Package 'cleanTS'

May 8, 2026

Type Package

Title Testbench for Univariate Time Series Cleaning

Version 0.1.2

Description A reliable and efficient tool for cleaning univariate time series data. It implements reliable and efficient procedures for automating the process of cleaning univariate time series data. The package provides integration with already developed and deployed tools for missing value imputation and outlier detection. It also provides a way of visualizing large time-series data in different resolutions.

License GPL (>= 3)

URL https://github.com/Mayur1009/cleanTS

BugReports https://github.com/Mayur1009/cleanTS/issues

Imports data.table, gganimate, ggplot2, ggtext, transformr, glue, imputeTestbench, imputeTS, lubridate, shiny, stringr, tibble

Suggests rmarkdown, gifski (>= 1.4.3), timetk, spelling

Encoding UTF-8

RoxygenNote 7.2.3

Language en-US

NeedsCompilation no

Author Mayur Shende [aut, cre] (ORCID: <https://orcid.org/0000-0002-1738-2573>), Neeraj Bokde [aut] (ORCID: <https://orcid.org/0000-0002-3493-9302>), Andrés E. Feijóo-Lorenzo [aut] (ORCID: <https://orcid.org/0000-0003-3172-7037>)

Maintainer Mayur Shende <mayur.k.shende@gmail.com>

Repository CRAN

Date/Publication 2023-07-06 07:33:10 UTC

Contents

`animate_interval` *Generate animated plot*

Description

`animate_interval()` creates an animated plot using a `cleanTS` object and a interval.

Usage

```
animate_interval(obj, interval)
```

Arguments

`obj`	A *cleanTS* object.
`interval`	A numeric or character, specifying the viewing interval.

Details

First, the data is split according to the `interval` argument passed to the function. If it is a numeric value, the cleaned data is split into dataframes containing `interval` observations. It can also be a string, like **1 week**, **3 months**, **14 days**, etc. In this case, the data is split according to the `interval` given, using the timestamp column. Then an animation is created using the spliited data, with the help of `gganimate` package. The `animate_interval()` function returns a list containing the `gganim` object used to generate the animation and the number of states in the data. The animation can be generated using the `gen.animation()` function and saved using the `anim_save()` function. The plots in the animation also contain a short summary, containing the statistical information and the number of missing values, outliers, missing timestamps, and duplicate timestamps in the data shown in that frame of animation.

Value

A list containing:

- animation: A gganim object.
- nstates: The number of states in the animation.

Examples

```
## Not run:
  # Create a `gganim` using `animate_interval()` function
  a <- animate_interval(cts, "10 year")

  # cts -> `cleanTS` object created using `cleanTS()`.

## End(Not run)
```

---

check_input *Check input data*

Description

This function is used to check and verify the input data given as input. The package needs a univariate time series as input. This function keeps the first 2 columns, first is renamed as time and second is renamed as value. If the optional time and value arguments are provided then they are used to determine the relevant columns in the data.

Usage

```
check_input(df, dt_format, time, value)
```

Arguments

df	A data frame containing the input data. If it contains more than two columns then specify the names of time and value columns using the time and value arguments.
dt_format	Format of timestamps used in the data. It uses lubridate formats as mentioned here.
time	The name of column in provided data to be used as time column.
value	The name of column in provided data, to be used as value(observations) column.

Value

Data containing 2 columns, time and value. Time column is converted to POSIX object and value to numeric.

cleanTS *Clean univariate time-series data*

Description

`cleanTS()`is the main function of the package which creates a cleanTS object. It performs all the different data cleaning tasks, such as converting the timestamps to proper format, imputation of missing values, handling outliers, etc. It is a wrapper function that calls all the other internal functions to performs different data cleaning tasks.

Usage

```
cleanTS(
  data,
  date_format,
  imp_methods = c("na_interpolation", "na_locf", "na_ma", "na_kalman"),
  time = NULL,
  value = NULL,
  replace_outliers = TRUE
)
```

Arguments

`data`	A data frame containing the input data. By default, it considers that the first column to contain the timestamps and the second column contains the observations.If that is not the case or if it contains more than two columns then specify the names of time and value columns using the `time` and `value` arguments.
`date_format`	Format of timestamps used in the data. It uses lubridate formats as mentioned here. More than one formats can be using a vectors of strings.
`imp_methods`	The imputation methods to be used.
`time`	Optional, the name of column in provided data to be used as time column.
`value`	Optional, the name of column in provided data, to be used as value column.
`replace_outliers`	Boolean, if `TRUE` then the outliers found will be removed and imputed using the given imputation methods.

Details

The first task is to check the input time series data for structural and data type-related errors. Since the functions need univariate time series data, the input data is checked for the number of columns. By default, the first column is considered to be the time column, and the second column to be the observations. Alternatively, if the time and value arguments are given, then those columns are used. The time column is converted to a POSIX object. The value column is converted to a numeric type. The column names are also changed to time and value. All the data is converted to a *data.table* object. This data is then passed to other functions to check for missing and duplicate timestamps. If duplicate timestamps are found, then the observation values are checked. If the

observations are the same, then only one copy of that observation is kept. But if the observations are different, then it is not possible to find the correct one, so the observation is set to NA. This data is the passed to a function for finding and handling missing observations. The methods given in the imp_methods argument are compared and selected. The MCAR and MAR values are handled seperately. After the best methods are found, imputation is performed using those methods. The user can also pass user-defined functions for comparison. The user-defined function should follow the structure as the default functions. It should take a numeric vector containing missing values as input, and return a numeric vector of the same length without missing values as output. Once the missing values are handled the data is checked for outliers. If the replace_outliers parameter is set to TRUE in the cleanTS() function, then the outliers are replaced by NA and imputed using the procedure mentioned for imputing missing values. Then it creates a cleanTS object which contains the cleaned data, missing timestamps, duplicate timestamps, imputation methods, MCAR imputation error, MAR imputation error, outliers, and if the outliers are replaced then imputation errors for those imputations are also included. The *cleanTS* object is returned by the function.

Value

A `cleanTS` object which contains:

- Cleaned data
- Missing timestamps
- Duplicate timestamps
- Imputation errors
- Outliers
- Outlier imputation errors

Examples

```
## Not run:
  # Convert sunspots.month to dataframe
  data <- timetk::tk_tbl(sunspot.month)
  print(data)

  # Randomly insert missing values to simulate missing value imputation
  set.seed(10)
  ind <- sample(nrow(data), 100)
  data$value[ind] <- NA

  # Perform cleaning
  cts <- cleanTS(data, date_format = "my", time = "index", value = "value")
  print(cts)

## End(Not run)
```

detect_outliers *Find outliers in the data*

Description

This function detects outliers/anomalies in the data. If the `replace_outlier` argument is set to `TRUE`, then the outliers are removed and imputed using the provided imputation methods.

Usage

```
detect_outliers(dt, replace_outlier, imp_methods)
```

Arguments

`dt`	A data.table.
`replace_outlier`	Boolean, defaults to `TRUE`. Specify if the outliers are to be removed and imputed.
`imp_methods`	The imputation methods to be used.

Value

The outliers found in the data. If the outliers are replaced, then the imputation errors are also returned.

duplicate_timestamps *Duplicate Timestamps*

Description

This function finds and removes the duplicate timestamps in the time columns of the data.

Usage

```
duplicate_timestamps(dt)
```

Arguments

`dt`	Input data

Value

A list of data.table without duplicate timestamps and the duplicate timestamps.

find_dif *Helper function to find the time difference between two given timestamps.*

Description

Helper function to find the time difference between two given timestamps.

Usage

```
find_dif(time1, time2)
```

Arguments

time1	POSIXt or Date object.
time2	POSIXt or Date object.

Value

String, specifying the time interval between time1 and time2. It contains a integer and the unit, for e.g., *5 weeks*, *6 months*, *14 hours*, etc.

gen.animation *Generate animation*

Description

This function takes the list outputted by animate_interval() and generates a GIF animation. It is a simple wrapper around the gganimate::animate() function with some defaults. The generated GIF can be saved using the anim_save() function. By default, in the animate() function only 50 states in the data are shown. So, to avoid this gen.animation() defines the default value for the number of frames. Also, the duration argument has a default value equal to the number of states, making the animation slower. More arguments can be passed, which are then passed to animate(), like, height, width, fps, renderer, etc.

Usage

```
gen.animation(anim, nframes = 2 * anim$nstates, duration = anim$nstate, ...)
```

Arguments

anim	List outputted by the animate_interval() function containing a gganim object and the number of states in the animation.
nframes	Number of frames. Defaults to double the number of states in the animation.
duration	The duration of animation. Defaults to the number of states in the animation.
...	Extra arguments passed to gganimate::animate().

Value

Does not return any value.

Examples

```
## Not run:
  a <- animate_interval(cts, "10 year")

  # Generate animation using `gen.animation()`
  if(interactive()){
    gen.animation(a, height = 700, width = 900)
  }

  # Save animation using `anim_save()`
  anim_save("filename.gif")

## End(Not run)
```

---

gen.report *Generate a report.*

Description

`gen.report()` generates a report of the entire process, the changes made to the original data and details about the impurities found in the data.

Usage

```
gen.report(obj)
```

Arguments

`obj`	A *cleanTS* object.

Value

Does not return any value.

Examples

```
## Not run:
  # Perform cleaning
  cts <- cleanTS(data, date_format = "my", time = "index", value = "value")

  gen.report(cts)

## End(Not run)
```

impute *Handle missing values in the data*

Description

This function handles missing values in the data. It compares various imputation methods and finds the best one for imputation.

Usage

```
impute(dt, methods)
```

Arguments

dt	A data.table.
methods	The imputation methods to be used.

Value

A data.table with missing data imputed, and the imputation errors.

interact_plot *Create interactive plot*

Description

Interactive plot is similar to the animated plot, but gives the used some control over the animation. It runs a shinyApp instead of creating a GIF.

Usage

```
interact_plot(obj, interval)
```

Arguments

obj	A *cleanTS* object.
interval	A numeric or character, specifying the viewing interval.

Details

The problem with an animated plot is that the user does not have any control over the animation. There is not play or pause functionality so that the user can observe any desired frame. This can be achieved by adding interactivity to the plot. The `interact_plot()` function creates and runs a shiny widget locally on the machine. It takes the cleanTS object and splits the cleaned data according to the `interval` argument, similar to the `animate interval()` function. It then creates a shiny widget which shows the plot for the current state and gives a slider used to change the state. Unlike `animate_interval()` it provides a global report containing information about complete data, and a state report giving information about the current state shown in the plot.

Value

Does not return any value.

Examples

```
## Not run:
   if(interactive()){
     # Using the same data used in `cleanTS()` function example.
     interact_plot(cts, interval = "1 week")
   }

## End(Not run)
```

`mergecsv` *Merge Multiple CSV files*

Description

`mergecsv()` takes a folder containing CSV files and merges them into a single *data.table*. It is assumed that the first column of all the CSVs contains the timestamps.

Usage

```
mergecsv(path, formats)
```

Arguments

`path`	Path to the folder.
`formats`	Datetime formats.

Details

All these files are read and the first column is parsed to a proper DateTime object using the formats given in the formats argu- ment. Then these dataframes are merged using the timestamp column as a common column. The merged data frame returned by the function contains the first column as the timestamps.

Value

Merged `data.table`.

missing_timestamps *Missing timestamps*

Description

This function finds and inserts the missing timestamps in the time columns of the data. The observations for the inserted timestamps are filled with NA.

Usage

```
missing_timestamps(dt)
```

Arguments

dt	Input data

Value

A list of data.table with inserted missing timestamps and the missing timestamps.

print.cleanTS *Print a* cleanTS *object*

Description

Print method for cleanTS class.

Usage

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

Arguments

x	cleanTS object
...	Other arguments

Value

Does not return any value.

Examples

```
## Not run:
# Using the same data as in `cleanTS()` function example.
cts <- cleanTS(data, "my")
print(cts)

## End(Not run)
```

Index