Package: butterfly (via r-universe)

April 4, 2025

Title Verification for Continually Updating Time Series Data

Version 1.1.2

Description Verification of continually updating time series data where we expect new values, but want to ensure previous data remains unchanged. Data previously recorded could change for a number of reasons, such as discovery of an error in model code, a change in methodology or instrument recalibration. Monitoring data sources for these changes is not always possible. Other unnoticed changes could include a jump in time or measurement frequency, due to instrument failure or software updates. Functionality is provided that can be used to check and flag changes to previous data to prevent changes going unnoticed, as well as unexpected jumps in time.

License MIT + file LICENSE

URL https://docs.ropensci.org/butterfly/, https://github.com/ropensci/butterfly/

BugReports https://github.com/ropensci/butterfly/issues

Depends R (>= 4.1.0)

Imports cli, dplyr, lifecycle, rlang, waldo

Suggests knitr, rmarkdown, testthat (>= 3.0.0)

VignetteBuilder knitr

Config/testthat/edition 3

Encoding UTF-8

LazyData true

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.2

Repository https://ropensci.r-universe.dev

RemoteUrl https://github.com/ropensci/butterfly

RemoteRef main

RemoteSha a5ceec3e6c726264154a4dfd4b1ab6f0e2010715

Contents

`butterflycount` *Butterfly count dummy data*

Description

A completely fictional dataset of monthly butterfly counts

Usage

```
butterflycount
```

Format

`butterflycount`**:**

A list with 5 dataframes (january, february, march, april, may) containing 3 columns, and 3 + n_month rows:

time The date on which the imaginary count took place, in yyyy-mm-dd format

count Number of fictional butterflies counted

species Butterfly species name, only appears in april ...

`butterflymess` *Butterfly count messy data*

Description

A version of butterflycount made messy using the messy package. This dataset is only used for testing purposes

Usage

```
butterflymess
```

Format

`butterflymess`**:**

A list with 5 dataframes (january, february, march, april, may) containing 3 columns, and 3 + n_month rows:

time The date on which the imaginary, and messy, count took place, in yyyy-mm-dd format

count Number of fictional butterflies counted

species Butterfly species name, only appears in april ...

`catch` *Catch: return dataframe containing only rows that have changed*

Description

This function matches two dataframe objects by their unique identifier (usually "time" or "datetime in a timeseries), and returns a new dataframe which contains only rows that have changed compared to previous data. It will not return any new rows.

Usage

```
catch(df_current, df_previous, datetime_variable, ...)
```

Arguments

`df_current`	data.frame, the newest/current version of dataset x.
`df_previous`	data.frame, the old version of dataset, for example x - t1.
`datetime_variable`	string, which variable to use as unique ID to join `df_current` and `df_previous`. Usually a "datetime" variable.
`...`	Other `waldo::compare()` arguments can be supplied here, such as `tolerance` or `max_diffs`. See `?waldo::compare()` for a full list.

Details

The underlying functionality is handled by `create_object_list()`.

Value

A dataframe which contains only rows of `df_current` that have changes from `df_previous`, but without new rows. Also returns a waldo object as in `loupe()`.

See Also

`loupe()`

`create_object_list()`

Examples

```
# Returning only matched rows which contain changes
df_caught <- butterfly::catch(
  butterflycount$march, # New or current dataset
  butterflycount$february, # Previous version you are comparing it to
  datetime_variable = "time" # Unique ID variable they have in common
)

df_caught
```

`create_object_list` *create_object_list: creates a list of objects used in all butterfly functions*

Description

This function creates a list of objects which is used by all of `loupe()`, `catch()` and `release()`.

Usage

```
create_object_list(df_current, df_previous, datetime_variable, ...)
```

Arguments

`df_current`	data.frame, the newest/current version of dataset x.
`df_previous`	data.frame, the old version of dataset, for example x - t1.
`datetime_variable`	string, which variable to use as unique ID to join `df_current` and `df_previous`. Usually a "datetime" variable.
`...`	Other `waldo::compare()` arguments can be supplied here, such as `tolerance` or `max_diffs`. See `?waldo::compare()` for a full list.

Details

This function matches two dataframe objects by their unique identifier (usually "time" or "datetime in a timeseries).

It informs the user of new (unmatched) rows which have appeared, and then returns a `waldo::compare()` call to give a detailed breakdown of changes.

The main assumption is that `df_current` and `df_previous` are a newer and older versions of the same data, and that the `datetime_variable` variable name always remains the same. Elsewhere new columns can of appear, and these will be returned in the report.

Value

A list containing boolean where TRUE indicates no changes to previous data and FALSE indicates unexpected changes, a dataframe of the current data without new rows and a dataframe of new rows only

Examples

```
butterfly_object_list <- butterfly::create_object_list(
  butterflycount$february, # New or current dataset
  butterflycount$january, # Previous version you are comparing to
  datetime_variable = "time" # Unique ID variable they have in common
)

butterfly_object_list

# You can pass other `waldo::compare()` options such as tolerance here
butterfly_object_list <- butterfly::create_object_list(
  butterflycount$march, # New or current dataset
  butterflycount$february, # Previous version you are comparing it to
  datetime_variable = "time", # Unique ID variable they have in common
  tolerance = 2
)

butterfly_object_list
```

---

forestprecipitation *Forest precipitation dummy data*

Description

A completely fictional dataset of daily precipitation

Usage

```
forestprecipitation
```

Format

`forestprecipitation`**:**

A list with 2 dataframes (january, february) containing 2 columns, and 6 rows. February intentionally resets to 1970-01-01

time The date on which the imaginary rainfall was measured took place, in yyyy-mm-dd format

rainfall_mm Rainfall in mm ...

loupe *Loupe: compare new and old data in continuously updated timeseries*

Description

A loupe is a simple, small magnification device used to examine small details more closely.

Usage

```
loupe(df_current, df_previous, datetime_variable, ...)
```

Arguments

df_current	data.frame, the newest/current version of dataset x.
df_previous	data.frame, the old version of dataset, for example x - t1.
datetime_variable	string, which variable to use as unique ID to join df_current and df_previous. Usually a "datetime" variable.
...	Other waldo::compare() arguments can be supplied here, such as tolerance or max_diffs. See ?waldo::compare() for a full list.

Details

This function is intended to aid in the verification of continually updating timeseries data where we expect new values but want to ensure previous values remains unchanged.

This function matches two dataframe objects by their unique identifier (usually "time" or "datetime in a timeseries).

It informs the user of new (unmatched) rows which have appeared, and then returns a waldo::compare() call to give a detailed breakdown of changes. If you are not familiar with waldo::compare(), this is an expanded and more verbose function similar to base R's all.equal().

loupe() will then return TRUE if there are not changes to previous data, or FALSE if there are unexpected changes. If you want to extract changes as a dataframe, use catch(), or if you want to drop them, use release().

The main assumption is that df_current and df_previous are a newer and older versions of the same data, and that the datetime_variable variable name always remains the same. Elsewhere new columns can of appear, and these will be returned in the report.

The underlying functionality is handled by create_object_list().

Value

A boolean where TRUE indicates no changes to previous data and FALSE indicates unexpected changes.

See Also

create_object_list()

Examples

```
# Checking two dataframes for changes
# Returning TRUE (no changes) or FALSE (changes)
# This example contains no differences with previous data
butterfly::loupe(
  butterflycount$february, # New or current dataset
  butterflycount$january, # Previous version you are comparing it to
  datetime_variable = "time" # Unique ID variable they have in common
)

# This example does contain differences with previous data
butterfly::loupe(
  butterflycount$march,
  butterflycount$february,
  datetime_variable = "time"
)
```

`release` *Release: return current dataframe without changed old rows*

Description

This function matches two dataframe objects by their unique identifier (usually "time" or "datetime in a timeseries), and returns a new dataframe which contains the new rows (if present) but matched rows which contain changes from previous data will be dropped.

Usage

```
release(df_current, df_previous, datetime_variable, include_new = TRUE, ...)
```

Arguments

`df_current`	data.frame, the newest/current version of dataset x.
`df_previous`	data.frame, the old version of dataset, for example x - t1.
`datetime_variable`	string, which variable to use as unique ID to join `df_current` and `df_previous`. Usually a "datetime" variable.
`include_new`	boolean, should new rows be included? Default is TRUE.
`...`	Other `waldo::compare()` arguments can be supplied here, such as `tolerance` or `max_diffs`. See `?waldo::compare()` for a full list.

Value

A dataframe which contains only rows of `df_current` that have not changed from `df_previous`, and includes new rows. Also returns a waldo object as in `loupe()`.

See Also

loupe()

create_object_list()

Examples

```
# Dropping matched rows which contain changes, and returning unchanged rows
df_released <- butterfly::release(
  butterflycount$march, # New or current dataset
  butterflycount$february, # Previous version you are comparing it to
  datetime_variable = "time", # Unique ID variable they have in common
  include_new = TRUE # Whether to include new rows or not, default is TRUE
)

df_released
```

`timeline` *timeline: check if a timeseries is continuous*

Description

Check if a timeseries is continuous. Even if a timeseries does not contain obvious gaps, this does not automatically mean it is also continuous.

Usage

```
timeline(df_current, datetime_variable, expected_lag = 1)
```

Arguments

`df_current`	data.frame, the newest/current version of dataset x.
`datetime_variable`	string, the "datetime" variable that should be checked for continuity.
`expected_lag`	numeric, the acceptable difference between timestep for a timeseries to be classed as continuous. Any difference greater than `expected_lag` will indicate a timeseries is not continuous. Default is 1. The smallest units of measurement present in the column will be used. In a column formatted YYYY-MM-DD day will be used, therefore 1 would be 1 day, 7 would be a week.

Details

Measuring instruments can have different behaviours when they fail. For example, during power failure an internal clock could reset to "1970-01-01", or the manufacturing date (say, "2021-01-01"). This leads to unpredictable ways of checking if a dataset is continuous.

The `timeline_group()` and `timeline()` functions attempt to give the user control over how to check for continuity by providing an `expected_lag`. The difference between timesteps in a dataset should not exceed the `expected_lag`.

Note: for monthly data it is recommended you convert your Date column to a monthly format (e.g 2024-October, 10-2024, Oct-2024 etc.), so a constant expected lag can be set (not a range of 29 - 31 days).

Value

A boolean, TRUE if the timeseries is continuous, and FALSE if there are more than one continuous timeseries within the dataset.

See Also

`timeline_group()`

Examples

```
# A nice continuous dataset should return TRUE
butterfly::timeline(
  forestprecipitation$january,
  datetime_variable = "time",
  expected_lag = 1
)

# In February, our imaginary rain gauge's onboard computer had a failure.
# The timestamp was reset to 1970-01-01
butterfly::timeline(
  forestprecipitation$february,
  datetime_variable = "time",
  expected_lag = 1
)
```

`timeline_group` *timeline_group: check if a timeseries is continuous*

Description

If after using `timeline()` you have established a timeseries is not continuous, or if you are working with data where you expect distinct sequences or events, you can use `timeline_group()` to extract and classify different distinct continuous chunks of your data.

Usage

```
timeline_group(df_current, datetime_variable, expected_lag = 1)
```

Arguments

`df_current`	data.frame, the newest/current version of dataset x.
`datetime_variable`	string, the "datetime" variable that should be checked for continuity.
`expected_lag`	numeric, the acceptable difference between timestep for a timeseries to be classed as continuous. Any difference greater than `expected_lag` will indicate a time-series is not continuous. Default is 1. The smallest units of measurement present in the column will be used. In a column formatted YYYY-MM-DD day will be used, therefore 1 would be 1 day, 7 would be a week.

Details

We attempt to do this without sorting, or changing the data for a couple of reasons:

1. There are no difference in dates: Some instruments might record dates that appear identical, but are still in chronological order. For example, high-frequency data in fractional seconds. This is a rare use case though.
2. Dates are generally ascending/descending, but the instrument has returned to origin. Probably more common, and will results in a non-continuous dataset, however the records are still in chronological order This is something we would like to discover. This is accounted for in the logic in case_when().

Note: for monthly data it is recommended you convert your Date column to a monthly format (e.g 2024-October, 10-2024, Oct-2024 etc.), so a constant expected lag can be set (not a range of 29 - 31 days).

Value

A data.frame, identical to `df_current`, but with extra columns `timeline_group`, which assigns a number to each continuous sets of data and `timelag` which specifies the time lags between rows.

Examples

```
# A nice continuous dataset should return TRUE
# In February, our imaginary rain gauge's onboard computer had a failure.
# The timestamp was reset to 1970-01-01

# We want to group these different distinct continuous sequences:
butterfly::timeline_group(
  forestprecipitation$february,
  datetime_variable = "time",
  expected_lag = 1
)
```

Index