Package: validation (via r-universe)

November 3, 2024

Type Package

Title Tools for assisting with package validation within the SCTO package validation projedct of the Statistics and Methodology platform

Version 0.3.7

Maintainer Alan G Haynes <alan.haynes@unibe.ch>

Description The validation platform consists of three GitHub repositories, one of which hosts this package. The others host tests for specific packages, and a location to reports the results of tests and validations. This package provides tools to work with that data.

License MIT

Encoding UTF-8

LazyData true

Imports cranlogs, crayon, dplyr, gh, glue, jsonlite, knitr, lubridate, purrr, readr, pkgsearch, sessioninfo, stringr, testthat, tidyr

RoxygenNote 7.3.1

Suggests rmarkdown, kableExtra, magrittr, devtools

VignetteBuilder knitr

URL https://github.com/SwissClinicalTrialOrganisation/validation

BugReports https://github.com/SwissClinicalTrialOrganisation/validation/issues

Config/testthat/edition 3

Repository https://ctu-bern.r-universe.dev

RemoteUrl https://github.com/SwissClinicalTrialOrganisation/validation

RemoteRef HEAD

RemoteSha b8296bc2aec90ce9f2d7da657617c04ba7213b13

Contents

`add_label` *Interact with the GitHub API to get or set issue information*

Description

These functions provide various ways to interact with github issues.

Open/update an issue:

`post_issue` opens a new issue in a repository, with a defined body and title.

`update_issue` updates an existing issue in a repository (could also be used instead of `add_labels` or `close_issue`).

Getting issues:

`get_issue` downloads a single issue from a repository.

`get_issues` downloads all issues from a repository.

`get_test_reports` downloads all issues from a repository, filtering those with test labels, and converts the issue to a dataframe.

Labels:

`add_label` adds a particular label to an issue.

`remove_label` removes a particular label from an issue.

Comments:

`get_comments` downloads all comments from an issue.

`post_comment` posts a comment to an issue.

Closing an issue:

`close_issue` closes an issue.

These functions are primarily for the use in github actions rather than for the standard user.

See https://docs.github.com/en/rest/issues/issues for full information.

Usage

```
add_label(issue, label, repo = sctoreports())

close_issue(issue, repo = sctoreports())

get_comments(issue, repo = sctoreports())

get_issue(issue, repo = sctoreports())

get_issues(repo = sctoreports())

get_test_reports(repo = sctoreports(), approved_only = FALSE)

post_comment(issue, comment, repo = sctoreports())

post_issue(body, title, repo = sctoreports())

remove_label(issue, label, repo = sctoreports())

update_issue(issue, ..., repo = sctoreports())
```

Arguments

`issue`	issue number to update
`label`	label to remove
`repo`	repository name ... could be e.g. body, title, labels (should be a list), state (open or closed) see https://docs.github.com/en/rest/issues/issues?apiVersion=2022-11-28#update-an-issue
`approved_only`	only return approved packages (logical)
`comment`	the comment to be made
`body`	issue body
`title`	issue title
`...`	arguments passed to gh

Value

a list of comments

dataframe

Examples

```
# remove_label(issue = 1, label = "test")

# post_comment(issue = 1, comment = "This is a test comment")
# get_issue(21)
# post_comment(issue = 1, comment = "This is a test comment")
# post_issue("Some body text", title = "Issue title")
# remove_label(issue = 1, label = "test")

# post_issue("Some body text", title = "Issue title")
```

---

calculate_pkg_score *Convert the text risk metric answers to numeric values and calculate a score*

Description

The score is the mean of the individual scores for each question.

Usage

```
calculate_pkg_score(pkgs = NULL)
```

Arguments

pkgs	a list of package objects

---

check_session *Check a sessions loaded packages against the validated packages list*

Description

Check a sessions loaded packages against the validated packages list

Usage

```
check_session(attached_only = TRUE, approved_only = TRUE)

## S3 method for class 'sctovalidity'
print(x, ...)
```

Arguments

`attached_only`	logical, should all loaded packages be shown (FALSE; default), or only those that are attached (TRUE)
`approved_only`	logical, should only validated (approved) packages be shown (TRUE; default), or also those awaiting approval (FALSE)
`x`	output from `check_session`
`...`	options passed to methods

Examples

```
## Not run:
check_session()

## End(Not run)
```

`gen_pkg_table` *Convert issues from to tables*

Description

These functions are intended for use by a github action to generate a table for the website

This is intended for use by a github action to generate a table for the website

Usage

```
gen_pkg_table(pkgs = NULL, ...)

gen_tests_table(tests = NULL, ...)
```

Arguments

`pkgs`	NULL or a list of issues from `get_issues`
`...`	options passed to `get_issues`
`tests`	NULL or a list of issues from `get_issues`

Functions

- `gen_tests_table()`: Generate a table of function tests

get_12month_downloads *Get number of downloads in the last 12 months*

Description

Note: this is only available for CRAN or bioconductor packages

Usage

```
get_12month_downloads(package, from = Sys.Date() - 365, to = Sys.Date())
```

Arguments

package	package name
from	start date
to	end date

Value

integer number of downloads

Examples

```
get_12month_downloads("ggplot2")
get_12month_downloads("secuTrialR")
```

get_n_deps *Get the number of dependencies for a package*

Description

Get the number of dependencies for a package

Usage

```
get_n_deps(package, fields = c("Depends", "Imports"), ...)
```

Arguments

package	package name
fields	list of fields to check for dependencies. Defaults to `c("Depends", "Imports")`
...	additional arguments passed to `packageDescription`.

Details

The `recursive` and `reverse` arguments are perhaps of most interest. `recursive` details the number of dependencies of the package, including those that are dependencies of dependencies. `reverse` details the number of packages that depend on the package, which may be an indicator of a high quality package. This function uses the function from the tools package.

Value

integer: the number of dependencies

See Also

`packageDescription`

For CRAN packages, `package_dependencies` may be of interest.

Examples

```
## Not run:
get_n_deps("nlme")
get_n_deps("validation")
get_n_deps("dplyr", reverse = TRUE)

## End(Not run)
```

`get_pkg_source` *Derive the location of a package's source code*

Description

This function tries to determine where a package was installed from, e.g. CRAN, GitHub, r-universe, etc. If the package comes from a git repository, it also attempts to extract the repository URL and commit hash.

Usage

```
get_pkg_source(pkg, ...)
```

Arguments

`pkg`	package to check the source of
`...`	parameters passed to other methods

Details

If the package was installed from a r-universe, the function will return the git repository URL and commit hash of the package, rather than the r-universe URL. E.g. this package is installed from the CTU-Bern r-universe, but the function returns the URL of the GitHub repository (`https://github.com/SwissClinicalTri`
and the commit hash.

`get_release_date` *Get the release date of a CRAN package*

Description

CRAN packages have a release date associated with them. This function downloads that date and returns it as a date object.

Usage

```
get_release_date(pkg)
```

Arguments

`pkg`	a package name

Details

If the package does not exist on CRAN, the function asks the user to search elsewhere for the information.

Value

character string containing either the date or a message to check elsewhere

Examples

```
get_release_date("dplyr")
```

`get_tested_pkgs` *Get the list of packages that have tests in a repository*

Description

Tests are assumed to be in a ‘tests‘ directory in the repository.

Usage

```
get_tested_pkgs(repo = sctotests())
```

Arguments

`repo`	repository to check for tests

Value

character vector of package (directory) names

Examples

```
get_tested_pkgs()
```

`get_test_data` *Download datasets from the SCTO validation tests repository*

Description

Tests sometimes require specific datasets. Where a built-in dataset is insufficient, a dataset that exists within the SCTO validation tests repository can be downloaded.

Usage

```
get_test_data(dataset, repo = sctotests(), dir = getwd())
```

Arguments

`dataset`	The filename of the dataset to download
`repo`	the repository to download from
`dir`	the directory to save the dataset to (purely for testing purposes - this option should not be used in everyday use)

Details

This function can be used in the setup-pkg.R file so that the datasets are available to the tests.

The function will download the dataset from the SCTO validation tests repository to the working directory, which testthat sets to the directory containing the test files. Therefore, the dataset can be downloaded and saved and used without referring to any other directory. The setup-pkg.R file might then contain:

```
...
library(testthat)

get_test_data("mtcars.csv")
dat <- read.csv("mtcars.csv")

withr::defer({})
```

Value

TRUE if the dataset is downloaded successfully

Examples

```
# download the file to a temporary directory
tempdir <- tempdir()
get_test_data("mtcars.csv", dir = tempdir)
dat <- read.csv(file.path(tempdir, "mtcars.csv"))
unlink(tempdir)

## Not run:
# within the setup-pkg.R file
get_test_data("mtcars.csv")
# then load the data using an appropriate function
dat <- read.csv("mtcars.csv")

## End(Not run)
```

get_valid_pkgs *Get the list of validated packages from GitHub*

Description

This function downloads issues from github and extracts relevant information from them

Usage

```
get_valid_pkgs(approved_only = TRUE)
```

Arguments

`approved_only`	only return approved packages (logical)

Value

dataframe including package, version, etc.

Examples

```
# only approved packages
get_valid_pkgs()
# all packages, regardless of status
get_valid_pkgs(FALSE)
```

`is_` *Check whether an issue has a particular label*

Description

These functions provide a way to check whether an issue has a particular label.

`is_` is the main function. The others are wrappers around this one.

`is_package` checks for a package label.

`is_test` checks for a test label.

`is_approved` checks for an approved label.

`is_triage` checks for a triage label.

`get_labels` gets all labels associated with an issue.

Usage

```
is_(issues, what)

is_package(issues)

is_test(issues)

is_approved(issues)

is_triage(issues)

get_labels(issue)
```

Arguments

`issues`	a list of issues
`what`	the label to check for
`issue`	a specific issue

Examples

```
# issues <- get_issues()
# is_package(issues)

# issue <- get_issue(21)
# is_package(list(issue))
```

load_pkg_table *Load the flat file of validated packages from github*

Description

Github contains a flat file of the package validation results. This function retrieves that file.
Github contains a flat file of the package validation results. This function retrieves that file.

Usage

```
load_pkg_table()

load_tests_table()
```

Functions

- `load_tests_table()`: Load a dataset of test results

parse_evidence_tab *Convert test evidence table back to a data frame*

Description

In order to print nicely in GitHub, tables are formatted using markdown syntax.

Usage

```
parse_evidence_tab(tab)
```

Arguments

`tab`	a string containing the markdown table

Details

This function can read that syntax and convert it back to a normal data frame.

Value

a data frame

Examples

```
tab <- c("|file      |context |test      | nb| passed|skipped |error | warning|",
         "|:---------|:-------|:---------|--:|------:|:-------|:-----|-------:|",
         "|test-lm.R |lm      |message 1 |  8|      4|FALSE   |FALSE |       4|",
         "|test-lm.R |lm      |message 2 |  3|      3|FALSE   |FALSE |       0|")
parse_evidence_tab(tab)
```

`risk_hook` *Hook for collecting user-defined chunk risk level in markdown documents*

Description

This is a function that can be used with Rmarkdown or quarto to collect the risk associated with a particular chunk. The risk level can then be tabulated towards the end of the document to give an overview of the risk associated with the product.

Usage

```
risk_hook(options, before)
```

Arguments

`options`	chunk options
`before`	whether the hook is called before or after the chunk (only before does anything)

Examples

```
# in e.g. setup chunk
knitr::knit_hooks$set(risk = validation::risk_hook)
RISKENV <- new.env()

# as a chunk option
#```{r highriskchunk, echo=FALSE, risk = "high"}
#```{r mediumriskchunk, echo=FALSE, risk = "medium"}
#```{r lowriskchunk, echo=FALSE, risk = "low"}

# towards the end of the script, tabulate RISKENV$chunk_risk for an overview
```

`sctoreports` *SCTO repositories*

Description

shortcuts for the SCTO package validation repositories. The validation platform consists of 3 repositories: a repository for reporting the validations (pkg_validation), one which contains tests for specific packages (validation_tests), and one for this package itself (validation). These three functions provide the names of the three repositories.

Usage

```
sctoreports()

sctotests()

sctopkg()
```

Value

the repository name (validation)

`test` *Run all tests for a given package*

Description

This function runs all predefined tests for a given package.

Usage

```
test(
  pkg,
  download = TRUE,
  cleanup = FALSE,
  dir = getwd(),
  repo = sctotests(),
  testthat_colours = TRUE
)

## S3 method for class 'validate_result'
print(x, ...)
```

Arguments

`pkg`	package name as a string
`download`	whether to download a fresh set of tests
`cleanup`	whether to delete testing folder
`dir`	directory to download tests to. This should have pkg at the end. If not present, it will be created.
`repo`	storing the tests
`testthat_colours`	include coloured testthat output (the colours are implemented as particular characters in the output, which make it harder to read in some circumstances)
`x`	a validate_result object (the output from ‘test‘)
`...`	additional arguments (not used)

Details

The print method supports the creation of a new issue in the package's repository by presenting the information in a format that can be copied and pasted in the issue template.

Value

Object of class `validate_result`

Examples

```
if(FALSE) test("stats")
```

test_skeleton *Add a new package skeleton*

Description

This function will create the directory if it does not exist, and add a 'info.txt' file with the package name, and a 'setup-pkg.R' file with the necessary setup code.

Usage

```
test_skeleton(pkg, funs, dir = getwd())
```

Arguments

`pkg`	package to be tested
`funs`	functions to be tested
`dir`	where to save the tests/skeleton

Value

a set of files in your working directory, which can be edited and uploaded to github

Examples

```
# test_skeleton(pkg = "dplyr", funs = c("select", "filter"))
```

`test_to_text` *Convert a test result object to a text string for posting to GitHub*

Description

Convert a test result object to a text string for posting to GitHub

Usage

```
test_to_text(x)
```

Arguments

`x`	test result object

Value

a list of character strings

Examples

```
# texts <- test_to_text(x)
# issue <- post_issue(texts$issue_body, texts$issue_title)
# map(texts$issue_tags, ~ add_label(issue$number, .x))
```

`update_pkg_table` *Append new issue data to existing tables*

Description

Where an issue is already in the table, the new data will replace the old data.

Where a package version is already in the table, the new data will replace the old data.

Usage

```
update_pkg_table(...)

update_tests_table(...)
```

Arguments

`...`	options passed to gen_pkg_table e.g. a list of issues from get_pkgs

Functions

- `update_tests_table()`: Append new test data to existing tests table

Examples

```
# update_pkg_table()
```

`validate_pkg_issue` *Validate the inputs for package or function tests*

Description

These functions validate the inputs from package or function tests.

Both functions return a list with two elements - an indicator of whether the validation passed without any errors and a character indicating what the error(s) was/were. The character string is formated as if it were a comment to a GitHub issue.

The functions are not intended to be used directly by the user, but are called by a github action with the results posted as a comment to github.

'validate_pkg_issue' checks the values from a package issue.

'validate_test_issue' checks the values from a function test issue.

Usage

```
validate_pkg_issue(score)

validate_test_issue(issue)
```

Arguments

`score`	the output from calculate_pkg_score for a single package
`issue`	a single function test issue

Value

list wit two elements: score_ok (logical) and message (character)

list wit two elements: ok (logical) and message (character)

Index