

# Package ‘testdat’

January 4, 2022

**Type** Package

**Title** Data Unit Testing for R

**Version** 0.4.0

**Description** Test your data! An extension of the 'testthat' unit testing framework with a family of functions and reporting tools for checking and validating data frames.

**License** MIT + file LICENSE

**URL** <https://socialresearchcentre.github.io/testdat/>,  
<https://github.com/socialresearchcentre/testdat>

**BugReports** <https://github.com/socialresearchcentre/testdat/issues>

**Depends** R (>= 3.2.2), testthat (>= 2.0.0)

**Imports** dplyr (>= 0.8.0), glue, lifecycle, rlang, stringr, tidyselect

**Suggests** covr, crayon, knitr, labelled, lubridate, openxlsx, rmarkdown

**VignetteBuilder** knitr

**Encoding** UTF-8

**RoxygenNote** 7.1.2

**Collate** 'chk-filter.R' 'chk.R' 'comparison.R' 'deprec-chk.R'  
'deprec-expect.R' 'deprec-reporter.R' 'expect-make.R'  
'expect-chk.R' 'expect-conditional.R' 'expect-data.R'  
'expect-datacomp.R' 'expect-exclusive.R' 'expect-generic.R'  
'expect-labels.R' 'expect-proportion.R' 'expect-unique.R'  
'expectation.R' 'reporter-excel.R' 'reporter-zzz.R'  
'testdat-package.R' 'utils.R' 'zzz.R'

**NeedsCompilation** no

**Author** Danny Smith [aut, cre],  
Kinto Behr [aut],  
The Social Research Centre [cph]

**Maintainer** Danny Smith <danny@gorcha.org>

**Repository** CRAN

**Date/Publication** 2022-01-04 12:50:02 UTC

**R topics documented:**

chk-dates . . . . .	2
chk-dummy . . . . .	3
chk-helper . . . . .	4
chk-labels . . . . .	5
chk-patterns . . . . .	7
chk-text . . . . .	8
chk-uniqueness . . . . .	9
chk-values . . . . .	9
conditional-expectations . . . . .	11
datacomp-expectations . . . . .	12
date-expectations . . . . .	14
exclusivity-expectations . . . . .	15
expect_make . . . . .	16
generic-expectations . . . . .	17
global-data . . . . .	19
label-expectations . . . . .	20
output_results_excel . . . . .	22
pattern-expectations . . . . .	22
proportion-expectations . . . . .	23
text-expectations . . . . .	26
uniqueness-expectations . . . . .	27
value-expectations . . . . .	29
<b>Index</b>	<b>31</b>

---

chk-dates	<i>Checks: dates</i>
-----------	----------------------

---

**Description**

Check that a vector conforms to a given date format such as YYYYMMDD.

**Usage**

```
chk_date_yyyymmdd(x)
```

```
chk_date_yyyymm(x)
```

```
chk_date_yyyy(x)
```

**Arguments**

x                    A vector to check.

**Value**

A logical vector flagging records that have passed or failed the check.

**See Also**

[Checks: data frame helpers](#)

[Expectations: dates](#)

Other vector checks: [chk-dummy](#), [chk-labels](#), [chk-patterns](#), [chk-text](#), [chk-uniqueness](#), [chk-values](#)

**Examples**

```
date <- c(20210101, 20211301, 20210132, 202101, 2021)
chk_date_yyyymmdd(date)
```

```
date <- c(202101, 202112, 202113, 2021)
chk_date_yyyymm(date)
```

```
date <- c("0001", "1688", "1775", "1789", "1791", "1848")
chk_date_yyyy(date)
```

---

chk-dummy

*Checks: dummy*

---

**Description**

These functions provide common, simple data checks.

**Usage**

```
chk_dummy(x)
```

**Arguments**

x                    A vector to check.

**Value**

A logical vector flagging records that have passed or failed the check.

**See Also**

[Checks: data frame helpers](#)

Other vector checks: [chk-dates](#), [chk-labels](#), [chk-patterns](#), [chk-text](#), [chk-uniqueness](#), [chk-values](#)

**Examples**

```
chk_dummy(LETTERS)
```

---

`chk-helper`*Checks: data frame helpers*

---

## Description

These helper functions allowing easy checking using an arbitrary function (`func`) over multiple columns (`vars`) of a data frame (`data`), with an optional filter (`flt`).

## Usage

```
chk_filter(data, vars, func, flt = TRUE, args = list())
```

```
chk_filter_all(data, vars, func, flt = TRUE, args = list())
```

```
chk_filter_any(data, vars, func, flt = TRUE, args = list())
```

## Arguments

<code>data</code>	A data frame to check.
<code>vars</code>	<a href="#">&lt;tidy-select&gt;</a> A set of columns to check.
<code>func</code>	A function to use for checking that takes a vector as the first argument and returns a logical vector of the same length showing whether an element passed or failed.
<code>flt</code>	<a href="#">&lt;data-masking&gt;</a> A filter specifying a subset of the data frame to test.
<code>args</code>	A list of additional arguments to be added to the function calls.

## Details

- `chk_filter()` applies `func` with `args` to `vars` in `data` filtered with `flt` and returns a data frame containing the resulting logical vectors.
- `chk_filter_all()` and `chk_filter_any()` both run `chk_filter()` and return a single logical vector flagging whether *all* or *any* values in each row are TRUE (i.e. the conjunction and disjunction, respectively, of the columns in the output of `chk_filter()`).

## Value

A logical vector or data frame of logical vectors flagging records that have passed or failed the check, with NA where records do not meet the filter condition.

## See Also

Other `chk_*`() functions such as [chk\\_values\(\)](#)

**Examples**

```
# Check that every 4-cylinder car has an engine displacement of < 100 cubic
# inches AND < 100 horsepower - return a data frame
chk_filter(
  mtcars,
  c("disp", "hp"),
  chk_range,
  cyl == 4,
  list(min = 0, max = 100)
)

# Check that every 4-cylinder car has an engine displacement of < 100 cubic
# inches AND < 100 horsepower
chk_filter_all(
  mtcars,
  c("disp", "hp"),
  chk_range,
  cyl == 4,
  list(min = 0, max = 100)
)

# Check that every 4-cylinder car has an engine displacement of < 100 cubic
# inches OR < 100 horsepower
chk_filter_any(
  mtcars,
  c("disp", "hp"),
  chk_range,
  cyl == 4,
  list(min = 0, max = 100)
)

# Check that columns made up of whole numbers are binary
chk_filter_all(
  mtcars,
  where(~ all(. %% 1 == 0)),
  chk_values,
  TRUE,
  list(0:1)
)
```

---

chk-labels

*Checks: labels*

---

**Description**

Check that a vector is labelled in a given way.

**Usage**

```
chk_labels(x, val_labels = NULL, var_label = NULL)
```

**Arguments**

x	A vector to check.
val_labels	What value label check should be performed? One of: <ul style="list-style-type: none"> <li>• A character vector of expected value labels.</li> <li>• A named vector of expected label-value pairs.</li> <li>• TRUE to test for the presence of value labels in general.</li> <li>• FALSE to test for the absence of value labels.</li> <li>• NULL to ignore value labels when checking.</li> </ul>
var_label	What variable label check should be performed? One of: <ul style="list-style-type: none"> <li>• A character vector of expected variable labels.</li> <li>• TRUE to test for the presence of a variable labels.</li> <li>• FALSE to test for the absence of a variable labels.</li> <li>• NULL to ignore the variable label when checking.</li> </ul>

**Value**

A logical vector flagging records that have passed or failed the check.

**See Also**

[Checks: data frame helpers](#)

[Expectations: labels](#)

Other vector checks: [chk-dates](#), [chk-dummy](#), [chk-patterns](#), [chk-text](#), [chk-uniqueness](#), [chk-values](#)

**Examples**

```
df <- data.frame(
  x = labelled::labelled(c("M", "M", "F"), c(Male = "M", Female = "F"), "Sex"),
  y = labelled::labelled(c("M", "M", "F"), c(Male = "M", Female = "F", Other = "X")),
  z = c("M", "M", "F")
)

# Check for a value-label pairing
chk_labels(df$x, c(Male = "M"))

# Check that two variables have the same values
chk_labels(df$x, labelled::val_labels(df$y))

# Check for the presence of a particular label
chk_labels(df$x, "Male")
chk_labels(df$x, var_label = "Sex")
```

```
# Check that a variable is labelled at all
chk_labels(df$z, val_labels = TRUE)
chk_labels(df$z, var_label = TRUE)

# Check that a variable isn't labelled
chk_labels(df$z, val_labels = FALSE)
chk_labels(df$z, var_label = FALSE)
```

---

chk-patterns

*Checks: patterns*

---

## Description

Check that a vector conforms to a certain pattern.

## Usage

```
chk_regex(x, pattern)

chk_max_length(x, len)
```

## Arguments

x	A vector to check.
pattern	A <a href="#">str_detect()</a> pattern to match.
len	Maximum string length.

## Value

A logical vector flagging records that have passed or failed the check.

## See Also

[Checks: data frame helpers](#)

[Expectations: patterns](#)

Other vector checks: [chk-dates](#), [chk-dummy](#), [chk-labels](#), [chk-text](#), [chk-uniqueness](#), [chk-values](#)

## Examples

```
x <- c("a_1", "b_2", "c_2", NA, "NULL")
chk_regex(x, "[a-z]_[0-9]")
chk_max_length(x, 3)
```

---

`chk-text`*Checks: text*

---

### Description

Check character vectors for non-ASCII characters or common NULL value placeholders.

### Usage

```
chk_ascii(x)
```

```
chk_text_miss(x, miss = getOption("testdat.miss_text"))
```

```
chk_text_nmiss(x, miss = getOption("testdat.miss_text"))
```

### Arguments

<code>x</code>	A vector to check.
<code>miss</code>	A vector of values to be treated as missing. The <code>testdat.miss</code> or <code>testdat.miss_text</code> option is used by default.

### Value

A logical vector flagging records that have passed or failed the check.

### See Also

[Checks: data frame helpers](#)

[Expectations: text](#)

Other vector checks: [chk-dates](#), [chk-dummy](#), [chk-labels](#), [chk-patterns](#), [chk-uniqueness](#), [chk-values](#)

### Examples

```
chk_ascii(c("a", "\U1f642")) # detect non-ASCII characters

imported_data <- c(1, "#n/a", 2, "", 3, NA)
chk_text_miss(imported_data)
chk_text_nmiss(imported_data) # Equivalent to !chk_text_miss(imported_data)
```



---

chk-uniqueness	<i>Checks: uniqueness</i>
----------------	---------------------------

---

**Description**

Check that each value in a vector is unique.

**Usage**

```
chk_unique(x)
```

**Arguments**

x                    A vector to check.

**Value**

A logical vector flagging records that have passed or failed the check.

**See Also**

[Checks: data frame helpers](#)

[Expectations: uniqueness](#)

Other vector checks: [chk-dates](#), [chk-dummy](#), [chk-labels](#), [chk-patterns](#), [chk-text](#), [chk-values](#)

**Examples**

```
x <- c(NA, 1:10, NA)
chk_unique(x)
```

```
x <- c(10, 1:10, 10)
chk_unique(x)
```

---

chk-values	<i>Checks: values</i>
------------	-----------------------

---

**Description**

Check that a vector contains only certain values.

**Usage**

```
chk_equals(x, val)

chk_values(x, ..., miss = getOption("testdat.miss"))

chk_range(x, min, max, ...)

chk_blank(x)
```

**Arguments**

x	A vector to check.
val	A scalar value for the equality check.
...	Vectors of valid values.
miss	A vector of values to be treated as missing. The <a href="#">testdat.miss</a> or <a href="#">testdat.miss_text</a> option is used by default.
min	Minimum value for range check.
max	Maximum value for range check.

**Value**

A logical vector flagging records that have passed or failed the check.

**See Also**

[Checks: data frame helpers](#)

[Expectations: values](#)

Other vector checks: [chk-dates](#), [chk-dummy](#), [chk-labels](#), [chk-patterns](#), [chk-text](#), [chk-uniqueness](#)

**Examples**

```
x <- c(NA, 0, 1, 0.5, 0, NA, 99)
chk_blank(x) # Blank
chk_equals(x, 0) # Either blank or 0
chk_values(x, 0, 1) # Either blank, 0, 1, or 99
chk_range(x, 0, 1) # Either blank or in [0,1]
chk_range(x, 0, 1, 99) # Either blank, in [0,1], or equal to 99
```

---

 conditional-expectations

*Expectations: consistency*


---

## Description

These functions test whether multiple conditions coexist.

## Usage

```
expect_cond(cond1, cond2, data = get_testdata())
```

```
expect_base(
  var,
  base,
  miss = getOption("testdat.miss"),
  missing_valid = FALSE,
  data = get_testdata()
)
```

## Arguments

cond1	<data-masking> First condition (antecedent) for consistency check.
cond2	<data-masking> Second condition (consequent) for consistency check.
data	A data frame to test. The <a href="#">global test data</a> is used by default.
var	An unquoted column name to test.
base	<data-masking> The condition that determines which records should be non-missing.
miss	A vector of values to be treated as missing. The <a href="#">testdat.miss</a> option is used by default.
missing_valid	Should missing values be treated as valid for records meeting the base condition? This allows 'one way' base checks. This is FALSE by default.

## Value

`expect_*`() functions are mainly called for their side effects. The expectation signals its result (e.g. "success", "failure"), which is logged by the current [test reporter](#). In a non-testing context the expectation will raise an error with class `expectation_failure` if it fails.

## Functions

- `expect_cond`: Checks the coexistence of two conditions. It can be read as "if cond1 then cond2".
- `expect_base`: A special case that checks missing data against a specified condition. It can be read as "if base then var not missing, if not base then var missing".

## See Also

Other data expectations: [datacomp-expectations](#), [date-expectations](#), [exclusivity-expectations](#), [generic-expectations](#), [label-expectations](#), [pattern-expectations](#), [proportion-expectations](#), [text-expectations](#), [uniqueness-expectations](#), [value-expectations](#)

## Examples

```
my_survey <- data.frame(
  resp_id = 1:5,
  q1a = c(0, 1, 0, 1, 0),
  q1b = c(NA, NA, NA, 1, 0), # Asked if q1a %in% 1
  q2a = c(90, 80, 60, 40, 90),
  q2b = c("", "", NA, "Some reason for low rating", "") # Asked if q2a < 50
)

# Check that q1b has a value if and only if q1a %in% 1
try(expect_base(q1b, q1a %in% 1, data = my_survey)) # Fails for resp_id 2 and 5

# Check that q2b has a value if and only if q2a < 50
expect_base(q2b, q2a < 50, data = my_survey)

# Check that if q1a %in% 0 then q2a > 50 (but not vice-versa)
expect_cond(q1a %in% 0, q2a > 50, data = my_survey)
```

---

datacomp-expectations *Expectations: comparisons*

---

## Description

### [Experimental]

These functions allow for comparison between two data frames.

## Usage

```
expect_valmatch(
  data2,
  vars,
  by,
  not = FALSE,
  flt = TRUE,
  data = get_testdata()
)

expect_subset(data2, by = NULL, not = FALSE, flt = TRUE, data = get_testdata())
```

**Arguments**

data2	The data frame to compare against.
vars	<tidy-select> A set of columns to test.
by	A character vector of columns to join by. See <code>dplyr::join()</code> for details.
not	Reverse the results of the check?
flt	<data-masking> A filter specifying a subset of the data frame to test.
data	A data frame to test. The <code>global test data</code> is used by default.

**Details**

- `expect_valmatch()` compares the observations appearing in one data frame (`data`) to the same observations, as picked out by a key (`by`), in another data frame (`data2`). It fails if the selected columns (`vars`) aren't the same for those observations in both data frames.
- `expect_subset()` compares one data frame (`data`) to another (`data2`) and fails if all of the observations in the first, as picked out by a key (`by`), do not appear in the second.

**Value**

`expect_*()` functions are mainly called for their side effects. The expectation signals its result (e.g. "success", "failure"), which is logged by the current `test reporter`. In a non-testing context the expectation will raise an error with class `expectation_failure` if it fails.

**See Also**

Other data expectations: [conditional-expectations](#), [date-expectations](#), [exclusivity-expectations](#), [generic-expectations](#), [label-expectations](#), [pattern-expectations](#), [proportion-expectations](#), [text-expectations](#), [uniqueness-expectations](#), [value-expectations](#)

**Examples**

```
df1 <- data.frame(
  id = 0:99,
  binomial = sample(0:1, 100, TRUE),
  even = abs(0:99%%2 - 1) * 0:99
)

df2 <- data.frame(
  id = 0:99,
  binomial = sample(0:1, 100, TRUE),
  odd = 0:99%%2 * 0:99
)

# Check that same records 'succeeded' across data frames
try(expect_valmatch(df2, binomial, by = "id", data = df1))

# Check that all records in `df1`, as picked out by `id`, exist in `df2`
expect_subset(df2, by = "id", data = df1)
```

---

date-expectations      *Expectations: dates*

---

### Description

Test whether variables in a data frame conform to a given date format such as YYYYMMDD.

### Usage

```
expect_date_yyyy(vars, flt = TRUE, data = get_testdata())
```

```
expect_date_yyyymm(vars, flt = TRUE, data = get_testdata())
```

```
expect_date_yyyymmdd(vars, flt = TRUE, data = get_testdata())
```

### Arguments

`vars`            [<tidy-select>](#) A set of columns to test.

`flt`             [<data-masking>](#) A filter specifying a subset of the data frame to test.

`data`            A data frame to test. The [global test data](#) is used by default.

### Value

`expect_*`() functions are mainly called for their side effects. The expectation signals its result (e.g. "success", "failure"), which is logged by the current [test reporter](#). In a non-testing context the expectation will raise an error with class `expectation_failure` if it fails.

### See Also

[Checks: date](#)

Other data expectations: [conditional-expectations](#), [datacomp-expectations](#), [exclusivity-expectations](#), [generic-expectations](#), [label-expectations](#), [pattern-expectations](#), [proportion-expectations](#), [text-expectations](#), [uniqueness-expectations](#), [value-expectations](#)

### Examples

```
sales <- data.frame(
  sale_id = 1:5,
  date = c("20200101", "20200101", "20200102", "20200103", "20220101"),
  quarter = c(202006, 202009, 202012, 20203, 20200101),
  published = c(1999, 19991, 21, 0001, 20200101)
)

try(expect_date_yyyymmdd(date, data = sales)) # Full date of sale valid
try(expect_date_yyyymm(quarter, data = sales)) # Quarters given as YYYYMM
try(expect_date_yyyy(published, data = sales)) # Publication years valid
```

---

 exclusivity-expectations

*Expectations: exclusivity*


---

### Description

`expect_exclusive` tests that vars are exclusive - that, if any one of vars is set to `exc_val`, no other column in vars or `var_set` is also set to `exc_val`.

### Usage

```
expect_exclusive(vars, var_set, exc_val = 1, flt = TRUE, data = get_testdata())
```

### Arguments

<code>vars</code>	<code>&lt;tidy-select&gt;</code> A set of columns to test.
<code>var_set</code>	<code>&lt;tidy-select&gt;</code> The full set of columns to check against. This should include all columns specified in the <code>vars</code> argument.
<code>exc_val</code>	The value that flags a variable as "selected" (default: 1)
<code>flt</code>	<code>&lt;data-masking&gt;</code> A filter specifying a subset of the data frame to test.
<code>data</code>	A data frame to test. The <a href="#">global test data</a> is used by default.

### Details

This expectation is designed to check exclusivity in survey multiple response sets, where one response is only valid on its own.

See the example data set below:

- No record should have `q10_98`, "None of the above", selected while also having any other response selected, so we refer to this as an "exclusive" response.
- `expect_exclusive()` checks whether `q10_98` "None of the above" or `q10_99` "Don't know", the exclusive responses, have been selected alongside any other `q10_*` response.
- The expectation fails, since the first record has both `q10_1` and `q10_98` selected.

### Value

`expect_*`() functions are mainly called for their side effects. The expectation signals its result (e.g. "success", "failure"), which is logged by the current [test reporter](#). In a non-testing context the expectation will raise an error with class `expectation_failure` if it fails.

### See Also

Other data expectations: [conditional-expectations](#), [datacomp-expectations](#), [date-expectations](#), [generic-expectations](#), [label-expectations](#), [pattern-expectations](#), [proportion-expectations](#), [text-expectations](#), [uniqueness-expectations](#), [value-expectations](#)

**Examples**

```

my_q_block <- data.frame(
  resp_id = 1:5, # Unique to respondent
  q10_1 = c(1, 1, 0, 0, 0),
  q10_2 = c(0, 1, 0, 0, 0),
  q10_3 = c(0, 0, 1, 0, 0),
  q10_98 = c(1, 0, 0, 1, 0), # None of the above
  q10_99 = c(0, 0, 0, 0, 1) # Item not answered
)

# Make sure that if "None of the above" and "Item skipped" are selected
# none of the other question options are selected:
try(
  expect_exclusive(
    c(q10_98, q10_99),
    starts_with("q10_"),
    data = my_q_block
  )
)

```

---

 expect\_make

*Create an expectation from a check function*


---

**Description**

expect\_make() creates an expectation from a vectorised checking function to allow simple generation of domain specific data checks.

**Usage**

```

expect_make(
  func,
  func_desc = NULL,
  vars = FALSE,
  all = TRUE,
  env = caller_env()
)

```

**Arguments**

func	A function whose first argument takes a vector to check, and returns a logical vector of the same length with the results.
func_desc	A character function description to use in the expectation failure message.
vars	Included for backwards compatibility only.
all	Function to use to combine results for each vector.
env	The parent environment of the function, defaults to the calling environment of expect_make().



**Value**

An expect\_\*() style function.

**Examples**

```
# Create a custom check
chk_binary <- function(x) {
  suppressWarnings(as.integer(x) %in% 0:1)
}

# Create custom expectation function
expect_binary <- expect_make(chk_binary)

# Validate a data frame
try(expect_binary(vs, data = mtcars))
try(expect_binary(cyl, data = mtcars))
```

---

generic-expectations *Expectations: generic helpers*

---

**Description**

These functions allow for testing of multiple columns (vars) of a data frame (data), with an optional filter (flt), using an arbitrary function (func).

**Usage**

```
expect_allany(
  vars,
  func,
  flt = TRUE,
  data = get_testdata(),
  args = list(),
  allany = c(chk_filter_all, chk_filter_any),
  func_desc = NULL
)

expect_all(...)

expect_any(...)
```

**Arguments**

vars	<tidy-select> A set of columns to test.
func	A function to use for testing that takes a vector as the first argument and returns a logical vector of the same length showing whether an element passed or failed.

<code>flt</code>	< <a href="#">data-masking</a> > A filter specifying a subset of the data frame to test.
<code>data</code>	A data frame to test. The <a href="#">global test data</a> is used by default.
<code>args</code>	A named list of arguments to pass to <code>func</code> .
<code>allany</code>	The function to combine the <code>func</code> results for each row.
<code>func_desc</code>	A human friendly description of <code>func</code> to use in the expectation failure message.
<code>...</code>	Arguments to pass to <code>expect_allany()</code> .

### Details

- `expect_allany()` tests the columns in `vars` to see whether `func` returns `TRUE` for each of them, and combines the results for each row using the function in `allany`. Both `expect_all()` and `expect_any()` are wrappers around `expect_allany()`.
- `expect_all()` tests the `vars` to see whether `func` returns `TRUE` for *all* of them (i.e. whether the conjunction of results of applying `func` to each of the `vars` is `TRUE`).
- `expect_any()` tests the `vars` to see whether `func` returns `TRUE` for *any* of them (i.e. whether the disjunction of the results of applying `func` to each of the `vars` is `TRUE`).

### Value

`expect_*`() functions are mainly called for their side effects. The expectation signals its result (e.g. "success", "failure"), which is logged by the current [test reporter](#). In a non-testing context the expectation will raise an error with class `expectation_failure` if it fails.

### See Also

`chk_*`() functions such as [chk\\_values\(\)](#)

Other data expectations: [conditional-expectations](#), [datacomp-expectations](#), [date-expectations](#), [exclusivity-expectations](#), [label-expectations](#), [pattern-expectations](#), [proportion-expectations](#), [text-expectations](#), [uniqueness-expectations](#), [value-expectations](#)

### Examples

```
# Check that every 4-cylinder car has an engine displacement of < 100 cubic
# inches *AND* < 100 horsepower
try(
  expect_all(
    vars = c(displ, hp),
    func = chk_range,
    flt = (cyl == 4),
    args = list(min = 0, max = 100),
    data = mtcars
  )
)

# Check that every 4-cylinder car has an engine displacement of < 100 cubic
# inches *OR* < 100 horsepower
try(
  expect_any(
    vars = c(displ, hp),
```

```
    func = chk_range,
    flt = (cyl == 4),
    args = list(min = 0, max = 100),
    data = mtcars
  )
)

# Check petal dimensions are positive
expect_allany(
  vars = where(is.numeric),
  func = chk_range,
  args = list(min = 0, max = Inf),
  allany = chk_filter_all,
  data = iris
)
```

---

global-data

*Get/set test data*

---

## Description

A global test data set is used to avoid having to re-specify the testing data frame in every test. These functions get and set the global data or set the data for the current context.

## Usage

```
set_testdata(data, quosure = TRUE)
```

```
get_testdata()
```

```
with_testdata(data, code, quosure = TRUE)
```

```
data %E>% code
```

## Arguments

data	Data frame to be used.
quosure	If TRUE, the default, the data frame is stored as a <a href="#">quosure</a> and lazily evaluated when <code>get_testdata()</code> is called, so <code>get_testdata()</code> will return the current state of the data frame. If FALSE, the data frame will be copied and <code>get_testdata()</code> will return the state of the data frame at the time <code>set_testdata()</code> was called.
code	Code to execute with the test data set to data.

**Value**

- `set_testdata()` invisibly returns the previous test data. The test data is returned as it was stored - if it was stored with `quosure = TRUE` it will be returned as a quosure.
- `get_testdata()` returns the current test data frame.
- `with_testdata()` and the test data pipe `%E>%` invisibly return the input data for easy piping.

**Examples**

```
set_testdata(mtcars)
head(get_testdata())

with_testdata(iris, {
  x <- get_testdata()
  print(head(x))
})

mtcars %E>%
  expect_base(mpg, TRUE) %E>%
  expect_range(carb, 1, 8)
```

---

label-expectations      *Expectations: labels*

---

**Description**

Test whether variables in a data frame are labelled in a given way.

**Usage**

```
expect_labels(
  vars,
  val_labels = NULL,
  var_label = NULL,
  flt = TRUE,
  data = get_testdata()
)
```

**Arguments**

`vars`                    `<tidy-select>` A set of columns to test.

`val_labels`              What value label check should be performed? One of:

- A character vector of expected value labels.
- A named vector of expected label-value pairs.
- `TRUE` to test for the presence of value labels in general.
- `FALSE` to test for the absence of value labels.
- `NULL` to ignore value labels when checking.

var_label	What variable label check should be performed? One of: <ul style="list-style-type: none"> <li>• A character vector of expected variable labels.</li> <li>• TRUE to test for the presence of a variable labels.</li> <li>• FALSE to test for the absence of a variable labels.</li> <li>• NULL to ignore the variable label when checking.</li> </ul>
flt	<data-masking> A filter specifying a subset of the data frame to test.
data	A data frame to test. The <a href="#">global test data</a> is used by default.

**Value**

expect\_\*() functions are mainly called for their side effects. The expectation signals its result (e.g. "success", "failure"), which is logged by the current [test reporter](#). In a non-testing context the expectation will raise an error with class `expectation_failure` if it fails.

**See Also**

[Checks: labels](#)

Other data expectations: [conditional-expectations](#), [datacomp-expectations](#), [date-expectations](#), [exclusivity-expectations](#), [generic-expectations](#), [pattern-expectations](#), [proportion-expectations](#), [text-expectations](#), [uniqueness-expectations](#), [value-expectations](#)

**Examples**

```
df <- data.frame(
  x = labelled::labelled(c("M", "M", "F"), c(Male = "M", Female = "F"), "Sex"),
  y = labelled::labelled(c("M", "M", "F"), c(Male = "M", Female = "F", Other = "X")),
  z = c("M", "M", "F")
)

# Check for a value-label pairing
try(expect_labels(x, c(Male = "M"), data = df))

# Check that two variables have the same values
expect_labels(x, labelled::val_labels(df$y), data = df) # N.B. This passes!

# Check for the presence of a particular label
try(expect_labels(x, "Male", data = df))
expect_labels(x, var_label = "Sex", data = df)

# Check that a variable is labelled at all
try(expect_labels(z, val_labels = TRUE, data = df))
try(expect_labels(z, var_label = TRUE, data = df))

# Check that a variable isn't labelled
expect_labels(z, val_labels = FALSE, data = df)
expect_labels(z, var_label = FALSE, data = df)
```

---

output\_results\_excel    *Output ListReporter results in Excel format*

---

### Description

Output formatted ListReporter results to an Excel workbook using [openxlsx](#). The workbook consists of a summary sheet showing aggregated results for each context, and one sheet per context showing details of each unsuccessful test.

### Usage

```
output_results_excel(results, file)
```

### Arguments

results	An object of class <code>testthat_results</code> , e.g. output from <a href="#">test_dir()</a> or <a href="#">test_file()</a> .
file	Output file name

### Value

The return value of [openxlsx::saveWorkbook\(\)](#).

### Examples

```
## Not run:  
# Output the results from running all tests in a directory  
x <- test_dir(".")  
output_results_excel(x, "Test results.xlsx")  
  
## End(Not run)
```

---

pattern-expectations    *Expectations: patterns*

---

### Description

Test whether variables in a data frame conform to a given pattern.

### Usage

```
expect_regex(vars, pattern, flt = TRUE, data = get_testdata())  
  
expect_max_length(vars, len, flt = TRUE, data = get_testdata())
```

**Arguments**

<code>vars</code>	< <a href="#">tidy-select</a> > A set of columns to test.
<code>pattern</code>	A <a href="#">str_detect()</a> pattern to match.
<code>flt</code>	< <a href="#">data-masking</a> > A filter specifying a subset of the data frame to test.
<code>data</code>	A data frame to test. The <a href="#">global test data</a> is used by default.
<code>len</code>	Maximum string length.

**Value**

`expect_*`() functions are mainly called for their side effects. The expectation signals its result (e.g. "success", "failure"), which is logged by the current [test reporter](#). In a non-testing context the expectation will raise an error with class `expectation_failure` if it fails.

**See Also**

Checks: [patterns](#)

Other data expectations: [conditional-expectations](#), [datacomp-expectations](#), [date-expectations](#), [exclusivity-expectations](#), [generic-expectations](#), [label-expectations](#), [proportion-expectations](#), [text-expectations](#), [uniqueness-expectations](#), [value-expectations](#)

**Examples**

```
sales <- data.frame(
  sale_id = 1:5,
  item_code = c("a_1", "b_2", "c_2", NA, "NULL")
)

try(expect_regex(item_code, "[a-z]_[0-9]", data = sales)) # Codes match regex
try(expect_max_length(item_code, 3, data = sales)) # Code width <= 3
```

---

proportion-expectations

*Expectations: proportions*

---

**Description**

These test the proportion of data in a data frame satisfying some condition. The generic functions, `expect_prop_lte()` and `expect_prop_gte()`, can be used with any arbitrary function. The `chk_*`() functions, like `chk_values()`, are useful in this regard.

**Usage**

```

expect_prop_lte(
  var,
  func,
  prop,
  flt = TRUE,
  data = get_testdata(),
  args = list(),
  func_desc = NULL
)

expect_prop_gte(
  var,
  func,
  prop,
  flt = TRUE,
  data = get_testdata(),
  args = list(),
  func_desc = NULL
)

expect_prop_nmiss(
  var,
  prop,
  miss = getOption("testdat.miss"),
  flt = TRUE,
  data = get_testdata()
)

expect_prop_values(var, prop, ..., flt = TRUE, data = get_testdata())

```

**Arguments**

<code>var</code>	An unquoted column name to test.
<code>func</code>	A function to use for testing that takes a vector as the first argument and returns a logical vector of the same length showing whether an element passed or failed.
<code>prop</code>	The proportion of the data frame expected to satisfy the condition.
<code>flt</code>	<a href="#">&lt;data-masking&gt;</a> A filter specifying a subset of the data frame to test.
<code>data</code>	A data frame to test. The <a href="#">global test data</a> is used by default.
<code>args</code>	A named list of arguments to pass to <code>func</code> .
<code>func_desc</code>	A human friendly description of <code>func</code> to use in the expectation failure message.
<code>miss</code>	A vector of values to be treated as missing. The <a href="#">testdat.miss</a> option is used by default.
<code>...</code>	Vectors of valid values.



**Details**

Given the use of quasi-quotation within these functions, to make a new functions using one of the generics such as `expect_prop_gte()` one must defuse the `var` argument using the embracing operator `{{ }}`. See the examples sections for an example.

**Value**

`expect_*`() functions are mainly called for their side effects. The expectation signals its result (e.g. "success", "failure"), which is logged by the current [test reporter](#). In a non-testing context the expectation will raise an error with class `expectation_failure` if it fails.

**See Also**

`chk_*`() functions such as [chk\\_values\(\)](#)

Other data expectations: [conditional-expectations](#), [datacomp-expectations](#), [date-expectations](#), [exclusivity-expectations](#), [generic-expectations](#), [label-expectations](#), [pattern-expectations](#), [text-expectations](#), [uniqueness-expectations](#), [value-expectations](#)

**Examples**

```
sales <- data.frame(
  sale_id = 1:5,
  date = c("20200101", "20200101", "20200102", "20200103", "2020003"),
  sale_price = c(10, 20, 30, 40, -1),
  book_title = c(
    "Phenomenology of Spirit",
    NA,
    "Critique of Practical Reason",
    "Spirit of Trust",
    "Empiricism and the Philosophy of Mind"
  ),
  stringsAsFactors = FALSE
)

# Create a custom expectation
expect_prop_length <- function(var, len, prop, data) {
  expect_prop_gte(
    var = {{var}}, # Notice the use of the embracing operator
    func = chk_max_length,
    prop = prop,
    data = data,
    args = list(len = len),
    func_desc = "length_check"
  )
}

# Use it to check that dates are mostly <= 8 char wide
expect_prop_length(date, 8, 0.9, sales)

# Check price values mostly between 0 and 100
try(expect_prop_values(sale_price, 0.9, 1:100, data = sales))
```

---

text-expectations      *Expectations: text*

---

## Description

Test whether variables in a data frame contain common NULL placeholders.

## Usage

```
expect_text_miss(  
  vars,  
  miss = getOption("testdat.miss_text"),  
  flt = TRUE,  
  data = get_testdata()  
)  
  
expect_text_nmiss(  
  vars,  
  miss = getOption("testdat.miss_text"),  
  flt = TRUE,  
  data = get_testdata()  
)
```

## Arguments

vars	<tidy-select> A set of columns to test.
miss	A vector of values to be treated as missing. The <code>testdat.miss</code> or <code>testdat.miss_text</code> option is used by default.
flt	<data-masking> A filter specifying a subset of the data frame to test.
data	A data frame to test. The <code>global test data</code> is used by default.

## Value

`expect_*`() functions are mainly called for their side effects. The expectation signals its result (e.g. "success", "failure"), which is logged by the current `test reporter`. In a non-testing context the expectation will raise an error with class `expectation_failure` if it fails.

## See Also

[Checks: text](#)

Other data expectations: [conditional-expectations](#), [datacomp-expectations](#), [date-expectations](#), [exclusivity-expectations](#), [generic-expectations](#), [label-expectations](#), [pattern-expectations](#), [proportion-expectations](#), [uniqueness-expectations](#), [value-expectations](#)

## Examples

```
sales <- data.frame(
  sale_id = 1:5,
  date = c("20200101", "null", "20200102", "20200103", "null"),
  sale_price = c(10, -1, 30, 40, -1)
)

# Dates not missing
try(expect_text_nmiss(date, data = sales))

# Date missing if price negative
try(expect_text_miss(date, flt = sale_price %in% -1, data = sales))
```

---

uniqueness-expectations

*Expectations: uniqueness*

---

## Description

These functions test variables for uniqueness.

## Usage

```
expect_unique(
  vars,
  exclude = getOption("testdat.miss"),
  flt = TRUE,
  data = get_testdata()
)

expect_unique_across(
  vars,
  exclude = getOption("testdat.miss"),
  flt = TRUE,
  data = get_testdata()
)

expect_unique_combine(
  vars,
  exclude = getOption("testdat.miss"),
  flt = TRUE,
  data = get_testdata()
)
```

**Arguments**

<code>vars</code>	<code>&lt;tidy-select&gt;</code> A set of columns to test.
<code>exclude</code>	a vector of values to exclude from uniqueness check. The <code>testdat.miss</code> option is used by default. To include all values, set <code>exclude = NULL</code> .
<code>flt</code>	<code>&lt;data-masking&gt;</code> A filter specifying a subset of the data frame to test.
<code>data</code>	A data frame to test. The <code>global test data</code> is used by default.

**Details**

- `expect_unique()` tests a set of columns (`vars`) and fails if the combined columns do not uniquely identify each row.
- `expect_unique_across()` tests a set of columns (`vars`) and fails if each row does not have unique values in each column.
- `expect_unique_combine()` tests a set of columns (`vars`) and fails if any value appears more than once across all of them.

By default the uniqueness check excludes missing values (as specified by the `testdat.miss` option). Setting `exclude = NULL` will include all values.

**Value**

`expect_*()` functions are mainly called for their side effects. The expectation signals its result (e.g. "success", "failure"), which is logged by the current `test reporter`. In a non-testing context the expectation will raise an error with class `expectation_failure` if it fails.

**See Also**

[Checks: uniqueness](#)

Other data expectations: [conditional-expectations](#), [datacomp-expectations](#), [date-expectations](#), [exclusivity-expectations](#), [generic-expectations](#), [label-expectations](#), [pattern-expectations](#), [proportion-expectations](#), [text-expectations](#), [value-expectations](#)

**Examples**

```
student_fruit_preferences <- data.frame(
  student_id = c(1:5, NA, NA),
  apple = c(1, 1, 1, 1, 99, NA, NA),
  orange = c(2, 3, 2, 3, 99, NA, NA),
  banana = c(3, 2, 3, 2, 99, NA, NA),
  phone1 = c(123, 456, 789, 987, 654, NA, NA),
  phone2 = c(345, 678, 987, 567, 000, NA, NA)
)

# Check that key is unique, excluding NAs by default
expect_unique(student_id, data = student_fruit_preferences)

# Check that key is unique, including NAs
try(expect_unique(student_id, exclude = NULL, data = student_fruit_preferences))
```

```

# Check each fruit has unique preference number
try(
  expect_unique_across(
    c(apple, orange, banana),
    data = student_fruit_preferences
  )
)

# Check each fruit has unique preference number, allowing multiple 99 (item
# skipped) codes
expect_unique_across(
  c(apple, orange, banana),
  exclude = c(99, NA), data = student_fruit_preferences
)

# Check that each phone number appears at most once
try(expect_unique_combine(c(phone1, phone2), data = student_fruit_preferences))

```

---

value-expectations      *Expectations: values*

---

## Description

Test whether variables in a data frame contain only certain values.

## Usage

```

expect_values(
  vars,
  ...,
  miss = getOption("testdat.miss"),
  flt = TRUE,
  data = get_testdata()
)

expect_range(vars, min, max, ..., flt = TRUE, data = get_testdata())

```

## Arguments

vars	<tidy-select> A set of columns to test.
...	Vectors of valid values.
miss	A vector of values to be treated as missing. The <code>testdat.miss</code> or <code>testdat.miss_text</code> option is used by default.
flt	<data-masking> A filter specifying a subset of the data frame to test.
data	A data frame to test. The <code>global test data</code> is used by default.
min	Minimum value for range check.
max	Maximum value for range check.

**Value**

`expect_*()` functions are mainly called for their side effects. The expectation signals its result (e.g. "success", "failure"), which is logged by the current [test reporter](#). In a non-testing context the expectation will raise an error with class `expectation_failure` if it fails.

**See Also**

[Checks: values](#)

Other data expectations: [conditional-expectations](#), [datacomp-expectations](#), [date-expectations](#), [exclusivity-expectations](#), [generic-expectations](#), [label-expectations](#), [pattern-expectations](#), [proportion-expectations](#), [text-expectations](#), [uniqueness-expectations](#)

**Examples**

```
sales <- data.frame(  
  sale_id = 1:5,  
  date = c("20200101", "20200101", "20200102", "20200103", "20220101"),  
  sale_price = c(10, 20, 30, 40, -1)  
)
```

```
try(expect_values(date, 20000000:20210000, data = sales)) # Dates between 2000 and 2021  
try(expect_range(sale_price, min = 0, max = Inf, data = sales)) # Prices non-negative
```

# Index

- \* **data expectations**
  - conditional-expectations, [11](#)
  - datacomp-expectations, [12](#)
  - date-expectations, [14](#)
  - exclusivity-expectations, [15](#)
  - generic-expectations, [17](#)
  - label-expectations, [20](#)
  - pattern-expectations, [22](#)
  - proportion-expectations, [23](#)
  - text-expectations, [26](#)
  - uniqueness-expectations, [27](#)
  - value-expectations, [29](#)
- \* **vector checks**
  - chk-dates, [2](#)
  - chk-dummy, [3](#)
  - chk-labels, [5](#)
  - chk-patterns, [7](#)
  - chk-text, [8](#)
  - chk-uniqueness, [9](#)
  - chk-values, [9](#)
- `%E>%` (global-data), [19](#)
- Checks: data frame helpers, [3, 6–10](#)
- Checks: date, [14](#)
- Checks: labels, [21](#)
- Checks: patterns, [23](#)
- Checks: text, [26](#)
- Checks: uniqueness, [28](#)
- Checks: values, [30](#)
- chk-dates, [2](#)
- chk-dummy, [3](#)
- chk-helper, [4](#)
- chk-labels, [5](#)
- chk-patterns, [7](#)
- chk-text, [8](#)
- chk-uniqueness, [9](#)
- chk-values, [9](#)
- chk\_ascii (chk-text), [8](#)
- chk\_blank (chk-values), [9](#)
- chk\_date\_yyyy (chk-dates), [2](#)
- chk\_date\_yyyymm (chk-dates), [2](#)
- chk\_date\_yyyymmdd (chk-dates), [2](#)
- chk\_dummy (chk-dummy), [3](#)
- chk\_equals (chk-values), [9](#)
- chk\_filter (chk-helper), [4](#)
- chk\_filter\_all (chk-helper), [4](#)
- chk\_filter\_any (chk-helper), [4](#)
- chk\_labels (chk-labels), [5](#)
- chk\_max\_length (chk-patterns), [7](#)
- chk\_range (chk-values), [9](#)
- chk\_regex (chk-patterns), [7](#)
- chk\_text\_miss (chk-text), [8](#)
- chk\_text\_nmiss (chk-text), [8](#)
- chk\_unique (chk-uniqueness), [9](#)
- chk\_values (chk-values), [9](#)
- chk\_values(), [4, 18, 25](#)
- conditional-expectations, [11](#)
- datacomp-expectations, [12](#)
- date-expectations, [14](#)
- dplyr:::join(), [13](#)
- exclusivity-expectations, [15](#)
- expect\_all (generic-expectations), [17](#)
- expect\_allany (generic-expectations), [17](#)
- expect\_any (generic-expectations), [17](#)
- expect\_base (conditional-expectations), [11](#)
- expect\_cond (conditional-expectations), [11](#)
- expect\_date\_yyyy (date-expectations), [14](#)
- expect\_date\_yyyymm (date-expectations), [14](#)
- expect\_date\_yyyymmdd (date-expectations), [14](#)
- expect\_exclusive (exclusivity-expectations), [15](#)
- expect\_labels (label-expectations), [20](#)
- expect\_make, [16](#)

- expect\_max\_length
  - (pattern-expectations), 22
- expect\_prop\_gte
  - (proportion-expectations), 23
- expect\_prop\_lte
  - (proportion-expectations), 23
- expect\_prop\_nmiss
  - (proportion-expectations), 23
- expect\_prop\_values
  - (proportion-expectations), 23
- expect\_range (value-expectations), 29
- expect\_regex (pattern-expectations), 22
- expect\_subset (datacomp-expectations), 12
- expect\_text\_miss (text-expectations), 26
- expect\_text\_nmiss (text-expectations), 26
- expect\_unique
  - (uniqueness-expectations), 27
- expect\_unique\_across
  - (uniqueness-expectations), 27
- expect\_unique\_combine
  - (uniqueness-expectations), 27
- expect\_valmatch
  - (datacomp-expectations), 12
- expect\_values (value-expectations), 29
- Expectations: dates, 3
- Expectations: labels, 6
- Expectations: patterns, 7
- Expectations: text, 8
- Expectations: uniqueness, 9
- Expectations: values, 10
  
- generic-expectations, 17
- get\_testdata (global-data), 19
- global test data, 11, 13–15, 18, 21, 23, 24, 26, 28, 29
- global-data, 19
  
- label-expectations, 20
  
- openxlsx, 22
- openxlsx::saveWorkbook(), 22
- output\_results\_excel, 22
  
- pattern-expectations, 22
- proportion-expectations, 23
  
- quosure, 19
  
- set\_testdata (global-data), 19
- str\_detect(), 7, 23
  
- test reporter, 11, 13–15, 18, 21, 23, 25, 26, 28, 30
- test\_dir(), 22
- test\_file(), 22
- testdat.miss, 8, 10, 11, 24, 26, 28, 29
- testdat.miss\_text, 8, 10, 26, 29
- text-expectations, 26
  
- uniqueness-expectations, 27
  
- value-expectations, 29
  
- with\_testdata (global-data), 19