(tib) { |
|  |  |
|  | if (dplyr::is.grouped\_df(tib)) { |
|  |  |
|  | group\_names <- dplyr::group\_vars(tib) |
|  |  |
|  | arrange\_date <- function(tib) { |
|  | date\_col <- timetk::tk\_get\_timeseries\_variables(tib)[[1]] |
|  | tib %>% |
|  | dplyr::arrange(!! rlang::sym(date\_col)) |
|  | } |
|  |  |
|  | tib <- tib %>% |
|  | tidyr::nest() %>% |
|  | dplyr::mutate(nested.col = |
|  | purrr::map(data, arrange\_date) |
|  | ) %>% |
|  | dplyr::select(-data) %>% |
|  | tidyr::unnest(cols = nested.col) %>% |
|  | dplyr::group\_by\_at(.vars = group\_names) |
|  |  |
|  |  |
|  | } else { |
|  | date\_col <- timetk::tk\_get\_timeseries\_variables(tib)[[1]] |
|  | tib <- tib %>% |
|  | dplyr::arrange(!! rlang::sym(date\_col)) |
|  |  |
|  | } |
|  |  |
|  | return(tib) |
|  | } |
|  |  |
|  | drop\_date\_and\_group\_cols <- function(tib) { |
|  |  |
|  | date\_col <- timetk::tk\_get\_timeseries\_variables(tib)[[1]] |
|  | group\_cols <- dplyr::groups(tib) %>% |
|  | as.character() |
|  | cols\_to\_remove <- c(date\_col, group\_cols) |
|  | tib\_names <- colnames(tib) |
|  | cols\_to\_remove\_logical <- tib\_names %in% cols\_to\_remove |
|  | tib\_names\_without\_date\_or\_group <- tib\_names[!cols\_to\_remove\_logical] |
|  |  |
|  | tib <- tib %>% |
|  | dplyr::ungroup() %>% |
|  | dplyr::select(!!! rlang::syms(tib\_names\_without\_date\_or\_group)) |
|  |  |
|  | return(tib) |
|  | } |