Package ‘assertHE’

April 24, 2025

Title Visualisation and Verification of Health Economic Decision Models

Version 1.0.0

Description Designed to help health economic modellers when building and reviewing models.
The visualisation functions allow users to more easily review the network of functions
in a project, and get lay summaries of them. The asserts included are intended to check for common errors,
thereby freeing up time for modellers to focus on tests specific to the individual model in development or review.
For more details see Smith and colleagues (2024)<doi:10.12688/wellcomeopenres.23180.1>.

License MIT + file LICENSE

Encoding UTF-8

RoxygenNote 7.3.2

Suggests testthat (>= 3.0.0), colourpicker, clipr, jsonlite

Config/testthat/edition 3

Depends R (>= 4.1.0)

Imports assertthat, ggplot2, dplyr, utils, visNetwork, covr, htmltools, officer, flextable, knitr, shiny, shinyjs, rstudioapi, roxygen2, methods, waiter, igraph, httr

URL `https://dark-peak-analytics.github.io/assertHE/`,
`https://github.com/dark-peak-analytics/assertHE`

BugReports `https://github.com/dark-peak-analytics/assertHE/issues`

NeedsCompilation no

Author Robert Smith [aut, cre, cph] (<`https://orcid.org/0000-0003-0245-3217`>),
Wael Mohammed [aut] (<`https://orcid.org/0000-0003-0370-4903`>),
Jack Smith [aut],
Oliver Dolin [aut] (<`https://orcid.org/0000-0003-0766-1277`>),
Dark Peak Analytics Ltd [cph, fnd]

Maintainer Robert Smith <rsmith@darkpeakanalytics.com>

Repository CRAN

Date/Publication 2025-04-24 17:10:02 UTC

Contents

`.called_by` *Called By*

Description

Identify functions called by a given function within a specified project folder

Usage

```
.called_by(fname, all_functions, pkg_env)
```

Arguments

`fname`	The name of the target function.
`all_functions`	A character vector of all function names in the project.
`pkg_env`	The package environment where the functions are defined (e.g. global).

Details

The function identifies functions called by the target function `fname` within the specified package environment `pkg_env`. It searches for dependencies within the literal code of the function body and returns a dataframe with two columns ("from" and "to") representing the dependencies. If no dependencies are found, it returns a dataframe with "from" as the target function and "to" as NA.

Note: This function may potentially miss calls if they are in attributes of the closure. For example when function is defined within another function, capturing the environment of the outer function.

Value

A dataframe with two columns ("from" and "to") representing the dependencies of the target function. Returns NA if no dependencies are found.

`.parse_function` *Parse Function*

Description

This function parses an R expression, breaking it down into its components.

Usage

```
.parse_function(x)
```

Arguments

`x`	An R expression to be parsed.

Details

If the input expression x is not an atomic value, symbol, or an environment pointer, the function breaks it up into a list of components. It also handles expressions of the form foo$bar by splitting them up, keeping only the relevant parts for parsing.

If x is a list of expressions, the function recursively parses each expression until they can no longer be listed, filtering out atomic values in the process.

If x is not listable (e.g. a function), it is deparsed into a character string.

Value

A character string or a list of parsed components, depending on the input expression.

---

assertHE_example *Get path to assertHE example*

Description

assertHE comes bundled with a number of sample files in its inst/extdata directory. This function make them easy to access

Usage

```
assertHE_example(file = NULL)
```

Arguments

file	Name of file. If NULL, the example files will be listed.

Value

If file is NULL, returns a character vector containing the names of all files and directories available in the package's directory (extdata). If file specifies the name of an existing example file, returns a character vector of length one containing the full path to that file. Stops with an error if the specified file does not exist within the example directory.

Examples

```
assertHE_example()
assertHE_example("example_scripts/example_tricky_functions.R")
```

check_init *Check and initialize a vector*

Description

This function checks a given vector for several conditions, including values being within the range 0 to 1 inclusive and the sum of values being equal to 1. If the vector is named, the function checks all elements have names and no names are duplicates.

Usage

```
check_init(x)
```

Arguments

x	A numeric vector with named elements.

Value

If successful there is no message, otherwise, it issues warnings with informative messages for each failed condition.

Examples

```
x <- setNames(object = c(0.2, 0.3, 0.4, 0.1), nm = letters[1:4])
check_init(x) # x is a valid input, no warnings issued

x <- setNames(c(0.2, 0.3, 0.4, 0.1), nm = c("H", NA, "NA", "D"))
check_init(x) # Should issue a warning about missing names

x <- c(-2, 0.3, 0.4, 0.1)
check_init(x) # Should issue a warning about a value below 0 and about not summing to 1
```

check_markov_trace *Check Markov Trace*

Description

This function checks the properties of a markov trace conform to expectations. That it is: numeric, values are between 0 and 1 with all rows summing to 1. Also allows users to check that the dead state is monotonically decreasing (if provided)

Usage

```
check_markov_trace(
  m_TR,
  dead_state = NULL,
  confirm_ok = FALSE,
  stop_if_not = FALSE
)
```

Arguments

`m_TR` The markov trace to be checked.

`dead_state` character vector length 1 denoting dead state (e.g. "D")

`confirm_ok` if OK, return a message confirming all checks passed.

`stop_if_not` return error messages. The default (FALSE) returns warnings.

Value

A message indicating whether the matrix passed all the checks or an error message if any check failed.

Examples

```
v_hs_names <- c("H", "S", "D")
n_hs <- length(v_hs_names)
n_t <- 10

m_TR <- matrix(data = NA,
               nrow = n_t,
               ncol = n_hs,
               dimnames = list(NULL, v_hs_names))

m_TR[, "H"] <- seq(1, 0, length.out = n_t)
m_TR[, "S"] <- seq(0, 0.5, length.out = n_t)
m_TR[, "D"] <- 1 - m_TR[, "H"] - m_TR[, "S"]
check_markov_trace(m_TR = m_TR, dead_state = "D", confirm_ok = TRUE)

# the following results in an error because the trace has infeasible values
m_TR[10, "D"] <- 0
m_TR[9, "S"] <- 1
try(check_markov_trace(m_TR = m_TR, stop_if_not = TRUE, dead_state = "D", confirm_ok = TRUE))
```

`check_trans_prob_array` *Check Transition Probability Array*

Description

This function checks the properties of a transition probability array with 2 or three dimensions conform to standard expectations. That it is that each slice is: square, numeric, values are between 0 and 1 with all rows summing to 1. If a dead state is provided, it checks that the dead state -> dead state probability in each slice is equal to 1.

Usage

```
check_trans_prob_array(a_P, dead_state = NULL, stop_if_not = FALSE)
```

Arguments

a_P	The transition probability array to be checked.
dead_state	character vector length 1 denoting dead state (e.g. "D")
stop_if_not	return error messages. The default (FALSE) returns warnings.

Value

A message indicating whether the array passed all the checks or a warning/error message if any check failed.

Examples

```
v_hs_names <- c("H", "S", "D")
n_hs <- length(v_hs_names)
a_P <- array(
  data = 0,
  dim = c(n_hs, n_hs, 1000),
  dimnames = list(v_hs_names, v_hs_names, 1:1000)
)
a_P["H", "S",] <- 0.3
a_P["H", "D",] <- 0.01
a_P["S", "D",] <- 0.1
a_P["S", "H",] <- 0.5

for(x in 1:1000){
  diag(a_P[,,x]) <- 1 - rowSums(a_P[,,x])
}

check_trans_prob_array(a_P = a_P, stop_if_not = FALSE)
# introduce error
a_P["H", "S", 1:10] <- 0

try(check_trans_prob_array(a_P = a_P, stop_if_not = FALSE))
```

check_trans_prob_mat *Check Transition Probability Matrix*

Description

This function checks the properties of a transition probability matrix conform to standard expectations. That it is: square, numeric, values are between 0 and 1 with all rows summing to 1. If a dead state is provided, it checks that the dead state -> dead state probability is 1.

Usage

```
check_trans_prob_mat(
  m_P,
  dead_state = NULL,
  confirm_ok = FALSE,
  stop_if_not = FALSE
)
```

Arguments

`m_P`	The transition probability matrix to be checked.
`dead_state`	character vector length 1 denoting dead state (e.g. "D")
`confirm_ok`	if OK, return a message confirming all checks passed.
`stop_if_not`	return error messages. The default (FALSE) returns warnings.

Value

A message indicating whether the matrix passed all the checks or a warning/error message if any check failed.

Examples

```
v_hs_names <- c(”H”, ”S”, ”D”)
n_hs <- length(v_hs_names)
m_P <- matrix(data = 0, nrow = n_hs, ncol = n_hs,
              dimnames = list(v_hs_names, v_hs_names))
m_P[”H”, ”S”] <- 0.3
m_P[”H”, ”D”] <- 0.01
m_P[”S”, ”D”] <- 0.1
m_P[”S”, ”H”] <- 0.5
diag(m_P) <- (1 - rowSums(m_P))
check_trans_prob_mat(m_P)

# introduce error
m_P[”H”, ”S”] <- 0.2
try(check_trans_prob_mat(m_P,  confirm_ok = TRUE, stop_if_not = TRUE))
```

create_prompt *Create a prompt for a LLM*

Description

Uses the function arguments and function body as inputs to create a prompt for the LLM.

Usage

```
create_prompt(foo_arguments, foo_body, foo_name, foo_desc, foo_title)
```

Arguments

foo_arguments	the arguments to the function
foo_body	the body of the function
foo_name	function name
foo_desc	function description
foo_title	function title

Value

a single prompt in the form of a character string

Examples

```
create_prompt(
foo_arguments = LETTERS[1:3],
foo_body = "D <- A+B+C; return(D)",
foo_name = "calculate_QALYs",
foo_desc = "This function calcs QALYs",
foo_title = "Calculate the QALYs")
```

define_app_server *Create Shiny app server logic*

Description

Create Shiny app server logic

Usage

```
define_app_server(network_object, project_path, foo_path)
```

Arguments

`network_object` visNetwork object to be displayed in the shiny app

`project_path` Path to the project directory

`foo_path` path to the function folder

Value

Shiny app server logic

`define_app_ui` *Create Shiny app UI*

Description

Create Shiny app UI

Usage

```
define_app_ui(network_title)
```

Arguments

`network_title` Character string representing the title of the network to be displayed above the network.

Value

Shiny app user interface

`extract_function_name` *Extract function name from a string*

Description

Extract function name from a long string. This works by identifying "function(" in the string and then finding the operand before and splitting on that before keeping the character there.

Usage

```
extract_function_name(string)
```

Arguments

`string` A string containing a function definition, this must contain the word ’function’

Value

A string containing the function name

Examples

```
extract_function_name("better_name <- function(x){\n more code} asfdas <- function(x){}")
extract_function_name("better_name <- function(x){\n more code}")
```

find_files *find_files*

Description

Find files based upon regular expression searching IMPORTANT - a Directory is NOT a file. (for most instances of file systems)

Usage

```
find_files(
  file_regx = ".R",
  path = ".",
  recursive = TRUE,
  exclude_files = NULL,
  exclude_dirs = NULL
)
```

Arguments

`file_regx`	= ".*" - a regular expression for files to source
`path`	= "." - a path to search
`recursive`	= TRUE - recurse into subdirectories
`exclude_files`	= NULL - regx for files to exclude
`exclude_dirs`	= NULL - regx for directories to exclude

Value

list of files

Examples

```
find_files(file_regx = ".*",  ## any file name
 path = ".*",   # the current directory and all subdirectories
 recursive = FALSE,  # don't recurse
 exclude_files = ".*utility.*", # exclude "utility" anywhere in basename
 exclude_dirs = "\\<tmp\\>|/tmp/|/tmp\\>|\\<tmp/"  # exclude any directory named "tmp", or subdirs
 )
```

find_folder_function_definitions

Creates summary of R files in folder with functions defined within and locations.

Description

Applies find_function_definitions to each file in a folder and aggregate results

Usage

```
find_folder_function_definitions(
  foo_folder = ".",
  f_excl = NULL,
  d_excl = NULL
)
```

Arguments

foo_folder	A folder to apply find_function_definitions to each script in.
f_excl	A regular expression for files to NOT process (basename)
d_excl	A regular expression for directories to NOT process (dirname)

Value

A dataframe containing a column for function string and a column for function location.

Examples

```
# Skip listed files "somefile.R", and "another_file.R"
folder_path <- assertHE_example("example_project")
find_folder_function_definitions(
   foo_folder = folder_path,
   f_excl = "\\b(somefile\\.R|another_file\\.R)\\b"
)
```

find_function_calls_in_file

Find all function calls in file

Description

Searches through a file for function calls using SYMBOL_FUNCTION_CALL

Usage

```
find_function_calls_in_file(
  relative_path = NULL,
  foo_strings,
  filter_for_test_that = FALSE
)
```

Arguments

`relative_path` path of file to search in

`foo_strings` string vector of function names to search for

`filter_for_test_that`
whether to filter for only functions used after the call to test_that. Default FALSE.

Value

a dataframe with the columns 'foo' for function name and 'location' which gives the file in which the function is called with the line in which the function is called appended.

Examples

```
file_path <- assertHE_example("example_project/tests/testthat/test-calculate_costs.R")
find_function_calls_in_file(
  relative_path = file_path,
  foo_strings = "calculate_costs"
)
```

find_function_calls_in_folder

Find specific function calls in a folder

Description

Runs find_function_calls_in_file on all files in a folder, and combined results into a single dataframe

Usage

```
find_function_calls_in_folder(
  test_folder,
  foo_strings,
  filter_for_test_that = FALSE
)
```

Arguments

`test_folder`	folder containing all tests
`foo_strings`	string vector of function names to search for
`filter_for_test_that`	whether to filter for only functions used after the call to test_that. Default FALSE.

Value

dataframe with two columns. 'foo' contains function names, location contains the location of the tests for each function (file and line number).

Examples

```
folder_path <- assertHE_example("example_project/tests/testthat")
find_function_calls_in_folder(
  foo_strings = c("calculate_costs", "calculate_QALYs",
    "create_Markov_trace", "FOO_WITH_NO_TESTS"),
  test_folder = folder_path
)
```

find_function_definitions

Parses an R source file, returns function names defined within.

Description

Using utils::getParseData(), searches for function definitions by matching the FUNCTION keyword (i.e. "function") with it's associated SYMBOL (i.e the function name)

Usage

```
find_function_definitions(filename)
```

Arguments

`filename`	A string containing a path to an R source file

Value

A dataframe with interesting information

Examples

```
file_path <- assertHE_example("example_scripts/example_tricky_functions.R")
find_function_definitions(filename = file_path)
```

`find_next_vector_element` *Find the next element of the vector after a value*

Description

Find the next element of the vector after a value

Usage

```
find_next_vector_element(value, vector, LTE = FALSE)
```

Arguments

`value`	A value of numeric values
`vector`	A vector of numeric values
`LTE`	a boolean to determine collection on "greater than or equal"

Value

The next element of the vector after the value

Examples

```
find_next_vector_element(value = 5, vector = 1:10)
find_next_vector_element(value = 5, vector = 1:4)
find_next_vector_element(value = 5, vector = 1:5, LTE = FALSE)
find_next_vector_element(value = 5, vector = 1:5, LTE = TRUE)
```

`find_previous_vector_element` *Find the previous element of the vector before a value*

Description

Find the previous element of the vector before a value

Usage

```
find_previous_vector_element(value, vector, LTE = FALSE)
```

Arguments

`value`	A value of numeric values
`vector`	A vector of numeric values
`LTE`	a boolean to determine collection on "less than" or "less than equal"

Value

The previous element of the vector before the value

Examples

```
find_previous_vector_element(value = 5, vector = 1:10)
find_previous_vector_element(value = 5, vector = 6:10)
find_previous_vector_element(value = 5, vector = 5:10, LTE = FALSE)
find_previous_vector_element(value = 5, vector = 5:10, LTE = TRUE)
```

get_active_functions *get all active functions that exist in the global environment*

Description

get all active functions that exist in the global environment

Usage

```
get_active_functions(packages = "assertHE")
```

Arguments

packages	a vector containing the names of packages to include in the search

Value

a vector containing the names of all active functions in the global environment

get_file_cheers_classifications *Get cheers classification tags from a given file*

Description

For a provided filepath, identify the cheers classification tags and the function names that follow them.

Usage

```
get_file_cheers_classifications(
  filename,
  cheers_pattern,
  function_pattern = "(\\s|=|-)function\\("
)
```

Arguments

`filename`	A string containing the filepath to the file to be checked
`cheers_pattern`	A string containing the roxygen tag for cheers which is used as an identifier
`function_pattern`	A string containing the pattern to identify functions

Value

A list containing the cheers tags and the function names that follow them

See Also

Other cheers: `get_folder_cheers_classifications()`

`get_folder_cheers_classifications` *Get cheers classification tags from a given folder*

Description

For a provided folder path, identify the cheers classification tags and the function names that follow them.

Usage

```
get_folder_cheers_classifications(path, cheers_pattern, path_ignore = "tests/")
```

Arguments

`path`	A string containing the filepath to the folder to be checked
`cheers_pattern`	A string containing the roxygen tag for cheers which is used as an identifier
`path_ignore`	A string containing the pattern to identify files to ignore

Value

A list containing the cheers tags and the function names that follow them

See Also

Other cheers: `get_file_cheers_classifications()`

get_foo_coverage *Get coverage by function*

Description

Get coverage by function

Usage

```
get_foo_coverage(foo_folder, test_folder)
```

Arguments

foo_folder	folder containing functions
test_folder	folder containing tests

Value

a dataframe with a column for functions and a column for coverage

Examples

```
# Example takes more than 5 seconds to run
if(require(testthat)) {
  folder_path1 <- assertHE_example("example_project/R")
  folder_path2 <- assertHE_example("example_project/tests/testthat")
  get_foo_coverage(
    foo_folder = folder_path1,
    test_folder = folder_path2
  )
}
```

get_function_data *Retrieve Function data to a list*

Description

This function retrieves data about the arguments and body of a specified function.

Usage

```
get_function_data(foo_name, envir = environment())
```

Arguments

`foo_name`	The name of the function to retrieve data from.
`envir`	The environment in which to look for the function.

Value

A list with elements for 'arguments' and 'body' of the specified function.

`get_function_line` *Extract function line in file path*

Description

Extract function line in file path

Usage

```
get_function_line(file_location)
```

Arguments

`file_location`	Character scalar specifying the path of a file.

Value

A numeric scalar

`get_function_path` *Remove artefacts from file path*

Description

Remove artefacts from file path

Usage

```
get_function_path(file_location, project_path)
```

Arguments

`file_location`	Character scalar specifying the path of a file.
`project_path`	Character scalar specifying the path of the project.

Value

A character scalar

get_isolated_foo *Get Isolated Functions*

Description

Get Isolated Functions

Usage

```
get_isolated_foo(df_edges)
```

Arguments

df_edges	A dataframe with two columns ("from" and "to") representing the dependencies.

Value

A vector of isolated function names.

get_roxygen_description *Get Title and Description from Parsed List*

Description

This function extracts the title and description from a parsed list.

Usage

```
get_roxygen_description(parsed_list)
```

Arguments

parsed_list	A list containing parsed elements.

Value

A list containing the title and description.

Examples

```
parsed_list <- list(list(tag = "title", val = "Sample Title"),
                    list(tag = "description", val = "This is a sample description."))
get_roxygen_description(parsed_list)
```

`get_roxygen_description_from_foo` *Get roxygen title and description from function*

Description

Get roxygen title and description from function

Usage

```
get_roxygen_description_from_foo(foo_name)
```

Arguments

`foo_name`	function for which want descriotion

Value

text containing description

`identify_dependencies` *Identify Dependencies*

Description

Identify dependencies between functions.

Usage

```
identify_dependencies(v_unique_foo, pkg_env = environment())
```

Arguments

`v_unique_foo`	Vector of unique function strings.
`pkg_env`	The package environment where the functions are defined (e.g. global).

Value

A dataframe with two columns ("from" and "to") representing the dependencies.

`locate_funcs` *locate_funcs*

Description

locates the lines which define a function within a single file

Usage

```
locate_funcs(file)
```

Arguments

`file`	= a connection object or a character string path to a file.

Value

Returns a data frame with the following columns: func_num: The ID of the function - monotonic increasing from 1. func_start: The line number (within the file) of the function start. func_end: The line number of the function end.

`make_closable_tab` *Create closable shiny tab*

Description

Create closable shiny tab

Usage

```
make_closable_tab(tab_name, content_output_Id, output_type = "text")
```

Arguments

`tab_name`	Character scalar representing the name or title of the shiny tab.
`content_output_Id`	Character scalar representing the id of the shiny tab.
`output_type`	Character scalar specifying the type of rendered output. Default is `"text"` and can also accept `"HTML"`.

Value

A tab that can be passed to `shiny::tabsetPanel()`

`plotNetwork` *Plot Network*

Description

Visualize a network plot using the visNetwork package.

Usage

```
plotNetwork(
  df_edges,
  from_col = "from",
  to_col = "to",
  df_summary,
  df_coverage,
  color_no_test = c(background = "#fad1d0", border = "#9c0000", highlight = "#9c0000"),
  color_with_test = c(background = "#e6ffe6", border = "#65a765", highlight = "#65a765"),
  color_mod_coverage = c(background = "#FFD580", border = "#E49B0F", highlight =
    "#E49B0F"),
  moderate_coverage_range = c(0.2, 0.8),
  show_in_shiny = FALSE,
  network_title = NULL,
  scale_node_size_by_degree = FALSE
)
```

Arguments

`df_edges`	A data frame containing columns "from" and "to" representing the edges of the network.
`from_col`	Name of the column in df_edges representing the source nodes.
`to_col`	Name of the column in df_edges representing the target nodes.
`df_summary`	A summary dataframe containing the information about each function.
`df_coverage`	a summary dataframe with function names and test coverages
`color_no_test`	named vector with hexcodes for background, border and highlight
`color_with_test`	named vector with hexcodes for background, border and highlight
`color_mod_coverage`	named vector with hexcodes for background, border and highlight where coverage moderate
`moderate_coverage_range`	vector of two values giving range defined as moderate coverage.
`show_in_shiny`	logical scalar indicating whether to prepare/deploy the network using a built in shiny app. Default is `FALSE`.
`network_title`	title of the network plot.
`scale_node_size_by_degree`	Scale the node size by the degree centrality of the node.

Value

A visNetwork object representing the network plot.

`plot_PSA_stability` *Plot cumulative mean Probabilistic Sensitivity Analysis results*

Description

This function plots the cumulative mean of incremental net monetary benefit (INMB), incremental cost-effectiveness ratio (ICER), incremental costs, or incremental effects for different strategies compared to a specified comparator.

Usage

```
plot_PSA_stability(
  m_eff,
  m_cost,
  lambda,
  currency_symbol = "$",
  v_strategy_labels = NULL,
  v_strategy_colors = NULL,
  comparator = NULL,
  output = "inmb",
  include_reference_line = TRUE,
  log_x = FALSE
)
```

Arguments

`m_eff` Numeric matrix of effects for different strategies.

`m_cost` Numeric matrix of costs for different strategies.

`lambda` Numeric value specifying the willingness-to-pay threshold for ICER.

`currency_symbol`
String specifying the currency symbol for y-axis labels.

`v_strategy_labels`
Named vector of strategy labels e.g. c("A" = "Strategy A").

`v_strategy_colors`
Named vector of strategy colors e.g. c("A" = "#665BA6").

`comparator` Column name representing the comparator strategy (e.g. "A").

`output` String specifying the type of plot, limited to: "inmb", "icer", "costs", or "effects".

`include_reference_line`
Logical indicating whether to include a reference line.

`log_x` Logical indicating whether to use a logarithmic scale on the x-axis.

Value

A ggplot object representing the cumulative mean PSA stability plot.

Examples

```
# create example matrices
n_psa <- 10000

m_eff <- matrix(data = runif(n = n_psa * 4, min = 0, max = 1),
                nrow = n_psa,
                ncol = 4,
                dimnames = list(NULL, paste0("Strategy ", c("A", "B", "C", "D"))))

m_cost <- matrix(data = runif(n = n_psa * 4, min = 5000, max = 20000),
                 nrow = n_psa,
                 ncol = 4,
                 dimnames = list(NULL, paste0("Strategy ", c("A", "B", "C", "D"))))

v_strategy_colors <- setNames(object = grDevices::palette.colors(n = ncol(m_eff)),
                              nm = colnames(m_eff))

plot_PSA_stability(m_eff = m_eff,
                   m_cost = m_cost,
                   lambda = 20000,
                   currency_symbol = "\u0024",
                   v_strategy_labels = colnames(m_eff),
                   v_strategy_colors = v_strategy_colors,
                   comparator  = colnames(m_eff)[1],
                   output = "inmb",
                   include_reference_line = TRUE,
                   log_x = FALSE)
```

---

processNodes *Process Nodes*

Description

Process unique nodes from a dataframe of edges.

Usage

```
processNodes(df_edges, from_col = "from", to_col = "to")
```

Arguments

df_edges	A data frame containing columns "from" and "to" representing the edges of the network.
from_col	Name of the column in df_edges representing the source nodes.
to_col	Name of the column in df_edges representing the target nodes.

Value

A data frame of unique nodes with labels.

return_message *Extract the content from the output of the LLM*

Description

Extracts content and prints the number of tokens used as a message.

Usage

```
return_message(API_response, verbose = TRUE)
```

Arguments

`API_response`	response from the LLM API
`verbose`	whether to include the message for the number of token's used

Value

A single string summary of the content of the LLM response

run_shiny_app *Run a Shiny app to host a network visualization*

Description

Run a Shiny app to host a network visualization

Usage

```
run_shiny_app(
  uiFunction = define_app_ui,
  serverFunction = define_app_server,
  network_object,
  network_title = "Function Network",
  project_path,
  foo_path
)
```

Arguments

`uiFunction`	Function defining shiny user-interface
`serverFunction`	Function defining shiny server logic
`network_object`	visNetwork object to be displayed in the shiny app
`network_title`	Title to be displayed in hte app above the title
`project_path`	Path to the project directory
`foo_path`	Path to the function folder

Value

A shiny app

`source_funcs` *source_funcs*

Description

Sources *only* the functions discovered in an R file.

Usage

```
source_funcs(file, env)
```

Arguments

`file`	a connection object or a character string path to a file.
`env`	the environment in which to source the functions.

Value

No return value, called for side effects.

IMPORTANT !!!

Sourcing *this* file is a mistake - may result in infinite recursion.

source_lines *source_lines*

Description

Sources specified lines within a single file.

Usage

```
source_lines(file, lines, env)
```

Arguments

file	a connection object or a character string path to a file.
lines	A vector of integers specifying the lines to be sourced.
env	the environment in which to source the lines.

Value

No return value, called for side effects.

IMPORTANT !!!

Sourcing *this* file is a mistake - may result in infinite recursion.

summarise_function_from_arguments_and_body *Summarise a function from its arguments and body*

Description

Summarise a function using a LLM via API and retrieve the result

Usage

```
summarise_function_from_arguments_and_body(
  foo_name,
  foo_arguments,
  foo_body,
  foo_title,
  foo_desc,
  model_name = "gpt-3.5-turbo-0125",
  llm_api_url = Sys.getenv("LLM_API_URL"),
  llm_api_key = Sys.getenv("LLM_API_KEY")
)
```

Arguments

`foo_name`	function name
`foo_arguments`	vector of arguments
`foo_body`	single character containing the unparsed body
`foo_title`	function title
`foo_desc`	function description
`model_name`	name of the LLM to use (default gpt-3.5-turbo-0125)
`llm_api_url`	url to the API for the LLM
`llm_api_key`	key for the API for the LLM

Value

response from LLM containing all pertinant information & tokens used

Examples

```
## Not run:
tmp <- summarise_function_from_arguments_and_body(
  foo_arguments = LETTERS[1:3],
  foo_body = "D <- A+B+C; return(D)",
  model_name = "gpt-3.5-turbo-0125",
  llm_api_url = Sys.getenv("LLM_API_URL"),
  llm_api_key = Sys.getenv("LLM_API_KEY"),
 foo_desc = "add three numbers, these numbers relate to the number of apples on three trees",
  foo_title = "apple adder",
  foo_name = "apple_add"
)
httr::content(tmp)

## End(Not run)
```

`summarise_function_with_LLM` *Summarize a function using a Large Language Model*

Description

This function summarizes another function using a Language Model.

Usage

```
summarise_function_with_LLM(
  foo_name,
  llm_api_url = Sys.getenv("LLM_API_URL"),
  llm_api_key = Sys.getenv("LLM_API_KEY"),
  envir = environment()
)
```

Arguments

`foo_name`	function name
`llm_api_url`	url to the API for the LLM
`llm_api_key`	key for the API for the LLM
`envir`	The environment in which to look for the function.

Value

A character string with a summary of the function based on its arguments and body.

Examples

```
## Not run:
summarise_function_with_LLM(foo_name = "get_active_functions",
                            llm_api_url = Sys.getenv("LLM_API_URL"),
                            llm_api_key = Sys.getenv("LLM_API_KEY"),
                            envir = rlang::ns_env("assertHE"))

## End(Not run)
```

`summarise_model` *Summarise the model functions in a single folder.*

Description

Summarise the model functions in a single folder.

Usage

```
summarise_model(
  project_path = ".",
  foo_folder = "R",
  exclude_files = NULL,
  exclude_dirs = NULL,
  test_folder = NULL,
  output_format = "dataframe"
)
```

Arguments

`project_path`	path to the project folder, if not provided, will use current working directory.
`foo_folder`	path to folder containing all functions for the model
`exclude_files`	A regular expression for files to NOT process (basename)
`exclude_dirs`	A regular expression for directories to NOT process (dirname)
`test_folder`	folder containing all tests
`output_format`	output format to use, defaults to dataframe, options include latex and word.

Value

dataframe with three columns. 'foo_string' contains function names, 'foo_location' contains the location of the function definitions, 'test_location' contains the locations of tests for each function (both file and line number).

Examples

```
project_path <- assertHE_example("example_project")
foo_folder  <- "R"
test_folder <- "tests/testthat"

summarise_model(
  project_path = project_path,
  foo_folder = foo_folder,
  test_folder =  test_folder
)

summarise_model(
  project_path = project_path,
  foo_folder = foo_folder,
  test_folder =  NULL
)
```

---

visualise_project *Visualize Project*

Description

Visualize the dependencies between functions in a project using a network plot.

Usage

```
visualise_project(
  project_path,
  foo_path = "R",
  test_path = NULL,
  exclude_files = NULL,
  exclude_dirs = NULL,
  run_coverage = FALSE,
 color_no_test = c(background = "#fad1d0", border = "#9c0000", highlight = "#9c0000"),
 color_with_test = c(background = "#e6ffe6", border = "#65a765", highlight = "#65a765"),
  color_mod_coverage = c(background = "#FFD580", border = "#E49B0F", highlight =
    "#E49B0F"),
  moderate_coverage_range = c(0.2, 0.8),
  print_isolated_foo = FALSE,
  show_in_shiny = FALSE,
  network_title = "Function Network",
  scale_node_size_by_degree = TRUE
)
```

Arguments

`project_path` Path to the project folder.

`foo_path` Path to the folder containing foo functions.

`test_path` Path to the folder containing test functions.

`exclude_files` A regular expression for files to NOT process (basename)

`exclude_dirs` A regular expression for directories to NOT process (dirname)

`run_coverage` Boolean determining whether to run coverage assessment

`color_no_test` named vector with hexcodes for background, border and highlight

`color_with_test` named vector with hexcodes for background, border and highlight

`color_mod_coverage` named vector with hexcodes for background, border and highlight where coverage moderate

`moderate_coverage_range` vector of two values giving range defined as moderate coverage.

`print_isolated_foo` Print the isolated functions to the console (default false)

`show_in_shiny` logical scalar indicating whether to prepare/deploy the network using a built in shiny app. Default is `FALSE`.

`network_title` title of the network plot.

`scale_node_size_by_degree` Scale the node size by the degree centrality of the node.

Value

A visNetwork object representing the network plot of function dependencies.

Examples

```
# Example takes more than 5 seconds to run
# Visualize project dependencies in HTML
if(require(testthat)) {
  folder_path <- assertHE_example("example_project")
  visualise_project(
    project_path = folder_path,
    foo_path = "R",
    test_path = "tests/testthat",
    run_coverage = TRUE
  )
}

# Visualize project dependencies in shiny
if(interactive()) {
  visualise_project(
    project_path = folder_path,
    foo_path = "R",
    test_path = "tests/testthat",
    run_coverage = TRUE,
    show_in_shiny = TRUE
  )
}
```

wrap_string *Wrap a string to lines of a specified width*

Description

This function takes an input string and wraps it to lines of a specified width, breaking the string at word boundaries.

Usage

```
wrap_string(input_string, width = 80)
```

Arguments

`input_string`	The input string to be wrapped.
`width`	The maximum width of each line. Default is 80 characters.

Value

A character vector where each element represents a line of the wrapped string.

Examples

```
input_string <- "This is a long string that needs to be wrapped to fit within
                a specified width."
wrapped_lines <- wrap_string(input_string, width = 30)
cat(wrapped_lines, sep = "\n")
```

Index