Package ‘testdat’

September 4, 2023

Type Package
Title Data Unit Testing for R
Version 0.4.2
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
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 7.2.3
Collate 'chk-filter.R' 'chk.R' 'comparison.R' 'deprec-chk.R'
     'deprec-expect.R' 'deprec-reporter.R' 'expectation.R'
     'expect-generic.R' 'expect-make.R' 'expect-chk.R'
     'expect-conditional.R' 'expect-data.R' 'expect-datacomp.R'
     'expect-exclusive.R' 'expect-labels.R' 'expect-proportion.R'
     'expect-unique.R' 'expect.depends.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 2023-09-04 00:10:02 UTC
R topics documented:

chk-dates
chk-dummy
chk-helper
chk-labels
chk-patterns
chk-text
chk-uniqueness
chk-values
conditional-expectations
datacomp-expectations
date-expectations
exclusivity-expectations
expect_depends
expect_make
generic-expectations
global-data
label-expectations
output_results_excel
pattern-expectations
proportion-expectations
text-expectations
uniqueness-expectations
value-expectations

Index

chk-dates Checks: dates

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

Usage
chk_date_yyyymmdd(x)
chk_date_yyymm(x)
chk_date_yyyy(x)

Arguments
x A vector to check.
**chk-dummy**

**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**

```r
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**

```r
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`
### chk-helper

**Checks: data frame helpers**

#### Examples

```r
cchk_dummy(LETTERS)
```

#### 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

- `data`: A data frame to check.
- `vars`: A `tidy-select` set of columns to check.
- `func`: 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.
- `flt`: A filter specifying a subset of the data frame to test.
- `args`: 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:
              • 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:
            • A character vector of expected variable labels.
            • TRUE to test for the presence of a variable labels.
            • FALSE to test for the absence of a variable labels.
            • NULL to ignore the variable label when checking.

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 str_detect() 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)
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

x A vector to check.

miss A vector of values to be treated as missing. The testdat.miss or testdat.miss_text 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

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

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 testdat.miss or testdat.miss_text 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
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 global test data 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 testdat.miss 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, expect_depends(), 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, 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)

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 `dplyr::join()` 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 global test data 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, expect_depends(), generic-expectations, label-expectations, pattern-expectations, proportion-expectations, text-expectations, uniqueness-expectations, value-expectations

Examples

```r
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  

**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, expect_depends(), generic-expectations, label-expectations, pattern-expectations, proportion-expectations, text-expectations, uniqueness-expectations, value-expectations

**Examples**

```r
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
```
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

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

Arguments

- `vars` <`tidy-select`> A set of columns to test.
- `var_set` <`tidy-select`> The full set of columns to check against. This should include all columns specified in the `vars` argument.
- `exc_val` The value that flags a variable as "selected" (default: 1)
- `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.

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`, `expect_depends()`, `generic-expectations`, `label-expectations`, `pattern-expectations`, `proportion-expectations`, `text-expectations`, `uniqueness-expectations`, `value-expectations`
Examples

```r
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_depends  

**Expectations: functional dependency**

**Description**

Test whether one set of variables functionally depend on another set of variables.

**Usage**

```r
expect_depends(vars, on, flt = TRUE, data = get_testdata())
```

**Arguments**

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

- `on`  
  <tidy-select> A set of columns which `vars` are expected to depend on.

- `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.

**Details**

One set of variables, X, functionally depends on another, Y, if and only if each value in Y corresponds to exactly one value in X. For instance, `course_duration` and `course_topic` functionally depend on `course_code` if each `course_code` corresponds to just one combination of `course_duration` and `course_topic`. That is, if two records have the same `course_code` then they must have the same `course_duration` and `course_topic`.

See the [wikipedia page](https://en.wikipedia.org/wiki/Functional_dependency) for more information.
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, exclusivity-expectations, generic-expectations, label-expectations, pattern-expectations, proportion-expectations, text-expectations, uniqueness-expectations, value-expectations

Examples

```r
student_course <- data.frame(
  student_id = 1:5,
  course_code = c(1, 2, 1, 3, 4),
  course_duration = c(12, 12, 12, 12, 12),
  course_topic = c("Song", "Dance", "Song", "Painting", "Pottery")
)
# Check that each `course_code` corresponds to exactly one combination of
# `course_duration` and `course_topic`
expect_depends(
  c(course_duration, course_topic),
  on = course_code,
  data = student_course
)
```

---

**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

```r
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.

eqv
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))
expect_any(
  vars,
  func,
  flt = TRUE,
  data = get_testdata(),
  args = list(),
  func_desc = NULL
)

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.
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.
args A named list of arguments to pass to func.
func_desc A human friendly description of func to use in the expectation failure message.

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, expectdepends(), 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(


```r
vars = c(disp, 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(disp, hp),
    func = chk_range,
    flt = (cyl == 4),
    args = list(min = 0, max = 100),
data = mtcars
  )
)

# Check that all variables are numeric:
try(expect_all(
    vars = everything(),
    func = is.numeric,
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.
If `TRUE`, the data frame is stored as a `quosure` and lazily evaluated when `get_testdata()` is called, so `get_testdata()` will return the current state of the data frame.

If `FALSE`, the data frame will be copied and `get_testdata()` will return the state of the data frame at the time `set_testdata()` 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

```r
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)
```

---

### Description

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

### Usage

```r
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:
- A character vector of expected variable labels.
- TRUE to test for the presence of a variable label.
- FALSE to test for the absence of a variable label.
- NULL to ignore the variable label when checking.

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: labels
Other data expectations: conditional-expectations, datacomp-expectations, date-expectations, exclusivity-expectations, expect_depends(), 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)
```r
# 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
```r
output_results_excel(results, file)
```

#### Arguments
- **results**: An object of class `testthat_results`, e.g. output from `test_dir()` or `test_file()`.
- **file**: Output file name

#### Value
The return value of `openxlsx::saveWorkbook()`.

#### Examples
```r
## 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

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

Arguments

- `vars`  
  
  `<tidy-select>` A set of columns to test.
- `pattern`  
  
  A `str_detect()` pattern to match.
- `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.
- `len`  
  
  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`, `expect_depends()`, `generic-expectations`, `label-expectations`, `proportion-expectations`, `text-expectations`, `uniqueness-expectations`, `value-expectations`

Examples

```r
sales <- data.frame(  
  sale_id = 1:5,  
  item_code = c("a_1", "b_2", "c_2", NA, "NULL")  
)
try(expect_regex(item_code, 
  pattern = "[a-z][0-9]", 
  data = sales)) # Codes match regex
try(expect_max_length(item_code, 
  len = 3, 
  data = sales)) # Code width <= 3
```
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

```r
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

- **var**: An unquoted column name 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.
prop  The proportion of the data frame expected to satisfy the condition.
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.
args  A named list of arguments to pass to func.
func_desc  A human friendly description of func to use in the expectation failure message.
miss  A vector of values to be treated as missing. The testdat.miss option is used by default.
...  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, expect_depends(), 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),
  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))

---

**Description**

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

**Usage**

```r
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 testdat.miss or testdat.miss_text 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 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: text

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

Examples

```r
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

```r
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
vars <tidy-select> A set of columns to test.
exclude a vector of values to exclude from uniqueness check. The testdat.miss option is used by default. To include all values, set exclude = NULL.
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.

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, expect_depends(), 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)
value-expectations

# 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**

```r
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.
value-expectations

... Vectors of valid values.

miss A vector of values to be treated as missing. The testdat.miss or testdat.miss_text 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 global test data 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, expect_depends(), 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
expect_depends, 16
generic-expectations, 18
label-expectations, 21
pattern-expectations, 24
proportion-expectations, 25
text-expectations, 27
uniqueness-expectations, 28
value-expectations, 30
* 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), 20

Checks:  data frame helpers, 3, 6–10
Checks: date, 14
Checks: labels, 22
Checks: patterns, 24
Checks: text, 28
Checks: uniqueness, 29
Checks: values, 31
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_yyy (chk-dates), 2
chk_date_yyyymmm (chk-dates), 2
chk_date_yyyymmd (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, 19, 26
conditional-expectations, 11
datacomp-expectations, 12
date-expectations, 14
dplyr::join(), 13
exclusivity-expectations, 15
expect_all (generic-expectations), 18
expect_any (generic-expectations), 18
expect_base (conditional-expectations), 11
expect_cond (conditional-expectations), 11
expect_date_yyy (date-expectations), 14
expect_date_yyyymm (date-expectations), 14
expect_date_yyyymmd (date-expectations), 14
expectdepends, 12–15, 16, 19, 22, 24, 26, 28, 29, 31
expect_exclusive (exclusivity-expectations), 15
expect_labels (label-expectations), 21
INDEX

expect_make, 17
expect_max_length
  (pattern-expectations), 24
expect_prop_gte
  (proportion-expectations), 25
expect_prop_lte
  (proportion-expectations), 25
expect_prop_nmiss
  (proportion-expectations), 25
expect_prop_values
  (proportion-expectations), 25
expect_range
  (value-expectations), 30
expect_regex (pattern-expectations), 24
expect_subset (datacomp-expectations), 12
expect_text_miss (text-expectations), 27
expect_text_nmiss (text-expectations), 27
expect_unique
  (uniqueness-expectations), 28
expect_unique_across
  (uniqueness-expectations), 28
expect_unique_combine
  (uniqueness-expectations), 28
expect_valmatch
  (datacomp-expectations), 12
expect_values (value-expectations), 30
Expectations: dates, 3
Expectations: labels, 6
Expectations: patterns, 7
Expectations: text, 8
Expectations: uniqueness, 9
Expectations: values, 10
generic-expectations, 18
global test data, 11, 13–16, 17, 19, 22, 24, 26, 27, 29, 31
global-data, 20
label-expectations, 21
openxlsx, 23
openxlsx::saveWorkbook(), 23
output_results_excel, 23
pattern-expectations, 24
proportion-expectations, 25
quosure, 21
set_testdata (global-data), 20
str_detect(), 7, 24
test reporter, 11, 13–15, 17, 19, 22, 24, 26, 27, 29, 31
test_dir(), 23
test_file(), 23
testdat.miss, 8, 10, 11, 26, 27, 29, 31
testdat.miss_text, 8, 10, 27, 31
text-expectations, 27
uniqueness-expectations, 28
value-expectations, 30
with_testdata (global-data), 20