# Package 'healthyR.ts'

February 9, 2021

**Title** The Time Series Modeling Companion to 'healthyR'

**Version** 0.1.1

**Description** Hospital time series data analysis workflow tools, modeling, and automations. This library provides many useful tools to review common administrative time series hospital data. Some of these include average length of stay, and readmission rates. The aim is to provide a simple and consistent verb framework that takes the guesswork out of everything.

**License** MIT + file LICENSE

**Encoding** UTF-8

**LazyData** true

**RoxygenNote** 7.1.1

**URL** https://github.com/spsanderson/healthyR.ts

**BugReports** https://github.com/spsanderson/healthyR.ts/issues

**Imports** magrittr, rlang (>= 0.1.2), tibble, timetk, modeltime, modeltime.ensemble, modeltime.resample, dplyr, purrr, ggplot2, tidyquant, healthyR.data, lubridate, stringr, plotly

**Suggests** knitr, rmarkdown, roxygen2, scales

**VignetteBuilder** knitr

**Depends** R (>= 2.10)

**NeedsCompilation** no

**Author** Steven Sanderson [aut, cre], Steven Sanderson [cph]

**Maintainer** Steven Sanderson <spsanderson@gmail.com>

**Repository** CRAN

**Date/Publication** 2021-02-09 19:00:02 UTC

# R topics documented:



---

`ts_compare_data` *Compare data over time periods*

---

## Description

Given a tibble/data.frame, you can get date from two different but comparative date ranges. Lets say you want to compare visits in one year to visits from 2 years before without also seeing the previous 1 year. You can do that with this function.

## Usage

```
ts_compare_data(.data, .date_col, .start_date, .end_date, .periods_back)
```

## Arguments

| | |
|---|---|
| `.data` | The date.frame/tibble that holds the data |
| `.date_col` | The column with the date value |
| `.start_date` | The start of the period you want to analyze |
| `.end_date` | The end of the period you want to analyze |
| `.periods_back` | How long ago do you want to compare data too. Time units are collapsed using `lubridate::floor_date()`. The value can be:<br>• second<br>• minute<br>• hour<br>• day<br>• week<br>• month<br>• bimonth<br>• quarter<br>• season<br>• halfyear<br>• year<br>Arbitrary unique English abbreviations as in the `lubridate::period()` constructor are allowed. |

### Details

- Uses the `timetk::filter_by_time()` function in order to filter the date column.
- Uses the `timetk::subtract_time()` function to subtract time from the start date.

### Value

A tibble.

### Author(s)

Steven P. Sanderson II, MPH

### Examples

```
suppressPackageStartupMessages(library(healthyR.data))
suppressPackageStartupMessages(library(dplyr))
suppressPackageStartupMessages(library(timetk))
ts_compare_data(
  .data           = healthyR_data
  , .date_col     = visit_start_date_time
  , .start_date   = "2019-01-01"
  , .end_date     = "2019-12-31"
  , .periods_back = "2 years"
  ) %>%
  select(visit_start_date_time) %>%
  summarise_by_time(
    .date_var = visit_start_date_time
    , .by     = "year"
    , visits  = n()
  )

ts_compare_data(
 .data = healthyR_data
 , .date_col     = visit_end_date_time
 , .start_date   = "2019-01-01"
 , .end_date     = "2019-12-31"
 , .periods_back = "2 years"
)
```

---

## ts_qc_run_chart *Quality Control Run Chart*

---

### Description

A control chart is a specific type of graph that shows data points between upper and lower limits over a period of time. You can use it to understand if the process is in control or not. These charts commonly have three types of lines such as upper and lower specification limits, upper and lower limits and planned value. By the help of these lines, Control Charts show the process behavior over time.

## Usage

```
ts_qc_run_chart(
  .data,
  .date_col,
  .value_col,
  .interactive = FALSE,
  .median = TRUE,
  .cl = TRUE,
  .mcl = TRUE,
  .ucl = TRUE,
  .lc = FALSE,
  .lmcl = FALSE,
  .llcl = FALSE
)
```

## Arguments

| | |
|---|---|
| .data | The data.frame/tibble to be passed. |
| .date_col | The column holding the timestamp. |
| .value_col | The column with the values to be analyzed. |
| .interactive | Default is FALSE, TRUE for an interactive plotly plot. |
| .median | Default is TRUE. This will show the median line of the data. |
| .cl | This is the first upper control line |
| .mcl | This is the second sigma control line positive |
| .ucl | This is the third sigma control line positive |
| .lc | This is the first negative control line |
| .lmcl | This is the second sigma negative control line |
| .llcl | This si the thrid sigma negative control line |

## Details

- Expects a time-series tibble/data.frame
- Expects a date column and a value column

## Value

A static ggplot2 graph or if .interactive is set to TRUE a plotly plot

## Author(s)

Steven P. Sanderson II, MPH

## Examples

```
library(healthyR.data)
library(timetk)
library(dplyr)
library(stringr)

df <- healthyR_data

df_monthly_tbl <- df %>%
   mutate(ip_op_flag = str_squish(ip_op_flag)) %>%
   filter(ip_op_flag == "I") %>%
   select(visit_end_date_time, length_of_stay) %>%
   arrange(visit_end_date_time) %>%
   summarise_by_time(
       .date_var = visit_end_date_time
       , .by = "month"
       , alos = round(mean(length_of_stay, na.rm = TRUE), 2)
       , .type = "ceiling"
   ) %>%
   mutate(
     visit_end_date_time = visit_end_date_time %>%
       subtract_time("1 day")
   )

df_monthly_tbl %>%
  ts_qc_run_chart(
    .date_col    = visit_end_date_time
    , .value_col = alos
    , .llcl      = TRUE
  )
```

---

`ts_random_walk` *Random Walk Function*

---

## Description

This function takes in four arguments and retunrs a tibble of random walks.

## Usage

```
ts_random_walk(
  .mean = 0,
  .sd = 0.1,
  .num_walks = 100,
  .periods = 100,
  .initial_value = 1000
)
```

### Arguments

| | |
|---|---|
| .mean | The desired mean of the random walks |
| .sd | The standard deviation of the random walks |
| .num_walks | The number of random walks you want generated |
| .periods | The length of the random walk(s) you want generated |
| .initial_value | The intial value where the random walks should start |

### Details

Monte Carlo simulations were first formally designed in the 1940's while developing nuclear weapons, and since have been heavily used in various fields to use randomness solve problems that are potentially deterministic in nature. In finance, Monte Carlo simulations can be a useful tool to give a sense of how assets with certain characteristics might behave in the future. While there are more complex and sophisticated financial forecasting methods such as ARIMA (Auto-Regressive Integrated Moving Average) and GARCH (Generalised Auto-Regressive Conditional Heteroskedasticity) which attempt to model not only the randomness but underlying macro factors such as seasonality and volatility clustering, Monte Carlo random walks work surprisingly well in illustrating market volatility as long as the results are not taken too seriously.

### Value

A tibble

### Author(s)

Steven P. Sanderson II, MPH

### Examples

```
ts_random_walk(
.mean = 6,
.sd = 1,
.num_walks = 25,
.periods = 180,
.initial_value = 6
)
```

---

## ts_random_walk_ggplot_layers

*Get Random Walk* ggplot2 *layers*

---

### Description

Get layers to add to a ggplot graph from the ts_random_walk() function.

## Usage

```
ts_random_walk_ggplot_layers(.data)
```

## Arguments

| | |
|---|---|
| .data | The data passed to the function. |

## Details

- Set the intercept of the initial value from the random walk
- Set the max and min of the cumulative sum of the random walks

## Value

A ggplot2 layers object

## Author(s)

Steven P. Sanderson II, MPH

## Examples

```
library(ggplot2)

df <- ts_random_walk()

df %>%
  ggplot(
    mapping = aes(
      x = x
      , y = cum_y
      , color = factor(run)
      , group = factor(run)
   )
 ) +
 geom_line(alpha = 0.8) +
 ts_random_walk_ggplot_layers(df)
```

# Index