How to Write Unit Tests

Unit tests generally fit into the following categories:

Type of test	Minimum per function	Include	Other considerations
Normal operation test	1	Should check the function fulfils its intended purpose as laid out in the roxygen header.	Consider splitting the function up if there is more than one aspect to check.
Error handling tests	One per input	Check that the function is able to cope with any potential errors introduced by previous functions, either by fixing values or throwing an error message.	

For more detail on each please and **templates** see the below sections.

Step-by-step guide

Most unit tests would be built using the following steps:

1. Create dummy input datasets.

a. Get the actual inputs to the function and take a subset of them to use as the inputs to your test.

```
input_df1_downsized<- df1[c(1:3),c(relevant_columns)]
input_df2_downsized<- df2[c(1:3),c(relevant columns)]
```

b. Apply the `dput` function to your downsized datasets. This will output (in the console) code that can be used to produce your dummy dataset as required. For example, `dput(input_df1_downsized)` would produce:

```
structure(
  list(
    `as.Date(date)` = structure(c(12537, 12544, 12551), class = "Date"),
    Clear_Low = c(29, 28, 28),
    Clear_High = c(35, 36, 35)
  ),
  row.names = c(NA, 3L),
  class = "data.frame"
)
```

If the test is checking for error handling you may want to introduce some errors to this, for instance incorrect negative values or NAs.

2. Next, work out what the output of your function should be for your dummy data.

Either manually calculate or use `dput` to create an output object for the correct operation of your function.

If you are testing error catching there may not be a correct output, an error message may be expected instead.

3. Use testthat package to check operation of function.

The `testthat` package and `test_that` function allow us to perform various checks with the dummy inputs and outputs we have created already.

For instance `expect_equal` allows us to check the function produces the outputs we expect, and `expect_error` allows us to check the correct error messages are thrown, both shown below.

```
testthat::test_that("function_performs_calcualtions", {
  testthat::expect_equal(function_to_test(input_df1_downsized, input_df2_downsized),
                         output_df)
  testthat::expect_error(function_to_test(input_df1_with_errors))
})
```

Other possible `expect_` functions can be found at the testthat reference pages.

Normal Operation Tests

Unit tests of the normal operation of a function should test the code's core functionality. They should cover all cases the function applies to, and ideally test specific calculation methods.

At least one normal operation unit test should be needed for each function.

A **template** is provided below, you should add detail to the description as well as changing object names:

```
#' @title Unit test 1
#' @description Tests the normal operation of function_name [ADD DETAIL]

testthat::test_that("function name normal operation test", {
  #Create dummy data

  function_name_ut1_input_data_1 <- #Paste dput of first three or four rows and relevant columns of dat

  function_name_ut1_input_data_2 <- #Same again

  function_name_ut1_output_data <- #Paste dput of expected outputs or write code to manually calculate

  #Test
  testthat::expect_equal(
    function_name(
      function_name_ut1_input_data_1,
      function_name_ut1_input_data_2
    ),
    function_name_ut1_output_data,
    tolerance = 0.001
  )
})
```

Error catching tests

These can come in two forms:

- Checking that a function fixes expected errors,
- Checking that a function throws error messages for errors that cannot be fixed.

Code should be able to catch errors in each input and so we should have a unit test for each.

The first type would use a template very similar to that used for normal operation tests, but with the input datasets tweaked to include any errors that should be fixed.

The latter type are slightly different, we have created a library of these for ease of use and consistency, the following are available:

function_name(params)	**Error message**	**Notes**
`validate_input_type(param, type)`	`param` must be of type `type`.	`type` may be "data.frame", "numeric", "character", "logical" or "date".
`validate_input_choices(param, options)`	`param` must be of a valid option.	Additional message which states the possible options to choose from. `options` should be a vector c(option1,option2…)
`validate_country_inputs(param, module)`	`param` must be a valid devolved administration.	Additional message which lists and suggestions acceptable forms for country input.
`validate_column_values_NA(input_table, warn)`	**`input_table`** contains NA values.	Additional message reporting columns which have NA. use `warn = TRUE` to throw a warning instead of error.
`validate_column_values_negative(input_table, zeros_allowed, warn)`	`input_table` contains invalid values.	Additional message reporting numeric columns which have invalid values. Use `zeros_allowed = TRUE` to treat 0 as valid.
`validate_column_types(input_table, column_types, warn)`	`input_table` contains columns with mismatched types.	Additional message reporting columns which don't match expected type.
`validate_column_values_against_metadata(input_table, warn)`	`input_table` contains discrepancies with meta data.	Additional message highlighting columns and what discrepancies there are

When a custom error catching check is needed, the **template** below can be used, you should add detail to the description as well as changing object names:

```r
#' @title Unit test 3/4/5...
#' @description Test of error catching for error_type in input_dataframe_name
#' for function_name

testthat::test_that("Error catching test for error_type", {
  #Create dummy data

  function_name_ut3_input_data_1 <- #Adjusted dput with one error type in it

  function_name_ut3_input_data_2 <- #Or maybe the error is in this one

  #Unit test for expected error type
  testthat::expect_error(
    function_name(
      function_name_ut3_input_data_1,
      function_name_ut3_input_data_2
    ),
    "Stop error text goes here"
  )
})
```