Package ‘assertable’

October 12, 2022

Type Package

Title Verbose Assertions for Tabular Data (Data.frames and Data.tables)

Version 0.2.8

Maintainer Grant Nguyen <`grant.nguyen@gmail.com`>

Description Simple, flexible, assertions on data.frame or data.table objects with verbose output for vetting. While other assertion packages apply towards more general use-cases, assertable is tailored towards tabular data. It includes functions to check variable names and values, whether the dataset contains all combinations of a given set of unique identifiers, and whether it is a certain length. In addition, assertable includes utility functions to check the existence of target files and to efficiently import multiple tabular data files into one data.table.

Depends R (>= 3.1.0)

Imports data.table, parallel

License GPL-3

Encoding UTF-8

LazyData true

RoxygenNote 6.1.1

Suggests knitr, rmarkdown

VignetteBuilder knitr

NeedsCompilation no

Author Grant Nguyen [aut, cre],
Max Czapanskiy [ctb]

Repository CRAN

Date/Publication 2021-01-27 06:30:15 UTC

R topics documented:

`assert_colnames` *Assert that a data.frame contains specified column names*

Description

Given a data.frame or data.table object, assert that all columns in the colnames argument exist as columns.

Usage

```
assert_colnames(data, colnames, only_colnames = TRUE, quiet = FALSE)
```

Arguments

`data`	A data.frame or data.table
`colnames`	Character vector with column names corresponding to columns in *data*
`only_colnames`	Assert that the only columns in the data object should be those in *colnames*. Default = T.
`quiet`	Do you want to suppress the printed message when a test is passed? Default = F.

Value

Throws error if test is violated.

Examples

```
assert_colnames(CO2, c("Plant","Type","Treatment","conc","uptake"))
assert_colnames(CO2, c("Plant","Type"), only_colnames=FALSE)
```

`assert_coltypes` *Assert that a data.frame's columns are certain types*

Description

Given a data.frame or data.table object, assert that all columns in the names of the coltypes argument match the types of the elements of the coltypes argument.

Usage

```
assert_coltypes(data, coltypes, quiet = FALSE)
```

Arguments

`data`	A data.frame or data.table
`coltypes`	List with names corresponding to columns in *data*. The types of the columns in *data* will be tested against types of the elements in coltypes.
`quiet`	Do you want to suppress the printed message when a test is passed? Default = F.

Value

Throws error if test is violated.

Examples

```
# Should pass
assert_coltypes(CO2, list(Plant = integer(), conc = double()))
# Should fail
## Not run:
  assert_coltypes(CO2, list(Plant = character(), conc = character()))

## End(Not run)
```

`assert_ids` *Assert that a data.frame contains all unique combinations of specified ID variables, and doesn't contain duplicates within combinations*

Description

Given a data.frame or data.table object and a named list of id_vars, assert that all possible combinations of id_vars exist in the dataset, that no combinations of id_vars exist in the dataset but not in id_vars, and that there are no duplicate values within the dataset within unique combinations of id_vars.

If ids_only = T and assert_dups = T, returns all combinations of id_vars along with the *n_duplicates*: the count of duplicates within each combination. If ids_only = F, returns all duplicate observations from the original dataset along with *n_duplicates* and *duplicate_id*: a unique ID for each duplicate value within each combination of id_vars.

Usage

```
assert_ids(data, id_vars, assert_combos = TRUE, assert_dups = TRUE,
  ids_only = TRUE, warn_only = FALSE, quiet = FALSE)
```

Arguments

`data`	A data.frame or data.table
`id_vars`	A named list of vectors, where the name of each vector must correspond to a column in *data*
`assert_combos`	Assert that the data object must contain all combinations of *id_vars*. Default = T.
`assert_dups`	Assert that the data object must not contain duplicate values within any combinations of *id_vars*. Default = T.
`ids_only`	By default, with assert_dups = T, the function returns the unique combinations of id_vars that have duplicate observations. If ids_only = F, will return every observation in the original dataset that are duplicates.
`warn_only`	Do you want to warn, rather than error? Will return all offending rows from the first violation of the assertion. Default=F.
`quiet`	Do you want to suppress the printed message when a test is passed? Default = F.

Details

Note: if assert_combos = T and is violated, then assert_ids will stop execution and return results for assert_combos before evaluating the assert_dups segment of the code. If you want to make sure both options are evaluated even in case of a violation in assert_combos, call assert_ids twice (once with assert_dups = F, then assert_combos = F) with warn_only = T, and then conditionally stop your code if either call returns results.

Value

Throws error if test is violated. Will print the offending rows. If warn_only=T, will return all offending rows and only warn.

Examples

```
plants <- as.character(unique(CO2$Plant))
concs <- unique(CO2$conc)
ids <- list(Plant=plants,conc=concs)
assert_ids(CO2, ids)
```

`assert_nrows` *Assert that a data.frame contains a specified number of rows*

Description

Given a data.frame or data.table object and a target number of rows, check that a dataset has that many rows

Usage

```
assert_nrows(data, target_nrows, quiet = FALSE)
```

Arguments

`data`	A data.frame or data.table
`target_nrows`	Numeric – number of expected rows
`quiet`	Do you want to suppress the printed message when a test is passed? Default = F.

Value

Throws error if test is violated

Examples

```
assert_nrows(CO2,84)
```

`assert_values` *Assert that a data.frame's columns are non-NA/infinite, or are greater, less than, equal/not-equal, or contain specified values.*

Description

Given a data.frame or data.table object, make assertions about values of the columns within the object. Assert that a column contains no missing/infinite values, or that it is greater/less than, equal to, or contains either a single value, vector with nrow(data) values, or a vector of any length(for *in* option).

Usage

```
assert_values(data, colnames, test = "not_na", test_val = NA,
  display_rows = TRUE, na.rm = FALSE, warn_only = FALSE,
  quiet = FALSE)
```

Arguments

`data`	A data.frame or data.table
`colnames`	Character vector with column names corresponding to columns in *data*
`test`	The type of evaluation you want to assert in your data • *not_na*: All values must not be Na • *not_nan*: All values must not be NaN • *not_inf*: All values must not be infinite • *lt*: All values must be less than test_val • *lte*: All values must be less than or equal to test_val • *gt*: All values must be greater than test_val • *gte*: All values must be greater than or equal to test_val • *equal*: All values must be equal to test_val • *not_equal*: All values must not equal test_val • *in*: All values must be one of the values in test_val
`test_val`	A single value, a vector with length = nrow(data), or a vector of any length (if using the *in* option for test. Must match the character type of colnames.
`display_rows`	Do you want to show the actual rows that violate the assertion? Default=T
`na.rm`	Do you want to remove NA and NaN values from assertions? Default=F
`warn_only`	Do you want to warn, rather than error? Will return all offending rows from the first violation of the assertion Default=F
`quiet`	Do you want to suppress the printed messages when a test is passed? Default = F.

Value

Throws error if test is violated. If warn_only=T, will return all offending rows from the first violation of the assertion.

Examples

```
assert_values(CO2, colnames="uptake", test="gt", 0) # Are all values greater than 0?
assert_values(CO2, colnames="conc", test="lte", 1000) # Are all values less than/equal to 1000?
## Not run:
 assert_values(CO2, colnames="uptake", test="lt", 40) # Are all values less than 40?
 # Fails: not all values < 40.

## End(Not run)
assert_values(CO2, colnames="Treatment", test="in", test_val = c("nonchilled","chilled"))
CO2_mult <- CO2
CO2_mult$new_uptake <- CO2_mult$uptake * 2
```

```
assert_values(CO2, colnames="uptake", test="equal", CO2_mult$new_uptake/2)
## Not run:
 assert_values(CO2, colnames="uptake", test="gt", CO2_mult$new_uptake/2, display_rows=F)
 # Fails: uptake !> new_uptake/2

## End(Not run)
```

`check_files` *Check for the existence of a vector of files, optionally repeated for a set amount of time.*

Description

Given a character vector of filenames, check how many of them currently exist. Optionally, can keep checking for a specified amount of time, at a given frequency

Usage

```
check_files(filenames, folder = "", warn_only = FALSE,
  continual = FALSE, sleep_time = 30, sleep_end = (60 * 3),
  display_pct = 75)
```

Arguments

`filenames`	A character vector of filenames (specify full paths if you are checking files that are not in present working directory)
`folder`	An optional character containing the folder name that contains the files you want to check (if used, do not include folderpath in the filenames characters). If not specified, will search in present working directory.
`warn_only`	Boolean (T/F), whether to end with a warning message as opposed to an error message if files are still missing at the end of the checks.
`continual`	Boolean (T/F), whether to only run once or to continually keep checking for files for *sleep_end* minutes. Default = F.
`sleep_time`	numeric (seconds); if *continual* = T, specify the number of seconds to wait in-between file checks. Default = 30 seconds.
`sleep_end`	numeric (minutes); if *continual* = T, specify number of minutes to check at *sleep_time* intervals before terminating. Default = 180 minutes.
`display_pct`	numeric (0-100); at what percentage of files found do you want to print the full list of still-missing files? Default = 75 percent of files.

Value

Prints the number of files that match. If *warn_only* = T, returns a character vector of missing files

Examples

```
## Not run:
 for(i in 1:3) {
   data <- CO2
   data$id_var <- i
   write.csv(data,file=paste0("file_",i,".csv"),row.names=FALSE)
 }
 filenames <- paste0("file_",c(1:3),".csv")
 check_files(filenames)

## End(Not run)
```

`import_files` *Given a vector of filenames, append all files and return as one data.table using a user-defined function*

Description

Given a character vector of filenames, check how many of them currently exist. Optionally, can keep checking for a specified amount of time, at a given frequency

Usage

```
import_files(filenames, folder = "", FUN = fread, warn_only = FALSE,
  multicore = FALSE, use.names = TRUE, fill = TRUE,
  mc.preschedule = FALSE, mc.cores = getOption("mc.cores", 2L), ...)
```

Arguments

`filenames`	A character vector of filenames (specify full paths if you are checking files that are not in present working directory)
`folder`	An optional character containing the folder name that contains the files you want to check (if used, do not include folderpath in the filenames characters). If not specified, will look in present working directory.
`FUN`	function: The function that you want to use to import your data, e.g. read.csv, fread, read_dta, etc.
`warn_only`	Boolean (T/F), whether to send a warning message as opposed to an error message if files are missing prior to import. Will only import the files that do exist.
`multicore`	boolean, use lapply or mclapply (multicore = T) to loop over files in *filenames* for import. Default=F.
`use.names`	boolean, pass to the use.names option for *rbindlist*
`fill`	boolean, pass to the fill option for *rbindlist*
`mc.preschedule`	boolean, pass to the mc.preschedule option for *mclapply* if multicore = T. Default = F.
`mc.cores,`	pass to the mc.preschedule option for *mclapply* if multicore = T. Default = mclapply default.
`...`	named arguments of *FUN* to pass to *FUN*

Value

One data.table that contains all files in *filenames*, combined together using rbindlist. Returns an error if any file in *filenames* does not exist

Examples

```
## Not run:
 for(i in 1:3) {
   data <- CO2
   data$id_var <- i
   write.csv(data,file=paste0("file_",i,".csv"),row.names=FALSE)
 }
 filenames <- paste0("file_",c(1:3),".csv")
 import_files(filenames, FUN=fread)
 import_files(filenames, FUN=read.csv, stringsAsFactors=FALSE)
 import_files(filenames, FUN=fread, multicore=T, mc.cores=1) # Only if you have a multi-core system

## End(Not run)
```

Index