Package 'worcs'

February 2, 2021

Type Package

Date 2021-02-02

Title Workflow for Open Reproducible Code in Science

Version 0.1.8

Description Create reproducible and transparent research projects in 'R'.
This package is based on the Workflow for Open
Reproducible Code in Science (WORCS), a step-by-step procedure based on best
practices for
Open Science. It includes an 'RStudio' project template, several
convenience functions, and all dependencies required to make your project
reproducible and transparent. WORCS is explained in the tutorial paper
by Van Lissa, Brandmaier, Brinkman, Lamprecht, Struiksma, & Vreede (2020).
<doi:10.17605/OSF.IO/ZCVBS>.

License GPL (>= 3)

Encoding UTF-8

LazyData true

URL https://github.com/cjvanlissa/worcs

RoxygenNote 7.1.1

Imports methods, rmarkdown, prereg, gert, ranger, yaml, digest,
rticles

Suggests remotes, tinytex, renv, knitr, missRanger, testthat (>=
2.1.0)

SystemRequirements pandoc (>= 1.14) - http://pandoc.org

VignetteBuilder knitr

NeedsCompilation no

Author Caspar J. van Lissa [aut, cre]
(<https://orcid.org/0000-0002-0808-5024>),
Aaron Peikert [aut] (<https://orcid.org/0000-0001-7813-818X>),
Andreas M. Brandmaier [aut] (<https://orcid.org/0000-0001-8765-6982>)

Maintainer Caspar J. van Lissa <c.j.vanlissa@uu.nl>

Repository CRAN

Date/Publication 2021-02-02 13:20:03 UTC

R topics documented:

`add_manuscript` *Add Rmarkdown manuscript*

Description

Adds an Rmarkdown manuscript to a 'worcs' project.

Usage

```
add_manuscript(
  worcs_directory = ".",
  manuscript = "APA6",
  remote_repo = NULL,
  verbose = TRUE,
  ...
)
```

Arguments

`worcs_directory`	Character, indicating the directory in which to create the manuscript files. Default: '.', which points to the current working directory.
`manuscript`	Character, indicating what template to use for the 'R Markdown' manuscript. Default: 'APA6'. Available choices include: `"APA6","github_document","None","ams_article","a` For more information about `APA6`, see the 'papaja' package, at <https://github.com/crsh/papaja>. For more information about `github_document`, see `github_document`. The remaining formats are documented in the 'rticles' package.
`remote_repo`	Character, 'https' link to the remote repository for this project. This link should have the form `https://[...].git`. This link will be inserted in the draft manuscript.
`verbose`	Logical. Whether or not to print messages to the console during project creation. Default: TRUE
`...`	Additional arguments passed to and from functions.

Value

No return value. This function is called for its side effects.

Examples

```
the_test <- "worcs_manuscript"
old_wd <- getwd()
dir.create(file.path(tempdir(), the_test))
file.create(file.path(tempdir(), the_test, ".worcs"))
add_manuscript(file.path(tempdir(), the_test),
              manuscript = "github_document")
setwd(old_wd)
unlink(file.path(tempdir(), the_test))
```

`add_preregistration` *Add Rmarkdown preregistration*

Description

Adds an Rmarkdown preregistration template to a 'worcs' project.

Usage

```
add_preregistration(
  worcs_directory = ".",
  preregistration = "COS",
  verbose = TRUE,
  ...
)
```

Arguments

`worcs_directory`	Character, indicating the directory in which to create the manuscript files. Default: '.', which points to the current working directory.
`preregistration`	Character, indicating what template to use for the preregistration. Default: `"COS"`; use `"None"` to omit a preregistration. See Details for other available choices.
`verbose`	Logical. Whether or not to print messages to the console during project creation. Default: TRUE
`...`	Additional arguments passed to and from functions.

Details

Available choices include the templates `"COS"`,`"VantVeer"`,`"Brandt"`,`"AsPredicted"`, which are imported from the `cos_prereg` package, and documented there. Furthermore, several unique templates are included with `worcs`:

- `"PSS"`Preregistration and Sharing Software (Krypotos, Klugkist, Mertens, & Engelhard, 2019)
- `"Secondary"`Preregistration for secondary analyses (Mertens & Krypotos, 2019)

Value

No return value. This function is called for its side effects.

Examples

```
the_test <- "worcs_prereg"
old_wd <- getwd()
dir.create(file.path(tempdir(), the_test))
file.create(file.path(tempdir(), the_test, ".worcs"))
add_preregistration(file.path(tempdir(), the_test),
                    preregistration = "COS")
setwd(old_wd)
unlink(file.path(tempdir(), the_test))
```

`add_synthetic` *Add synthetic data to WORCS project*

Description

This function adds a user-specified synthetic data resource for public use to a WORCS project with closed data.

Usage

```
add_synthetic(
  data,
  synthetic_name = paste0("synthetic_", original_name),
  original_name,
  worcs_directory = ".",
  verbose = TRUE,
  ...
)
```

Arguments

`data`	A `data.frame` containing the synthetic data.
`synthetic_name`	Character, naming the file synthetic data should be written to. By default, prepends `"synthetic_"` to the `original_name`.
`original_name`	Character, naming an existing data resource in the WORCS project with which to associate the synthetic `data` object.
`worcs_directory`	Character, indicating the WORCS project directory to which to save data. The default value `"."` points to the current directory.
`verbose`	Logical. Whether or not to print status messages to the console. Default: TRUE
`...`	Additional arguments passed to and from functions.

Value

Returns `NULL` invisibly. This function is called for its side effects.

See Also

open_data closed_data save_data

Examples

```
# Create directory to run the example
old_wd <- getwd()
test_dir <- file.path(tempdir(), "add_synthetic")
dir.create(test_dir)
setwd(test_dir)
worcs:::write_worcsfile(".worcs")
# Prepare data
df <- iris[1:3, ]
# Run closed_data without synthetic
closed_data(df, codebook = NULL, synthetic = FALSE)
# Manually add synthetic
add_synthetic(df, original_name = "df.csv")
# Remove original from file and environment
file.remove("df.csv")
rm(df)
# See that load_data() now loads the synthetic file
```

```
load_data()
# Cleaning example directory
setwd(old_wd)
unlink(test_dir, recursive = TRUE)
```

---

check_worcs *Evaluate project with respect to WORCS checklist*

Description

Evaluates whether a project meets the criteria of the WORCS checklist (see `worcs_checklist`).

Usage

```
check_worcs(path = ".", verbose = TRUE)
```

Arguments

`path`	Character. Path to a WORCS project folder (a project with a `.worcs` file). Default: '.' (path to current directory).
`verbose`	Logical. Whether or not to show status messages while evaluating the checklist. Default: `TRUE`.

Value

A `data.frame` with a description of the criteria, and a column with evaluations (`$pass`). For criteria that must be evaluated manually, `$pass` will be `FALSE`.

Examples

```
example_dir <- file.path(tempdir(), "badge")
dir.create(example_dir)
write("a", file.path(example_dir, ".worcs"))
check_worcs(path = example_dir)
```

---

cite_all *Comprehensive citation Knit function for 'RStudio'*

Description

This is a wrapper for `render`. First, this function parses the citations in the document, converting citations marked with double at sign, e.g.: `@@reference2020`, into normal citations, e.g.: `@reference2020`. Then, it renders the file.

Usage

```
cite_all(...)
```

Arguments

`...`	All arguments are passed to `render`.

Value

Returns `NULL` invisibly. This function is called for its side effect of rendering an 'R Markdown' file.

Examples

```
# NOTE: Do not use this function interactively, as in the example below.
# Only specify it as custom knit function in an 'R Markdown' file, like so:
# knit: worcs::cite_all

if (rmarkdown::pandoc_available("1.14")){
  file_name <- file.path(tempdir(), "citeall.Rmd")
  loc <- rmarkdown::draft(file_name,
                          template = "github_document",
                          package = "rmarkdown",
                          create_dir = FALSE,
                          edit = FALSE)
  write(c("", "Optional reference: @reference2020"),
        file = file_name, append = TRUE)
  cite_all(file_name)
}
```

`cite_essential` *Essential citations Knit function for 'RStudio'*

Description

This is a wrapper for `render`. First, this function parses the citations in the document, removing citations marked with double at sign, e.g.: `@@reference2020`. Then, it renders the file.

Usage

```
cite_essential(...)
```

Arguments

`...`	All arguments are passed to `render`.

Value

Returns `NULL` invisibly. This function is called for its side effect of rendering an 'R Markdown' file.

Examples

```
# NOTE: Do not use this function interactively, as in the example below.
# Only specify it as custom knit function in an R Markdown file, like so:
# knit: worcs::cite_all

if (rmarkdown::pandoc_available("1.14")){
  file_name <- tempfile("citeessential", fileext = ".Rmd")
  rmarkdown::draft(file_name,
                   template = "github_document",
                   package = "rmarkdown",
                   create_dir = FALSE,
                   edit = FALSE)
  write(c("", "Optional reference: @reference2020"),
        file = file_name, append = TRUE)
  cite_essential(file_name)
}
```

closed_data *Use closed data in WORCS project*

Description

This function saves a data.frame as a `.csv` file (using `write.csv`), stores a checksum in '.worcs', appends the `.gitignore` file to exclude `filename`, and saves a synthetic copy of `data` for public use. To generate these synthetic data, the function `synthetic` is used.

Usage

```
closed_data(
  data,
  filename = paste0(deparse(substitute(data)), ".csv"),
  codebook = paste0("codebook_", deparse(substitute(data)), ".Rmd"),
  worcs_directory = ".",
  synthetic = TRUE,
  ...
)
```

Arguments

`data`	A data.frame to save.
`filename`	Character, naming the file data should be written to. By default, constructs a filename from the name of the object passed to `data`.
`codebook`	Character, naming the file the codebook should be written to. An 'R Markdown' codebook will be created and rendered to `github_document` ('markdown' for 'GitHub'). By default, constructs a filename from the name of the object passed to `data`, adding the word 'codebook'. Set this argument to `NULL` to avoid creating a codebook.

`worcs_directory`	Character, indicating the WORCS project directory to which to save data. The default value `"."` points to the current directory.
`synthetic`	Logical, indicating whether or not to create a synthetic dataset using the `synthetic` function. Additional arguments for the call to `synthetic` can be passed through `...`.
`...`	Additional arguments passed to and from functions.

Value

Returns `NULL` invisibly. This function is called for its side effects.

See Also

open_data closed_data save_data

Examples

```
old_wd <- getwd()
test_dir <- file.path(tempdir(), "data")
dir.create(test_dir)
setwd(test_dir)
worcs:::write_worcsfile(".worcs")
df <- iris[1:3, ]
closed_data(df, codebook = NULL)
setwd(old_wd)
unlink(test_dir, recursive = TRUE)
```

`descriptives` *Describe a dataset*

Description

Provide descriptive statistics for a dataset.

Usage

```
descriptives(x, ...)
```

Arguments

`x`	An object for which a method exists.
`...`	Additional arguments.

Value

A `data.frame` with descriptive statistics for `x`.

Examples

```
descriptives(iris)
```

---

export_project *Export project to .zip file*

Description

Export project to .zip file

Usage

```
export_project(zipfile = NULL, worcs_directory = ".", open_data = TRUE)
```

Arguments

zipfile	Character. Path to a `.zip` file that is to be created. The default argument `NULL` creates a `.zip` file in the directory one level above the 'worcs' project directory. By default, all files tracked by 'Git' are included in the `.zip` file, excluding 'data.csv' if `open_data = FALSE`.
worcs_directory	Character. Path to the WORCS project directory to export. Defaults to `"."`, which refers to the current working directory.
open_data	Logical. Whether or not to include the original data, 'data.csv', if this file exists. If `open_data = FALSE` and an open data file does exist, then it is excluded from the `.zip` file. If it does not yet exist, a synthetic data set is generated and added to the `.zip` file.

Value

Logical, indicating the success of the operation. This function is called for its side effect of creating a `.zip` file.

Examples

```
export_project(worcs_directory = tempdir())
```

git_ignore *Modify .gitignore file*

Description

Arguments passed through `...` are added to the .gitignore file. Elements already present in the file are modified. When `ignore = TRUE`, the arguments are added to the .gitignore file, which will cause 'Git' to not track them.

When `ignore = FALSE`, the arguments are prepended with `!`, This works as a "double negation", and will cause 'Git' to track the files.

Usage

```
git_ignore(..., ignore = TRUE, repo = ".")
```

Arguments

`...`	Any number of character arguments, representing files to be added to the .gitignore file.
`ignore`	Logical. Whether or not 'Git' should ignore these files.
`repo`	a path to an existing repository, or a git_repository object as returned by git_open, git_init or git_clone.

Value

No return value. This function is called for its side effects.

Examples

```
dir.create(".git")
git_ignore("ignorethis.file")
unlink(".git", recursive = TRUE)
file.remove(".gitignore")
```

git_update *Add, commit, and push changes.*

Description

This function is a wrapper for `git_add`, `git_commit`, and `git_push`. It adds all locally changed files to the staging area of the local 'Git' repository, then commits these changes (with an optional) `message`, and then pushes them to a remote repository. This is used for making a "cloud backup" of local changes. Do not use this function when working with privacy sensitive data, or any other file that should not be pushed to a remote repository. The `git_add` argument `force` is disabled by default, to avoid accidentally committing and pushing a file that is listed in `.gitignore`.

Usage

```
git_update(
  message = paste0("update ", Sys.time()),
  files = ".",
  repo = ".",
  author,
  committer,
  remote,
  refspec,
  password,
  ssh_key,
  mirror,
  force,
  verbose = TRUE
)
```

Arguments

`message`	a commit message
`files`	vector of paths relative to the git root directory. Use "." to stage all changed files.
`repo`	a path to an existing repository, or a git_repository object as returned by git_open, git_init or git_clone.
`author`	A git_signature value, default is git_signature_default.
`committer`	A git_signature value, default is same as author
`remote`	name of a remote listed in git_remote_list()
`refspec`	string with mapping between remote and local refs
`password`	a string or a callback function to get passwords for authentication or password protected ssh keys. Defaults to askpass which checks getOption('askpass').
`ssh_key`	path or object containing your ssh private key. By default we look for keys in ssh-agent and credentials::ssh_key_info.
`mirror`	use the –mirror flag
`force`	use the –force flag
`verbose`	display some progress info while downloading

Value

No return value. This function is called for its side effects.

Examples

```
git_update()
```

git_user *Set global 'Git' credentials*

Description

This function is a wrapper for `git_config_global_set`. It sets two name/value pairs at once: `name = "user.name"` is set to the value of the `name` argument, and `name = "user.email"` is set to the value of the `email` argument.

Usage

```
git_user(name, email, overwrite = !has_git_user(), verbose = TRUE)
```

Arguments

`name`	Character. The user name you want to use with 'Git'.
`email`	Character. The email address you want to use with 'Git'.
`overwrite`	Logical. Whether or not to overwrite existing 'Git' credentials. Use this to prevent code from accidentally overwriting existing 'Git' credentials. The default value uses `has_git_user` to set overwrite to `FALSE` if user credentials already exist, and to `TRUE` if no user credentials exist.
`verbose`	Logical. Whether or not to print status messages to the console. Default: TRUE

Value

No return value. This function is called for its side effects.

Examples

```
do.call(git_user, worcs:::get_user())
```

has_git_user *Check whether global 'Git' credentials exist*

Description

Check whether the values `user.name` and `user.email` exist exist in the 'Git' global configuration settings. Uses `git_config_global`.

Usage

```
has_git_user()
```

Value

Logical, indicating whether 'Git' global configuration settings could be retrieved, and contained the values `user.name` and `user.email`.

Examples

```
has_git_user()
```

`load_data` *Load WORCS project data*

Description

Scans the WORCS project file for data that have been saved using `open_data` or `closed_data`, and loads these data into the global (working) environment. The function will load the original data if available on the current system. If only a synthetic dataset is available, this function loads the synthetic data. The name of the object containing the data is derived from the file name by removing the file extension, and, when applicable, the prefix `"synthetic_"`. Thus, both `"data.csv"` and `"synthetic_data.csv"` will be loaded into an object called `data`.

Usage

```
load_data(
  worcs_directory = ".",
  to_envir = TRUE,
  envir = parent.frame(1),
  verbose = TRUE
)
```

Arguments

`worcs_directory`	Character, indicating the WORCS project directory from which to load data. The default value `"."` points to the current directory.
`to_envir`	Logical, indicating whether to load objects directly into the environment, or return a `list` containing the objects. The environment is designated by argument `envir`. Loading objects directly into the global environment is user-friendly, but has the risk of overwriting an existing object with the same name, as explained in `load`. The function `load_data` gives a warning when this happens.
`envir`	The environment where the data should be loaded. The default value `parent.frame(1)` refers to the global environment in an interactive session.
`verbose`	Logical. Whether or not to print status messages to the console. Default: TRUE

Value

Returns a list invisibly. If `to_envir = TRUE`, this list contains the loaded data files. If `to_envir = FALSE`, the list is empty, and the loaded data files are attached directly to the global environment.

Examples

```
test_dir <- file.path(tempdir(), "loaddata")
old_wd <- getwd()
dir.create(test_dir)
setwd(test_dir)
worcs:::write_worcsfile(".worcs")
df <- iris[1:5, ]
suppressWarnings(closed_data(df, codebook = NULL))
load_data()
data
rm("data")
file.remove("data.csv")
load_data()
data
setwd(old_wd)
unlink(test_dir, recursive = TRUE)
```

`load_entrypoint` *Load project entry points*

Description

Loads the designated project entry point into the default editor, using `file.edit`.

Usage

```
load_entrypoint(worcs_directory = ".", verbose = TRUE, ...)
```

Arguments

`worcs_directory`	Character, indicating the WORCS project directory to which to save data. The default value `"."` points to the current directory.
`verbose`	Logical. Whether or not to print status messages to the console. Default: TRUE
`...`	Additional arguments passed to `file.edit`.

Value

No return value. This function is called for its side effects.

Examples

```
## Not run:
# Create directory to run the example
old_wd <- getwd()
test_dir <- file.path(tempdir(), "entrypoint")
dir.create(test_dir)
setwd(test_dir)
```

```
# Prepare worcs file and dummy entry point
worcs:::write_worcsfile(".worcs", entry_point = "test.txt")
writeLines("Hello world", con = file("test.txt", "w"))
# Demonstrate load_entrypoint()
load_entrypoint()
# Cleaning example directory
setwd(old_wd)
unlink(test_dir, recursive = TRUE)

## End(Not run)
```

`make_codebook` *Create codebook for a dataset*

Description

Creates a codebook for a dataset in 'R Markdown' format, and renders it to 'markdown' for 'GitHub'. A codebook contains metadata and documentation for a data file. We urge users to customize the automatically generated 'R Markdown' document and re-knit it, for example, to add a paragraph with details on the data collection procedures. The variable descriptives are stored in a `.csv` file, which can be edited in 'R' or a spreadsheet program. Columns can be appended, and we encourage users to complete at least the following two columns in this file:

- category Describe the type of variable in this column. For example: "morality".
- description Provide a plain-text description of the variable. For example, the full text of a questionnaire item: "People should be willing to do anything to help a member of their family".

Re-knitting the 'R Markdown' file (using `render`) will transfer these changes to the 'markdown' file for 'GitHub'.

Usage

```
make_codebook(
  data,
  filename = "codebook.Rmd",
  render_file = TRUE,
  csv_file = gsub("Rmd$", "csv", filename),
  verbose = TRUE
)
```

Arguments

`data`	A data.frame for which to create a codebook.
`filename`	Character. File name to write the codebook `rmarkdown` file to.
`render_file`	Logical. Whether or not to render the document.

`csv_file`	Character. File name to write the codebook `rmarkdown` file to. By default, uses the filename stem of the `filename` argument. Set to `NULL` to write the codebook only to the 'R Markdown' file, and not to `.csv`.
`verbose`	Logical. Whether or not to print status messages to the console. Default: TRUE

Value

`Logical`, indicating whether or not the operation was successful. This function is mostly called for its side effect of rendering an 'R Markdown' codebook.

Examples

```
if(rmarkdown::pandoc_available("1.14")){
  library(rmarkdown)
  library(knitr)
  filename <- tempfile("codebook", fileext = ".Rmd")
  make_codebook(iris, filename = filename, csv_file = NULL)
  unlink(c(
    ".worcs",
    filename,
    gsub("\\.Rmd", "\\.md", filename),
    gsub("\\.Rmd", "\\.html", filename),
    gsub("\\.Rmd", "_files", filename)
  ), recursive = TRUE)
}
```

`notify_synthetic` *Notify the user when synthetic data are being used*

Description

This function prints a notification message when some or all of the data used in a project are synthetic (see `closed_data` and `synthetic`). See details for important information.

Usage

```
notify_synthetic(..., msg = NULL)
```

Arguments

`...`	Objects of class `worcs_data`. The function will check if these are original or synthetic data.
`msg`	Expression containing the message to print in case not all `worcs_data` are original. This message may refer to `is_synth`, a logical vector indicating which `worcs_data` objects are synthetic.

Details

The preferred way to use this function is to provide specific data objects in the function call, using the `...` argument. If no such objects are provided, `notify_synthetic` will scan the parent environment for objects of class `worcs_data`.

This function is emphatically designed to be included in an 'R Markdown' file, to dynamically generate a notification message when a third party 'Knits' such a document without having access to all original data.

Value

No return value. This function is called for its side effect of printing a notification message.

See Also

closed_data synthetic add_synthetic

Examples

```
df <- iris
class(df) <- c("worcs_data", class(df))
attr(df, "type") <- "synthetic"
notify_synthetic(df, msg = "synthetic")
```

`open_data` *Use open data in WORCS project*

Description

This function saves a data.frame as a `.csv` file (using `write.csv`), stores a checksum in '.worcs', and amends the `.gitignore` file to exclude `filename`.

Usage

```
open_data(
  data,
  filename = paste0(deparse(substitute(data)), ".csv"),
  codebook = paste0("codebook_", deparse(substitute(data)), ".Rmd"),
  worcs_directory = ".",
  ...
)
```

Arguments

`data`	A data.frame to save.
`filename`	Character, naming the file data should be written to. By default, constructs a filename from the name of the object passed to `data`.

`codebook`	Character, naming the file the codebook should be written to. An 'R Markdown' codebook will be created and rendered to `github_document` ('markdown' for 'GitHub'). By default, constructs a filename from the name of the object passed to `data`, adding the word 'codebook'. Set this argument to `NULL` to avoid creating a codebook.
`worcs_directory`	Character, indicating the WORCS project directory to which to save data. The default value `"."` points to the current directory.
`...`	Additional arguments passed to and from functions.

Value

Returns `NULL` invisibly. This function is called for its side effects.

See Also

open_data closed_data save_data

Examples

```
test_dir <- file.path(tempdir(), "data")
old_wd <- getwd()
dir.create(test_dir)
setwd(test_dir)
worcs:::write_worcsfile(".worcs")
df <- iris[1:5, ]
open_data(df, codebook = NULL)
setwd(old_wd)
unlink(test_dir, recursive = TRUE)
```

`skew_kurtosis` *Calculate skew and kurtosis*

Description

Calculate skew and kurtosis, standard errors for both, and the estimates divided by two times the standard error. If this latter quantity exceeds an absolute value of 1, the skew/kurtosis is significant. With very large sample sizes, significant skew/kurtosis is common.

Usage

```
skew_kurtosis(x, verbose = FALSE, se = FALSE, ...)
```

Arguments

`x`	An object for which a method exists.
`verbose`	Logical. Whether or not to print messages to the console, Default: FALSE
`se`	Whether or not to return the standard errors, Default: FALSE
`...`	Additional arguments to pass to and from functions.

Value

A `matrix` of skew and kurtosis statistics for `x`.

Examples

```
skew_kurtosis(datasets::anscombe)
```

`synthetic` *Generate synthetic data*

Description

Generates a synthetic version of a `data.frame`, with similar characteristics to the original. See Details for the algorithm used.

Usage

```
synthetic(
  data,
  model_expression = ranger(x = x, y = y),
  predict_expression = predict(model, data = xsynth)$predictions,
  missingness_expression = NULL,
  verbose = TRUE
)
```

Arguments

`data` A data.frame of which to make a synthetic version.

`model_expression`
: An R-expression to estimate a model. Defaults to `ranger(x = x,y = y)`, which uses the fast implementation of random forests in `ranger`. The expression is evaluated in an environment containing objects `x` and `y`, where `x` is a `data.frame` with the predictor variables, and `y` is a `vector` of outcome values (see Details).

`predict_expression`
: An R-expression to generate predicted values based on the model estimated by `model_expression`. Defaults to `predict(model,data = xsynth)$predictions`. This expression must return a vector of predicted values. The expression is evaluated in an environment containing objects `model` and `xsynth`, where `model` is the model estimated by `model_expression`, and `xsynth` is the `data.frame` of synthetic data used to predict the next column (see Details).

`missingness_expression`
: Optional. An R-expression to impute missing values. Defaults to `NULL`, which means listwise deletion is used. The expression is evaluated in an environment containing the object `data`, as specified in the call to `synthetic`. It must return a `data.frame` with the same dimensions and column names as the original data. For example, use `missingness_expression = missRanger::missRanger(data = data)` for a fast implementation of the excellent 'missForest' single imputation technique.

`verbose`	Logical, Default: TRUE. Whether to show a progress bar while running the algorithm and provide informative messages.

Details

Based on the work by Nowok, Raab, and Dibben (2016), this function uses a simple algorithm to generate a synthetic dataset with similar characteristics to the original. The algorithm is as follows:

1. Let x be the original data.frame, with columns 1:j
2. Let xsynth be a synthetic data.frame, with columns 1:j
3. Column 1 of xsynth is a bootstrapped version of column 1 of x
4. Using `model_expression`, a predictive model is built for column c, for c along 2:j, with c predicted from columns 1:(c-1) of the original data.
5. Using `predict_expression`, columns 1:(c-1) of the synthetic data are used to predict synthetic values for column c.

Variables are thus imputed in order of occurrence in the `data.frame`. To impute in a different order, reorder the data.

Note that, for data synthesis to work properly, it is essential that the `class` of variables is defined correctly. The default algorithm `ranger` supports numeric, integer, and factor types. Other types of variables should be converted to one of these types, or users can use a custom `model_expression` and `predict_expressio` when calling `synthetic`.

Note that for data synthesis to work properly, it is essential that the `class` of variables is defined correctly. The default algorithm `ranger` supports numeric, integer, factor, and logical data. Other types of variables should be converted to one of these types.

Users can provide use a custom `model_expression` and `predict_expression` to use a different algorithm when calling `synthetic`.

As demonstrated in the example, users could call `lm` as a `model_expression` to use linear regression, which preserves linear marginal relationships but can give rise to values out of range of the original data. Or users could call `sample` as a `predict_expression` to bootstrap each variable, a very quick solution that maintains univariate distributions but loses all marginal relationships. These examples are not exhaustive, and users can even create custom functions.

Value

A `data.frame` with synthetic data, based on `data`.

References

Nowok, B., Raab, G.M and Dibben, C. (2016). synthpop: Bespoke creation of synthetic data in R. Journal of Statistical Software, 74(11), 1-26. <doi:10.18637/jss.v074.i11>.

Examples

```
# Example using the iris dataset and default ranger algorithm
iris_syn <- synthetic(iris)

# Example using lm as prediction algorithm (only works for numeric variables)
# note that, within the model_expression, a new data.frame is created because
# lm() requires a separate data argument:
dat <- iris[, 1:4]
synthetic(dat,
          model_expression = lm(.outcome ~ .,
                                data = data.frame(.outcome = y,
                                xsynth)),
          predict_expression = predict(model, newdata = xsynth))

# Example using bootstrapping:
synthetic(iris,
          model_expression = NULL,
          predict_expression = sample(y, size = length(y), replace = TRUE))

# Example with missing data, no imputation
iris_missings <- iris
for(i in 1:10){
  iris_missings[sample.int(nrow(iris_missings), 1, replace = TRUE),
                sample.int(ncol(iris_missings), 1, replace = TRUE)] <- NA
}
iris_miss_syn <- synthetic(iris_missings)

# Example with missing data, imputation by median/mode substitution
# First, define a simple function for median/mode substitution:
imp_fun <- function(x){
  if(is.data.frame(x)){
    return(data.frame(sapply(x, imp_fun)))
  } else {
    out <- x
    if(inherits(x, "numeric")){
      out[is.na(out)] <- median(x[!is.na(out)])
    } else {
      out[is.na(out)] <- names(sort(table(out), decreasing = TRUE))[1]
    }
    out
  }
}

# Then, call synthetic() with this function as missingness_expression:
iris_miss_syn <- synthetic(iris_missings,
                           missingness_expression = imp_fun(data))
```

worcs_badge   *Add WORCS badge to README.md*

Description

Evaluates whether a project meets the criteria of the WORCS checklist (see worcs_checklist), and adds a badge to the project's README.md.

Usage

```
worcs_badge(
  path = ".",
  update_readme = "README.md",
  update_csv = "checklist.csv"
)
```

Arguments

`path`	Character. This can either be the path to a WORCS project folder (a project with a `.worcs` file), or the path to a `checklist.csv` file. The latter is useful if you want to evaluate a manually updated checklist file. Default: '.' (path to current directory).
`update_readme`	Character. Path to the `README.md` file to add the badge to. Default: 'README.md'. Set to `NULL` to avoid updating the `README.md` file.
`update_csv`	Character. Path to the `README.md` file to add the badge to. Default: 'checklist.csv'. Set to `NULL` to avoid updating the `checklist.csv` file.

Value

No return value. This function is called for its side effects.

Examples

```
example_dir <- file.path(tempdir(), "badge")
dir.create(example_dir)
write("a", file.path(example_dir, ".worcs"))
worcs_badge(path = example_dir,
update_readme = NULL)
```

`worcs_checklist` *WORCS checklist*

Description

This checklist can be used to see whether a project adheres to the principles of open reproducible code in science, as set out in the WORCS paper.

Usage

```
data(worcs_checklist)
```

Format

A data frame with 15 rows and 5 variables.

Details

category	`factor`	Category of the checklist element.
name	`factor`	Name of the checklist element.
description	`factor`	What are the requirements to claim that this checklist element is met?
importance	`factor`	Whether the checklist element is essential to obtain a green 'open science' badge, or optional.
check	`logical`	Whether the criterion is checked automatically by `worcs_badge`.

References

Van Lissa, C. J., Brandmaier, A. M., Brinkman, L., Lamprecht, A., Peikert, A., , Struiksma, M. E., & Vreede, B. (2020) <doi:10.17605/OSF.IO/ZCVBS>.

`worcs_project` *Create new WORCS project*

Description

Creates a new 'worcs' project. This function is invoked by the 'RStudio' project template manager, but can also be called directly to create a WORCS project through syntax or the console.

Usage

```
worcs_project(
  path = "worcs_project",
  manuscript = "APA6",
  preregistration = "COS",
  add_license = "CC_BY_4.0",
  use_renv = TRUE,
  remote_repo = "git@",
  verbose = TRUE,
  ...
)
```

Arguments

`path`	Character, indicating the directory in which to create the 'worcs' project. Default: 'worcs_project'.
`manuscript`	Character, indicating what template to use for the 'R Markdown' manuscript. Default: 'APA6'. Available choices include: `"APA6","github_document","None","ams_article","a` For more information, see `add_manuscript`.
`preregistration`	Character, indicating what template to use for the preregistration. Default: 'COS'. Available choices include: `"COS","VantVeer","Brandt","AsPredicted","PSS","Secondary","Non` For more information, see `add_preregistration`.

`add_license`	Character, indicating what license to include. Default: 'CC_BY_4.0'. Available options include: `"CC_BY_4.0","CC_BY-SA_4.0","CC_BY-NC_4.0","CC_BY-NC-SA_4.0","CC_BY-ND_` For more information, see <https://creativecommons.org/licenses/>.
`use_renv`	Logical, indicating whether or not to use 'renv' to make the project reproducible. Default: TRUE. See `init`.
`remote_repo`	Character, 'SSH' address of the remote repository for this project. This link should have the form `git@[...].git`. If a valid remote repository link is provided, a commit will be made containing the 'README.md' file, and will be pushed to the remote repository. Default: 'git@'.
`verbose`	Logical. Whether or not to print messages to the console during project creation. Default: TRUE
`...`	Additional arguments passed to and from functions.

Value

No return value. This function is called for its side effects.

Examples

```
the_test <- "worcs_template"
old_wd <- getwd()
dir.create(file.path(tempdir(), the_test))
do.call(git_user, worcs:::get_user())
worcs_project(file.path(tempdir(), the_test, "worcs_project"),
              manuscript = "github_document",
              preregistration = "None",
              add_license = "None",
              use_renv = FALSE,
              remote_repo = "git@")
setwd(old_wd)
unlink(file.path(tempdir(), the_test))
```

Index