Package: multimeter (via r-universe)

September 12, 2024

Title Inspect Data Pipelines

Version 0.0.0.9000

Description Inspect data pipelines (especially magrittr pipelines) for before-after changes during tricky operations.

License GPL (>= 3) + file LICENSE

Suggests covr, devtools, pkgdown, rmarkdown, styler, testthat (>= 3.0.0), usethis

Config/testthat/edition 3

Encoding UTF-8

LazyData true

Roxygen list(markdown = TRUE)

RoxygenNote 7.1.1

Imports R6, dplyr, magrittr, purrr, tibble, vctrs

URL https://github.com/bvancil/multimeter

BugReports https://github.com/bvancil/multimeter/issues

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

RemoteUrl https://github.com/bvancil/multimeter

RemoteRef HEAD

RemoteSha 04ca47abd11cfde8765185ad88f06963ca369b75

Contents

Counter 2
History 3
mm_get_identity_meter 5
mm_get_missing_meter 5
mm_get_names_meter 6
mm_get_value_meter 7
Multimeter 7

Index **10**

`Counter` *Counter R6 class to represent a counter*

Description

Counter R6 class to represent a counter

Counter R6 class to represent a counter

Public fields

`count` number of times that something has happened

Methods

Public methods:

- `Counter$new()`
- `Counter$print()`
- `Counter$increment()`
- `Counter$chain_increment()`
- `Counter$pipe_increment()`
- `Counter$reset()`
- `Counter$clone()`

Method `new()`**:** Create a new Counter object.

Usage:

```
Counter$new()
```

Returns: A new `Counter` object.

Method `print()`**:** Print a Counter object.

Usage:

```
Counter$print(...)
```

Arguments:

`...` required for compatibility with print()

Returns: `self`

Method `increment()`**:** Add one to counter.

Usage:

```
Counter$increment()
```

Returns: `NULL`

Method `chain_increment()`**:** Add one to counter.

Usage:

```
Counter$chain_increment()
```

Returns: `self`

Method `pipe_increment()`**:** Add one to counter.

Usage:

```
Counter$pipe_increment(.data)
```

Arguments:

`.data` piped dataset

Returns: `.data`

Method `reset()`**:** Reset counter to zero.

Usage:

```
Counter$reset()
```

Returns: `self`

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
Counter$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`History` *History R6 class to represent historical values*

Description

History R6 class to represent historical values

History R6 class to represent historical values

Public fields

`values` list of stored values

Methods

Public methods:

- `History$new()`
- `History$print()`
- `History$log()`
- `History$chain_log()`
- `History$pipe_log()`
- `History$reset()`
- `History$clone()`

Method `new()`**:** Create a new History object.

Usage:

```
History$new()
```

Returns: A new `History` object.

Method `print()`**:** Print summary of `Counter` object

Usage:

```
History$print(...)
```

Arguments:

`...` unused

Method `log()`**:** Store a value

Usage:

```
History$log(val)
```

Arguments:

`val` value to add to history

Returns: `NULL`

Method `chain_log()`**:** Store a value

Usage:

```
History$chain_log(val)
```

Arguments:

`val` value to add to history

Returns: `self`

Method `pipe_log()`**:** Store a value

Usage:

```
History$pipe_log(val)
```

Arguments:

`val` value to add to history

Returns: `val`

Method `reset()`**:** Clear history

Usage:

```
History$reset(x)
```

Arguments:

`x` pipeline value passed in (Optional)

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
History$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

mm_get_identity_meter *Simple meter that does not transform its input and performs no comparison of results other than returning them.*

Description

Simple meter that does not transform its input and performs no comparison of results other than returning them.

Usage

```
mm_get_identity_meter()
```

Value

`Multimeter` instance

Examples

```
if (interactive()) {
  m <- mm_get_identity_meter()
  m$probe(1L) == 1L
  m$probe(2L) == 2L
  m$comparison
  print(m)
}
```

mm_get_missing_meter *Compare missingness fractions in each column of a data frame and print any increases in missingness.*

Description

This is useful after `mutate` operations but may give false positives after a `summarize`.

Usage

```
mm_get_missing_meter()
```

Value

a new Multimeter tuned to look for missing values

Examples

```
if (interactive()) {
  # Function to randomly create NAs in a variable
  maybe_na <- function(x, prob = 0.2) {
    n <- base::length(x)
    missing_of_type_x <- vctrs::vec_cast(NA, to = vctrs::vec_ptype(x))
    dplyr::if_else(stats::runif(n) < prob, missing_of_type_x, x)
  }
  # In pieces, to see what's going on
  missing_meter <- mm_get_missingness_meter()
  library(dplyr)
  missing_meter$probe(starwars)
  set.seed(2021L + 06L + 28L)
  starwars2 <- starwars %>%
    mutate(homeworld = maybe_na(homeworld))
  missing_meter$probe(starwars2)
  print(missing_meter$comparison)

  # All together in one pipeline
  starwars2 <- starwars %>%
    missing_meter$reset() %>%
    missing_meter$probe() %>%
    mutate(homeworld = maybe_na(homeworld)) %>%
    missing_meter$probe()
  print(missing_meter$comparison)
}
```

`mm_get_names_meter` *Track column names before and after a transformation*

Description

Track column names before and after a transformation

Usage

```
mm_get_names_meter()
```

Value

`Multimeter` instance

Examples

```
if (interactive()) {
  m <- mm_get_names_meter()
  library(dplyr)
  mtcars2 <- mtcars %>%
    m$probe() %>%
```

```
    mutate(square_error_from_five = (cyl - 5)^2) %>%
    m$probe()
  print(m$comparison)
}
```

`mm_get_value_meter` *Track changes in columns provided to $probe() function*

Description

Track changes in columns provided to $probe() function

Usage

```
mm_get_value_meter()
```

Value

`Multimeter` instance

Examples

```
if (interactive()) {
  m <- mm_get_state_capture_meter()
  library(dplyr)
  mtcars2 <- mtcars %>%
    m$probe(cyl) %>%
    mutate(square_error_from_five = (cyl - 5)^2) %>%
    m$probe(square_error_from_five)
  print(m$comparison)
}
```

`Multimeter` *Multimeter R6 class to represent a multimeter type*

Description

Multimeter R6 class to represent a multimeter type

Multimeter R6 class to represent a multimeter type

Details

A new Multimeter can insert probes in various stages of a data pipeline, allowing you to compare output before and after a change. To facilitate use with large datasets for which you might only wish to store a summary of the data, a Multimeter object can take a `transform` function that transforms the data before storing it, and a `compare` function to point out salient differences.

Public fields

`transform` function of dataset

`compare` function of before and after transformed datasets

`before` transformed "before" dataset

`after` transformed "after" dataset

`comparison` result of `compare` on `before` and `after`

Methods

Public methods:

- `Multimeter$new()`
- `Multimeter$print()`
- `Multimeter$probe()`
- `Multimeter$save_memory()`
- `Multimeter$reset()`
- `Multimeter$clone()`

Method `new()`**:** Create a new Multimeter object.

Usage:

```
Multimeter$new(transformer, comparator)
```

Arguments:

`transformer` function of dataset to store

`comparator` function of transformed dataset to compare

Returns: A new `Multimeter` object.

Method `print()`**:** Print summary of `Multimeter` object

Usage:

```
Multimeter$print(...)
```

Arguments:

`...` unused

Method `probe()`**:** Insert a probe at this stage of data pipeline.
Because this operation returns the first argument, it may be used in magrittr or base R pipelines.

Usage:

```
Multimeter$probe(.data, ..., .print_comparison = TRUE)
```

Arguments:

`.data` dataset tibble or data.frame

`...` other arguments passed on to `self$transform`

`.print_comparison` logical whether to print a comparison on the second `probe`. Ignored otherwise. Default: `TRUE`

Returns: `.data`

Method `save_memory()`**:** Delete intermediate stored transformed before/after datasets to save memory.

Usage:

```
Multimeter$save_memory(.data)
```

Arguments:

`.data` dataset Optional if used in a data pipeline

Returns: `.data` (or `NULL` if `.data` is missing)

Method `reset()`**:** Reset multimeter for repeated use

Usage:

```
Multimeter$reset(.data)
```

Arguments:

`.data` dataset Optional if used in a data pipeline

Returns: `.data` (or `NULL` if `.data` is missing)

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
Multimeter$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Index

Counter, 2

History, 3

mm_get_identity_meter, 5
mm_get_missing_meter, 5
mm_get_names_meter, 6
mm_get_value_meter, 7
Multimeter, 7