Introduction to tester

Gaston Sanchez
www.gastonsanchez.com/tester

1 Introduction and Motivation

tester provides human readable functions to test characteristics of some common R objects. The main purpose behind tester is to help you validate objects, especially for programming and developing purposes (e.g. creating R packages)

Testing objects When we write a function, more often than not, we need to validate its arguments. In order to do so, we can use some of the already available functions in R that allow us to test whether objects have certain features. For instance, we can use `is.matrix(M)` to test if `M` is a matrix. Likewise, if you want to test if an object is a list, we can use the `is.list()` function.

The interesting part comes when we want to test for more specific characteristics, like testing if `M` is a numeric matrix, or test if a number is a positive integer, or maybe if it is a decimal number. Let's take the case in which we want to test whether an object is a character matrix. One way to do that would be to write something like this:

```
# test if object is a character matrix
object = matrix(letters[1:6], 2, 3)

if (is.matrix(object) & is.character(object)) TRUE else FALSE

## [1] TRUE
```

Now let's say we want to test if a given number is a positive integer:

```
# test if number is a positive integer
number = 1

if (number > 0 & is.integer(number)) TRUE else FALSE

## [1] FALSE
```

In this case, we know that `number = 1` but the test returned `FALSE`. The reason is that the number 1 is not an strict integer in R. Instead, we need to declare `number = 1L`. Now, if we test again we will get `TRUE`:

```
# test if number is a positive integer
number = 1L

if (number > 0 & is.integer(number)) TRUE else FALSE

## [1] TRUE
```

Easier tests If we just have a couple of functions, testing its arguments may not be a big deal. But when we have dozens or hundreds of functions, even if they are not in the form of a package, testing their arguments can be more complicated. Instead of writing expressions like the following one:

```
if (number > 0 & is.integer(number)) TRUE else FALSE
```

it would also be desirable to simply write something like this:

```
is_positive_integer(number)
```

This is precisely what `tester` allows us to do by providing a set of functions to test objects in a friendly way, following the so-called *literate programming* paradigm. Under this paradigm, instead of writing programs instructing the computer what to do, we write programs explaining humans what we want the computer to do. The advantage is that *when we read code, we should be able to do so as if we were reading a text.* In this sense, the goal of `tester` is twofold: 1) help you test objects, and 2) help you write more human readable code.

Here is another example. Suppose we want to check if a vector has missing values. One option to answer that quesiton is to use the function `is.na()`:

```
# test for missing values
is.na(c(1, 2, 3, 4, NA))

## [1] FALSE FALSE FALSE FALSE  TRUE
```

Depending on your goals, `is.na()` might be enough. But what if we just want to simply test if a vector has missing values? With `tester` now we can with do that using the function `has_missing()`:

```
# test for missing values
has_missing(c(1, 2, 3, 4, NA))

## [1] TRUE


# or equivalently
has_NA(c(1, 2, 3, 4, NA))

## [1] TRUE
```

2 About `tester`

To use `tester` (once you have installed it), load it with the function `library()`:

```
# load package tester
library(tester)
```

2.1 Numbers

To test if we have number, as well as different types of numbers, we can use one of the following functions:

Testing Numbers

Function	Description
`is_positive()`	tests if a number is positive
`is_negative()`	tests if a number is negative
`is_integer()`	tests if a number is an integer
`is_natural()`	tests if a number is a natural number
`is_odd()`	tests if a number is an odd number
`is_even()`	tests if a number is an even number
`is_positive_integer()`	tests if a number is a positive integer
`is_negative_integer()`	tests if a number is a negative integer
`is_decimal()`	tests if a number is decimal
`is_positive_decimal()`	tests if a number is a positive decimal
`is_negative_decimal()`	tests if a number is a negative decimal

2.2 Logical

To test if an object (or a condition) is `TRUE` or `FALSE`, we can use the following functions:

Testing Logicals

Function	Description
`is_TRUE()`	tests if an object is `TRUE`
`is_FALSE()`	tests if an object is `FALSE`
`true_or_false()`	tests if is `TRUE` or `FALSE`

2.3 Vectors

To test if we have different types of vectors we can use the following functions:

Testing Vectors

Function	Description
`is_vector()`	tests if an object is a vector
`is_numeric_vector()`	tests if an object is a numeric vector
`is_string_vector()`	tests if an object is a string vector
`is_logical_vector()`	tests if an object is a logical vector
`is_not_vector()`	tests if an object is not a vector

2.4 Matrices

Likewise, to test if we have different types of matrices we can use the following functions:

Testing Matrices

Function	Description
`is_matrix()`	tests if an object is a matrix
`is_numeric_matrix()`	tests if an object is a numeric matrix
`is_string_matrix()`	tests if an object is a string matrix
`is_logical_matrix()`	tests if an object is a logical matrix
`is_square_matrix()`	tests if an object is a string matrix
`is_diagonal()`	tests if an object is a diagonal matrix
`is_triangular()`	tests if an object is a triangular matrix
`is_lower_triangular()`	tests if a matrix is lower triangular
`is_upper_triangular()`	tests if a matrix is upper triangular
`is_not_matrix()`	tests if an object is not a matrix

2.5 Data Frame

To test if we have different types of data frames we can use the following functions:

Testing Data Frames

Function	Description
`is_dataframe()`	tests if an object is a data frame
`is_numeric_dataframe()`	tests if an object is a numeric data frame
`is_string_dataframe()`	tests if an object is a string data frame
`is_not_dataframe()`	tests if an object is not a data frame

2.6 Missing Values

For testing missing values, infinite values, not numbers, `tester` provides the following functions:

Testing Missing Values

Function	Description
`has_missing()`	tests if an object has missing values
`has_infinite()`	tests if an object has infinite values
`has_not_a_number()`	tests if an object has 'Not a Number'
`has_nas()`	tests if an object has `NA, Inf, -Inf, NaN`

2.7 Other

`tester` comes with many more functions that will allow you to check —in a friendly way— whether some common R objects have certain characteristics. Some of the extra available functions are:

Testing Missing Values

Function	Description
`has_dimension()`	tests if an object has dimension
`is_tabular()`	tests if an object is a matrix or data frame
`is_multiple()`	tests if a number is multiple of a given number
`list_of_vectors()`	tests if an object is a list of vectors
`list_of_numeric_vectors()`	tests if an object is a list of numeric vectors
`list_of_string_vectors()`	tests if an object is a list of string vectors
`list_of_logical_vectors()`	tests if an object is a list of logical vectors