Package: reschola (via r-universe)

September 13, 2024

Title The Schola Empirica Package

Version 0.5.6

Description A collection of utilies, themes and templates for data analysis at Schola Empirica.

License MIT + file LICENSE

URL https://github.com/scholaempirica/reschola

BugReports https://github.com/scholaempirica/reschola/issues

Imports bookdown, dplyr, extrafont, forcats, fs, ggplot2, ggtext, googledrive, grDevices, here, hrbrthemes, htmltools, httr, jsonlite, knitr, lifecycle, magrittr, officer, png, purrr (>= 1.0.0), RColorBrewer, readr, rlang (>= 0.4.11), rmarkdown, rstudioapi, scales, stringr, tibble, tidyr, usethis, utils

Suggests testthat (>= 3.0.0), spelling, roxygen2

VignetteBuilder knitr

Encoding UTF-8

Language en-US

Roxygen list(markdown = TRUE)

RoxygenNote 7.2.3

Config/testthat/edition 3

Repository https://scholaempirica.r-universe.dev

RemoteUrl https://github.com/scholaempirica/reschola

RemoteRef HEAD

RemoteSha bdf61d6c8d760d61415757cdf3138a42ee2a829c

Contents

as_czech_date *Make Date of Class* czech_date

Description

Appends the czech_date class attribute to the input object. Date of class czech_date is printed as a date in long format with correct Czech grammatical case (see Details and Grammatical cases section below).

Usage

```
as_czech_date(date, case = "genitive")
```

Arguments

date	*date or date-like object* to parse.
case	*character*, either "nominative", "locative" or "genitive" (default) or any unambiguous abbreviation of these.

Details

The grammatical case *should* be specified as and argument to print() method, but for convenience, you can predefine it in as_czech_date call directly. It is then stored as an attribute, later grabbed by the print method.

Note that as opposed to other date formating functions in R, as_date_czech trims leading zeros.

Value

Same as input, but with class czech_date and attribute gramm_case.

Grammatical cases

Three grammatical cases are supported:

- *nominative* – native form, i.e. "leden" in Czech
- *locative* – "in ...", i.e. "v lednu" in Czech
- *genitive* – "the 'nth' of ...", i.e. "5. ledna" in Czech

Czech months listed by case are available in .czech_months.

Examples

```
Sys.time() %>% as_czech_date()

# in "nominative" grammatical case (note the abbreviation)
Sys.time() %>% as_czech_date("nom")
```

`compile_and_open` *Safely Compile RMarkdown Document(s)*

Description

Saves everything opened in a current project, renders (compiles) chosen `.Rmd` document(s), and opens the resulting file if everything went OK.

Usage

```
compile_and_open(
  input,
  output_dir = "reports-output",
  open_on_success = TRUE,
  ...
)
```

Arguments

`input`	path to source `.Rmd` file
`output_dir`	output directory, default to "reports-output"
`open_on_success`	if compilation went successul, open the result, default to TRUE
`...`	Arguments passed on to `rmarkdown::render`

`output_format` The R Markdown output format to convert to. The option `"all"` will render all formats defined within the file. The option can be the name of a format (e.g. `"html_document"`) and that will render the document to that single format. One can also use a vector of format names to render to multiple formats. Alternatively, you can pass an output format object (e.g. `html_document()`). If using `NULL` then the output format is the first one defined in the YAML frontmatter in the input file (this defaults to HTML if no format is specified there). If you pass an output format object to `output_format`, the options specified in the YAML header or `_output.yml` will be ignored and you must explicitly set all the options you want when you construct the object. If you pass a string, the output format will use the output parameters in the YAML header or `_output.yml`.

`output_file` The name of the output file. If using `NULL` then the output filename will be based on filename for the input file. If a filename is provided, a path to the output file can also be provided. Note that the `output_dir` option allows for specifying the output file path as well, however, if also specifying the path, the directory must exist. If `output_file` is specified but does not have a file extension, an extension will be automatically added according to the output format. To avoid the automatic file extension, put the `output_file` value in `I()`, e.g., `I('my-output')`.

`output_options` List of output options that can override the options specified in metadata (e.g. could be used to force `self_contained` or `mathjax = "local"`). Note that this is only valid when the output format is read from metadata (i.e. not a custom format object passed to `output_format`).

`output_yaml` Paths to YAML files specifying output formats and their configurations. The first existing one is used. If none are found, then the function searches YAML files specified to the `output_yaml` top-level parameter in the YAML front matter, _output.yml or _output.yaml, and then uses the first existing one.

`intermediates_dir` Intermediate files directory. If a path is specified then intermediate files will be written to that path. If `NULL`, intermediate files are written to the same directory as the input file.

`knit_root_dir` The working directory in which to knit the document; uses knitr's `root.dir` knit option. If `NULL` then the behavior will follow the knitr default, which is to use the parent directory of the document.

`runtime` The runtime target for rendering. The `static` option produces output intended for static files; `shiny` produces output suitable for use in a Shiny document (see `run`). The default, `auto`, allows the `runtime` target specified in the YAML metadata to take precedence, and renders for a `static` runtime target otherwise.

`clean` Using `TRUE` will clean intermediate files that are created during rendering.

`params` A list of named parameters that override custom params specified within the YAML front-matter (e.g. specifying a dataset to read or a date range to confine output to). Pass `"ask"` to start an application that helps guide parameter configuration.

`knit_meta` (This option is reserved for expert use.) Metadata generated by **knitr**.

`envir` The environment in which the code chunks are to be evaluated during knitting (can use `new.env()` to guarantee an empty new environment).

`run_pandoc` An option for whether to run pandoc to convert Markdown output.

`quiet` An option to suppress printing during rendering from knitr, pandoc command line and others. To only suppress printing of the last "Output created: " message, you can set `rmarkdown.render.message` to `FALSE`

`encoding` Ignored. The encoding is always assumed to be UTF-8.

Value

No return value. Called for side effects.

Examples

```
## Not run:
# for which REDIZOs should the reports be compiled?
red_izos <- c("600000001", "600000002", "600000003")

purrr::map(
  red_izos,
```

```
  ~ compile_and_open("05_ucitele-reditele_ms.Rmd",
    open_on_success = FALSE,
    output_dir = here("reports-output", "ucitele-reditele-ms"),
    output_file = paste0("02_ucitele-reditele-ms_", .x, ".docx"),
    params = list(redizo = .x)
  )
)

## End(Not run)
```

`copy_schola_template` *Copy default Schola template into project directory*

Description

Copy default Schola template into project directory

Usage

```
copy_schola_template(format = "pdf", path = proj_get(), ...)
```

Arguments

`format`	*Character*, format which to look for. Defaults to `pdf`.
`path`	*Character*, path to copy to. Defaults to the current project root.
`...`	Arguments passed on to `base::file.copy` `overwrite` logical; should existing destination files be overwritten? `recursive` logical. If `to` is a directory, should directories in `from` be copied (and their contents)? (Like `cp -R` on POSIX OSes.) `copy.mode` logical: should file permission bits be copied where possible? `copy.date` logical: should file dates be preserved where possible? See `Sys.setFileTime`.

Value

No return value, called for side effect.

Author(s)

Jan Netik

See Also

Other Report templates and formats: `open_schola_template()`, `schola_pdf()`, `schola_word2()`, `schola_word()`

Examples

```
## Not run:
copy_schola_template()

## End(Not run)
```

---

czech_date_interval *Czech Date Interval*

Description

Returns the most space-efficient and at the same time grammatically correct interval of two Czech dates. When both dates are the same, only one is outputted. The function ensures that the interval is not negative (i.e., `start <= end`), otherwise, it is reversed.

Usage

```
czech_date_interval(start, end)
```

Arguments

`start`	*Date of date-like object*, start date or left boundary of an interval.
`end`	*Date of date-like object*, end date or right boundary of an interval.

Value

Character

Examples

```
czech_date_interval("2020-01-24", "2020-01-03") # note the argument order
```

---

dict_from_data *Get item code-label dictionary from data*

Description

Item labels have to be stored as variable attribute `label`. Exported by default by `ls_export()`, but many data wrangling operations strip them out, unfortunately.

Usage

```
dict_from_data(.data)
```

Arguments

`.data`	data.frame or tibble

Value

named character vector of items' labels, if no label found, NULL is introduced and is omitted in the output

---

`draft_pdf` *Create a* `.Rmd` *draft using Schola templates*

Description

Shortcut function to create a new `.Rmd` file using Schola standard templates and open it for editing.

Usage

```
draft_pdf(name = "pdf_draft", open = TRUE)

draft_word(name = "word_draft", open = TRUE)
```

Arguments

`name`	*character*, name to use for new file. With or without file extension.
`open`	*logical*, whether to open file for editing, defaults to TRUE.

Value

Path to the created file (invisibly).

See Also

Other Workflow helpers: `gd_download_folder()`, `manage_docx_header_logos()`

Examples

```
## Not run:
draft_pdf("best_report")

## End(Not run)
```

extract_schola_barplot_info *Extract detailed info from Schola barplot*

Description

Note that item labels are automatically derived using the labeller from given `schola_barplot` result

Usage

```
extract_schola_barplot_info(plot)
```

Arguments

`plot`	result of schola_barplot() call

Value

tibble with plot data

Examples

```
"TODO"
```

fct_nanify *NAnify factor level*

Description

The inverse of `fct_explicit_na()`. Turns the `level` into proper `NA`. **Retains the level.**

Usage

```
fct_nanify(f, level, negate = FALSE, ignore_case = TRUE)
```

Arguments

`f`	*factor* to work on
`level`	*character*, regular expression matching the desired level
`negate`	*logical*, whether to return non-matching elements. Defaults to `FALSE`.
`ignore_case`	*logical*, ignore case when matching? Defaults to `TRUE`.

Value

factor with NA-substituted level.

Examples

```
f <- factor(c("a", "b", "c", "nanify"))
fct_nanify(f, "nanify")
```

`flush_axis` *A shorcut for making axis text flush with axis*

Description

A shorcut for making axis text flush with axis

Usage

```
flush_axis
```

Format

An object of class `numeric` of length 4.

See Also

Other Making charts: `plot_lollipop()`, `prepare_lollipop_data()`, `schola_barplot()`, `theme_schola()`

Examples

```
library(ggplot2)
ggplot(mpg) +
  geom_bar(aes(y = class)) +
  scale_x_continuous(expand = flush_axis)
```

`formatscz` *Czech formatted scales for ggplot2*

Description

`label_percent_cz()` returns a formatter that outputs percent labels with a " %" suffix and a decimal comma. `label_number_cz()` uses space as thousand separator and decimal comma. Use these when you need to format labels on something other than X and Y axes

Usage

```
label_percent_cz(
  accuracy = NULL,
  scale = 100,
  prefix = "",
  suffix = " %",
  big.mark = " ",
  decimal.mark = ",",
  trim = TRUE,
  ...
)

label_number_cz(
  accuracy = NULL,
  scale = 1,
  prefix = "",
  suffix = "",
  big.mark = " ",
  decimal.mark = ",",
  trim = TRUE,
  ...
)
```

Arguments

accuracy	A number to round to. Use (e.g.) `0.01` to show 2 decimal places of precision. If `NULL`, the default, uses a heuristic that should ensure breaks have the minimum number of digits needed to show the difference between adjacent values. Applied to rescaled data.
scale	A scaling factor: `x` will be multiplied by `scale` before formatting. This is useful if the underlying data is very small or very large.
prefix	Additional text to display before the number. The suffix is applied to absolute value before `style_positive` and `style_negative` are processed so that `prefix = "$"` will yield (e.g.) `-$1` and `($1)`.
suffix	Additional text to display after the number.
big.mark	Character used between every 3 digits to separate thousands.
decimal.mark	The character to be used to indicate the numeric decimal point.
trim	Logical, if `FALSE`, values are right-justified to a common width (see `base::format()`).
...	Other arguments passed on to `base::format()`.

Examples

```
library(reschola)
library(ggplot2)
ggplot(mpg, aes(hwy, cty)) +
  geom_point(aes(colour = cty / max(cty), size = hwy * 100)) +
```

```
  theme_schola(family = "sans", title_family = "sans", gridlines = "scatter") +
  scale_color_binned(labels = label_percent_cz()) +
  scale_size_binned(labels = label_number_cz())
```

`gd_download_folder` *Download files from Google Drive folder*

Description

Downloads all downloadable files from given Google Drive folder and saves them to specified local directory. Files will have the same names. Use `googledrive::drive_download()` for downloading individual files.

Usage

```
gd_download_folder(
  folder_url = gd_get_proj(),
  dest_dir = "data/input",
  files_from_subfolders = FALSE,
  overwrite = TRUE
)
```

Arguments

`folder_url`	Folder URL.
`dest_dir`	Local URL in which to store files.
`files_from_subfolders`	Whether to download also files from subdirectories. Defaults to `TRUE`. See Note.
`overwrite`	Whether to overwrite if file of same name exists.

Value

vector of paths to downloaded files

Note

As it would be very difficult to mirror the folder exactly, if `files_from_subfolders` is set to `TRUE`, files from subfolders will be saved in the same directory, not in the respective subdirectories. Bear this in mind when naming files in GDrive subfolders so as to avoid naming conflicts.

See Also

Other Workflow helpers: `draft_pdf()`, `manage_docx_header_logos()`

Examples

```
## Not run:
gd_url <- "https://drive.google.com/drive/folders/1bCyR_VKAP_43NEujqisjN77hANnMKfHZ"
gd_download_folder(
  folder_url = gd_url,
  files_from_subfolders = T, overwrite = T
)

## End(Not run)
```

---

gd_get_proj *Get current reschola project Google Drive URL ID*

Description

Gets a hidden object `.gd_proj_url` (by default) created at project "startup". See the details below for reschola projects created prior reschola version 0.4.0.

Usage

```
gd_get_proj(url_object = ".gd_proj_url")
```

Arguments

`url_object`	*character*, name of the object URL is stored in. `.gd_proj_url` by default.

Value

A character vector of class `drive_id`.

Legacy usage prior to {reschola} version 0.4.0

Call `usethis::edit_r_profile(scope = "project")` and write (note the dot prefix):

```
.gd_proj_url <- "your_google_drive_url"
```

Note that **you have to restart your R session to apply any changes**. Note also that **the URL cannot contain any query** (parts with a leading question mark), i.e. `?usp=sharing`.

gd_upload_file *Upload a file to project's Google Drive*

Description

Upload a file to project's Google Drive

Usage

```
gd_upload_file(file, dir = gd_get_proj())
```

Arguments

file	*character*, path to the file.
dir	Directory at Google Drive, defaults to current reschola project directory picked up by `gd_get_proj()`.

Value

An object of class `dribble`, a tibble with one row per file.

get_input_data *Quick access to data in standard* reschola *project*

Description

Note that file paths are handled with `here()` already.

Usage

```
get_input_data(file)

get_intermediate_data(file)

get_processed_data(file)

write_input_data(.data, file)

write_intermediate_data(.data, file)

write_processed_data(.data, file)
```

Arguments

file	*character*, file name to get or write to, `.rds` is appended automatically if not already provided.
.data	*data object* to save as RDS.

Value

- `get_*` returns object(s) in the `.rds` being read
- `write_*` returns a `fs_path` path of `.rds` file being created invisibly

---

get_labs_df *Get labels of variables*

Description

Get labels of variables

Usage

```
get_labs_df(.data)
```

Arguments

`.data`	tibble or data.frame

Value

tibble with variables and its labels (as list-column)

Examples

```
# make labels for iris dataset, labels will be colnames
# with dot replaced for whitespace
iris_with_labs <- as.data.frame(mapply(function(x, y) {
  attr(x, "label") <- y
  return(x)
}, iris, gsub("\\.", " ", colnames(iris)), SIMPLIFY = FALSE))

get_labs_df(iris_with_labs)
```

---

import_fonts *Import Ubuntu fonts for use in charts and in the PDF reports*

Description

[Deprecated]

The function relies on {extrafont} package that comes with briken dependencies at the moment. An experimental `register_reschola_fonts()` is proposed.

Usage

```
import_fonts()
```

Details

This is an analogue of `hrbrthemes::import_roboto_condensed()`.

There is an option `reschola.loadfonts` which – if set to `TRUE` – will call `extrafont::loadfonts()` to register non-core fonts with R PDF & PostScript devices. If you are running under Windows, the package calls the same function to register non-core fonts with the Windows graphics device.

Note

If you install the fonts just for the current user (via right-click and Install), they will probably **not be discoverable** by the `fontspec` LaTeX package that is used for PDF report typesetting!

Source

Fonts are licensed under the Ubuntu Font License.

See Also

Other Font helpers and shortcuts: `install_reschola_fonts()`, `register_reschola_fonts()`, `use_reschola_fonts()`

`install_reschola_fonts`

Install reschola fonts on your computer

Description

Just a simple wizard.

Usage

```
install_reschola_fonts()
```

Value

Side effects.

See Also

Other Font helpers and shortcuts: `import_fonts()`, `register_reschola_fonts()`, `use_reschola_fonts()`

`ls_add_participants` *Add Participant(s) to the Survey*

Description

The function takes a `tibble` (or any object that is internally represented as a (named) `list` by `R`) of participant(s) data and adds them to the LimeSurvey participant database of the selected survey.

Usage

```
ls_add_participants(survey_id, part_data, create_token = TRUE)
```

Arguments

`survey_id`	*integer*, ID of the survey (as found with `ls_surveys`, e.g.).
`part_data`	*tibble / data.frame / list*, object with participant(s) data, i.e., `firstname`, `lastname`, `email` etc.
`create_token`	*logical*, whether to create token outright. Defaults to `TRUE`).

Details

Generally, your `part_data` object have to contain three variables: `firstname`, `lastname`, and `email`. That is something like a bare minimum, but you may add any attribute recognized by LimeSurvey – even custom attributes like `attribute_1` or so. For the human-readable list of custom attributes being held in the LimeSurvey participant database of the selected survey, use `ls_get_attrs()`. However, do not use the "semantic" form, as `ls_add_participants()` recognizes only raw, i.e. `attribute_1` notation.

Value

Called for a side effect, but returns the inserted data including additional new information like the token string.

See Also

Other LimeSurvey functions: `ls_call()`, `ls_export()`, `ls_get_attrs()`, `ls_invite()`, `ls_login()`, `ls_participants()`, `ls_responses()`, `ls_set_participant_properties()`, `ls_surveys()`

Examples

```
## Not run:
# create participant table
part_data <- tibble(
  firstname = "John",
  lastname = "Doe",
  email = "john@example.com",
  language = "cs",
  attribute_1 = "Example School"
```

```
)

# insert participant into the LimeSurvey database
ls_add_participants(123456, part_data)

# check if OK
ls_participants(123456)

## End(Not run)
```

`ls_call` *Call LimeSurvey API Directly*

Description

General function used internally by every other `ls_` fellow. Useful when you want something special that is not (yet) implemented in `reschola`.

Usage

```
ls_call(method, params = list())
```

Arguments

`method`	*character*, a method supported by LimeSurvey API.
`params`	*list*, arguments of the method. Need to be in order stated in documentation. **Note that** `sSessionKey` **auth credentials are already provided as the first entry of** `params` **list.**

Details

The list of available LimeSurvey API calls briefly documents the functionality provided. You must strictly adhere to the arguments positions, as they are passed as an array.

Value

A tibble, or raw object if server response cannot be reasonably coerced to a tibble.

See Also

Other LimeSurvey functions: `ls_add_participants()`, `ls_export()`, `ls_get_attrs()`, `ls_invite()`, `ls_login()`, `ls_participants()`, `ls_responses()`, `ls_set_participant_properties()`, `ls_surveys()`

Examples

```
## Not run:
ls_call("get_survey_properties", params = list(iSurveyID = 123456))

## End(Not run)
```

`ls_check_attributes` *Check for Illegal Attributes*

Description

Checks for illegal LS attributes and raises an error when any found.

Usage

```
ls_check_attributes(attributes)
```

Arguments

`attributes`	vector of attributes

`ls_export` *Export Responses with Participants Attached*

Description

Export Responses with Participants Attached

Usage

```
ls_export(
  survey_id,
  attributes = TRUE,
  clean_labels = TRUE,
  n_participants = 999,
  lang = "cs",
  part = "all",
  only_unused_tokens = FALSE,
  join_by = "token",
  ...
)
```

Arguments

survey_id	*integer*, ID of the survey (as found with ls_surveys, e.g.).
attributes	*logical* try to recover all attributes (default to TRUE), or *character vector* specifying requested attributes.
clean_labels	*logical*, whether to clean labels of subquestions from repeating parts. Defaults to TRUE.
n_participants	*integer*, the number of participants to list, defualt to 999.
lang	*character*, ISO 639 language code, default to cs.
part	*character*, completion status, either complete, incomplete or all (the default).
only_unused_tokens	*logical*, should only the unused tokens be listed? Default to FALSE.
join_by	*character*, the joining variable present in both responses and participants tibbles. Default to token. Pass NULL to join by common variables.
...	Arguments passed on to dplyr::left_join copy If x and y are not from the same data source, and copy is TRUE, then y will be copied into the same src as x. This allows you to join tables across srcs, but it is a potentially expensive operation so you must opt into it. suffix If there are non-joined duplicate variables in x and y, these suffixes will be added to the output to disambiguate them. Should be a character vector of length 2. keep Should the join keys from both x and y be preserved in the output? • If NULL, the default, joins on equality retain only the keys from x, while joins on inequality retain the keys from both inputs. • If TRUE, all keys from both inputs are retained. • If FALSE, only keys from x are retained. For right and full joins, the data in key columns corresponding to rows that only exist in y are merged into the key columns from x. Can't be used when joining on inequality conditions.

Value

A tibble.

See Also

Other LimeSurvey functions: ls_add_participants(), ls_call(), ls_get_attrs(), ls_invite(), ls_login(), ls_participants(), ls_responses(), ls_set_participant_properties(), ls_surveys()

Examples

```
## Not run:
ls_export(123456)

## End(Not run)
```

ls_get_attrs *Get Survey Attributes in Semantic Form*

Description

A survey can comprise one or more custom attributes useful for encoding participant characteristics directly within participants table. However, LimeSurvey refers to them as `attribute_x` (where `x` is the attribute position) which is not useful at all. The function aims to resolve this issue by returning so-called "semantic" form of attributes with human-readable description.

Usage

```
ls_get_attrs(survey_id)
```

Arguments

`survey_id`	*integer*, ID of the survey (as found, e.g., with `ls_surveys()`).

Value

A character vector of "semantic" attributes with names denoting "raw" attributes used internally by LimeSurvey.

See Also

Other LimeSurvey functions: `ls_add_participants()`, `ls_call()`, `ls_export()`, `ls_invite()`, `ls_login()`, `ls_participants()`, `ls_responses()`, `ls_set_participant_properties()`, `ls_surveys()`

Examples

```
## Not run:
ls_get_attrs(123456)

## End(Not run)
```

ls_invite *Invite Participant(s)*

Description

Send an email with a link to a survey to the particular participant(s). Uses email template specified in the LimeSurvey web interface. Please read the section *"On errors and messages from the API server"* of this documentation page before use.

Usage

```
ls_invite(survey_id, tid, uninvited_only = TRUE)
```

Arguments

survey_id	*integer*, ID of the survey (as found with `ls_surveys()`, e.g.).
tid	*integer(s)*, one ore more token IDs (**not tokens!**) from participant database to invite. Use `ls_participants()` to get the `tid`s.
uninvited_only	*logical*, if `TRUE`, send invitation for participants that have not been invited yet (default). If `FALSE`, send an invite even if already sent.

Details

LimeSurvey allows you to send so-called invitation to a participant, meaning he or she will get an email containing a link with his or her unique access token. If you wish to send the invitation even if it has been already sent, use `uninvited_only = FALSE`.

Value

Called for a side effect. Returns a message from the server.

On errors and messages from the API server

Note that the function passes on any messages from the LimeSurvey API server. As usual with LimeSurvey, many things are erroneous, buggy or does not make sense. In this case, a sign of a successful invitation is something like *"-1 left to send"* (where *"-1"* denotes the number of invitations sent).

Another message you may see is *"Error: No candidate tokens"*, which possibly means that the `tid`s you use are not present in the survey of concern. However, it can also indicate that the invitation has been already sent to the `tid`s and you have to use `uninvited_only = FALSE` to proceed.

Note that when you add an email entry that has not a proper email format, no participants are added and tibble with `errors$email` list-column is returned.

See Also

Other LimeSurvey functions: `ls_add_participants()`, `ls_call()`, `ls_export()`, `ls_get_attrs()`, `ls_login()`, `ls_participants()`, `ls_responses()`, `ls_set_participant_properties()`, `ls_surveys()`

Examples

```
## Not run:
ls_invite(123456, 18)

## End(Not run)
```

`ls_login` *Login to LimeSurvey API*

Description

Obtains XML-RPC/JSON-RPC session key and stores it in dedicated environment for further use by fellow `ls_` functions.

Usage

```
ls_login(
  api_url =
   "https://dotazniky.scholaempirica.org/limesurvey/index.php/admin/remotecontrol"
)
```

Arguments

`api_url`	*character*, URL of the API endpoint, default to SCHOLA EMPIRICA's LimeSurvey API.

Details

By default, https://dotazniky.scholaempirica.org/ is used as the LimeSurvey server prividing the API. The credentials used for user authentication are obtained through interactive prompts, mainly for security reasons. The function tries to obtain the credentials from the system environment variable table first. If none found, the user is asked for them and is provided with guidance for permanent credential storage.

Value

No "explicit" return value, but assigns the session key to a dedicated environment.

See Also

Other LimeSurvey functions: `ls_add_participants()`, `ls_call()`, `ls_export()`, `ls_get_attrs()`, `ls_invite()`, `ls_participants()`, `ls_responses()`, `ls_set_participant_properties()`, `ls_surveys()`

Examples

```
## Not run:
ls_login()

## End(Not run)
```

`ls_participants` *List Participants*

Description

Fetches participant list of the solicited survey. The function tries to retrieve as many attributes as possible and translate them to their "semantic" version by default. You can also provide character vector of requested attributes, but not in the "semantic" form (use, e.g., `attribute_1` or `usesleft`).

Usage

```
ls_participants(
  survey_id,
  attributes = TRUE,
  n_participants = 999,
  only_unused_tokens = FALSE,
  translate_attrs = TRUE
)
```

Arguments

`survey_id`	*integer*, ID of the survey (as found, e.g., with `ls_surveys()`).
`attributes`	*logical* try to recover all attributes (default to TRUE), or *character vector* specifying requested attributes.
`n_participants`	*integer*, the number of participants to list, defualt to 999.
`only_unused_tokens`	*logical*, should only the unused tokens be listed? Default to FALSE.
`translate_attrs`	*logical*, should the custom attributes be "translated" to "semantic" version? Default to TRUE.

Value

A tibble, or raw object if server response cannot be reasonably coerced to a tibble.

See Also

Other LimeSurvey functions: `ls_add_participants()`, `ls_call()`, `ls_export()`, `ls_get_attrs()`, `ls_invite()`, `ls_login()`, `ls_responses()`, `ls_set_participant_properties()`, `ls_surveys()`

Examples

```
## Not run:
ls_participants(123456, attributes = c("usesleft"))

## End(Not run)
```

`ls_responses` *Export Responses*

Description

Fetches responses and applies so-called "R-syntax" transformation script from LimeSurvey pertaining factor levels and items labels (those are readily available in RStudio data frame preview and can be extracted using `attr(.data, "label")`). By default, the function attempts to "clean" the labels (`clean_labels` argument), keeping only the content inside `[...]`, if there are any.

Usage

```
ls_responses(survey_id, clean_labels = TRUE, lang = "cs", part = "all", ...)
```

Arguments

`survey_id`	*integer*, ID of the survey (as found with `ls_surveys`, e.g.).
`clean_labels`	*logical*, whether to clean labels of subquestions from repeating parts. Defaults to `TRUE`.
`lang`	*character*, ISO 639 language code, default to `cs`.
`part`	*character*, completion status, either `complete`, `incomplete` or `all` (the default).
`...`	*other named arguments* used by "export_responses" method. Use at your own risk.

Value

A tibble, or raw object if server response cannot be reasonably coerced to a tibble.

See Also

Other LimeSurvey functions: `ls_add_participants()`, `ls_call()`, `ls_export()`, `ls_get_attrs()`, `ls_invite()`, `ls_login()`, `ls_participants()`, `ls_set_participant_properties()`, `ls_surveys()`

Examples

```
## Not run:
ls_responses(123456)

## End(Not run)
```

`ls_set_participant_properties` *Set or Edit Attribute(s) of an Participant*

Description

Set or Edit Attribute(s) of an Participant

Usage

```
ls_set_participant_properties(survey_id, participant, ...)
```

Arguments

`survey_id`	*integer*, ID of the survey (as found with `ls_surveys()`, e.g.).
`participant`	*integer* or *list*, **one** token ID (**not token!**) from participant database. Use `ls_participants()` to get the `tid`. Another option is to pass a list of one ore more participant properties, i.e. `list(lastname = "Doe")`
`...`	attributes in the form `attribute_name = attribute_value`.

Value

A tibble with the participant row just edited.

See Also

Other LimeSurvey functions: `ls_add_participants()`, `ls_call()`, `ls_export()`, `ls_get_attrs()`, `ls_invite()`, `ls_login()`, `ls_participants()`, `ls_responses()`, `ls_surveys()`

Examples

```
## Not run:
ls_set_participant_properties(123456,
  participant = 18, email = "new@email.cz",
  attribute_1 = 600123456
)

## End(Not run)
```

ls_surveys *List All Surveys at the Server*

Description

Lists every single survey found at the server you are logged in. Useful to gain information about the surveys IDs etc.

Usage

```
ls_surveys()
```

Value

A tibble.

See Also

Other LimeSurvey functions: `ls_add_participants()`, `ls_call()`, `ls_export()`, `ls_get_attrs()`, `ls_invite()`, `ls_login()`, `ls_participants()`, `ls_responses()`, `ls_set_participant_properties()`

Examples

```
## Not run:
ls_surveys()

## End(Not run)
```

manage_docx_header_logos *Add or replace logo in the header of a Word document*

Description

Takes an existing Word document created using the `reschola` templates and adds (by default, or replaces if `action = "replace_schola"`) the logo with the image file you point it to.

Usage

```
manage_docx_header_logos(
  docx_path,
  png_logo_path,
  action = c("add_client", "replace_schola"),
  height = NULL
)
```

Arguments

docx_path	The Word document in which to replace logos. Must contain the bookmarks schola_logo and client_logo in the header (files created from reschola templates do by default.)
png_logo_path	a PNG file which will be added/used as replacement
action	whether to add new logo on the right ("add_client") or replace default Schola logo on the left ("replace_schola")
height	height of the new logo in the resulting document, in cm. By default, uses the height of the existing Schola logo in the header.

Value

invisibly returns the name of the new Word doc, which is same as the input Word doc, with an an added _addedlogo suffix.

Note

This requires specific bookmarks in the header of the input document. This is taken from the skeleton.docx template in the template components. If you overwrite the header in the input document, this function will not work.

See Also

Other Workflow helpers: draft_pdf(), gd_download_folder()

Examples

```
## Not run:
library(reschola)
manage_docx_header_logos("draft.docx",
  png_logo_path = "logos/newlogo.png",
  action = "add_client"
)
manage_docx_header_logos("draft.docx",
  png_logo_path = "logos/newlogo.png",
  action = "replace_schola"
)

## End(Not run)
```

---

open_schola_template *Locate and open default Schola templates*

Description

Open the default Schola templates used by schola_word(), or schola_pdf() (specify with format argument). The templates are shipped with the package and are somewhat hidden in R library, so this auxiliary function can help you dig them up.

Usage

```
open_schola_template(format = "pdf")
```

Arguments

format	*Character*, format which to look for. Defaults to pdf.

Details

You can either edit the chosen template directly in its natural habitat (questionable short-term solution), or better – use "Save as" option and keep it and use it within the project directory.

Value

No return value, called for side effect.

Author(s)

Jan Netik

See Also

Other Report templates and formats: copy_schola_template(), schola_pdf(), schola_word2(), schola_word()

Examples

```
## Not run:
open_schola_template()

## End(Not run)
```

---

plot_lollipop *Plot lollipop*

Description

[Maturing]

Usage

```
plot_lollipop(
  plot_data,
  direction = "blue_larger",
  var_labels = waiver(),
  negative_label = NULL,
  positive_label = NULL,
  ref_label = NULL,
```

```
  xlab = "rozdíl mezi Vaší školou a ostatními",
  observations_alpha = 0.2
)
```

Arguments

plot_data	a
direction	a
var_labels	a
negative_label	a
positive_label	a
ref_label	a
xlab	a
observations_alpha	opacity of individual observations

Value

ggplot2 plot

See Also

Other Making charts: flush_axis, prepare_lollipop_data(), schola_barplot(), theme_schola()

---

prepare_lollipop_data *Prepare data for* plot_lollipop()

Description

[Maturing]

Usage

```
prepare_lollipop_data(.data, vars, group)
```

Arguments

.data	tibble or data.frame with variables
vars	variables to reshape from wide to long, uses tidyselect syntax
group	grouping variable (have to be logical!), usually denoting for which school the plot should be tailored for

Value

several tibbles inside list, intended for data-mining and for plotting

See Also

Other Making charts: flush_axis, plot_lollipop(), schola_barplot(), theme_schola()

---

recover_labs *Recover lost labels from same-structured tibble with labels*

Description

Many operations, such as rbind, cbind or its tidyverse analogues, strips out the variable labels. Use recover_labs() to bring them back from a tibble or data.frame where they are last present. The function attempts a few checks for new and original data compatibility. Note that the infix operator is available for a quick and self-explanatory usage.

Usage

```
recover_labs(new_data, orig_data)

new_data %labs_from% orig_data
```

Arguments

new_data	new dataframe that you want to recover the labs for
orig_data	original dataframe with variable labels present

Value

Tibble or data.frame with variable labels restored.

Examples

```
# make labels for iris dataset, labels will be colnames
# with dot replaced for whitespace
iris_with_labs <- as.data.frame(mapply(function(x, y) {
  attr(x, "label") <- y
  return(x)
}, iris, gsub("\\.", " ", colnames(iris)), SIMPLIFY = FALSE))

iris_with_recovered_labs <- recover_labs(iris, iris_with_labs)
iris_with_recovered_labs_infix <- iris %labs_from% iris_with_labs

# check
get_labs_df(iris_with_recovered_labs)
get_labs_df(iris_with_recovered_labs_infix)
```

`register_reschola_fonts` *Register reschola fonts on Windows*

Description

[Stable]

The function is only for Windows, where it tries to register font family provided with "Windows bitmap device". On other platforms, fonts *should* be readily available after installation. Note that even on Windows, you can instruct RStudio to use smarter graphics device, such as "AGG" or "Cairo" (*Tools* > *Global Options* > *General* > *Graphics* > *Backend*).

Usage

```
register_reschola_fonts(family = c("Ubuntu", "Ubuntu Condensed"))
```

Arguments

`family`	font family/families to register, default is reschola recommended.

Value

Called for side effects, but returns logical on process result invisibly.

See Also

Other Font helpers and shortcuts: `import_fonts()`, `install_reschola_fonts()`, `use_reschola_fonts()`

`remove_empty_at` *Remove rows empty only at specified columns*

Description

Drop rows that are completely empty *at specified variables*. Note `tidyr::drop_na()` have a similar usage, but it drops rows containing *any* missing values (so only complete observations are retained), not *all* missings. `remove_empty_at()` drops only those rows that have no non-missing data at the variables specified in `vars`.

Usage

```
remove_empty_at(.data, vars)
```

Arguments

`.data`	tibble or data.frame
`vars`	variables to check for empty values, supports tidyselect syntax

Value

tibble or data.frame without observations with all `vars` empty

Examples

```
airquality %>% remove_empty_at(c(Ozone, Solar.R))
```

`scalescz` *Continuous scales with Czech percent labels*

Description

This is exactly the same function as scale_*_continuous, just with different label defaults to save repetitively setting these parameters, except that the parameters which you would normally find inside the `scales::format_*()` functions are accessible directly in the `scale_[xy]_*_cz()` functions.

Usage

```
scale_x_percent_cz(
  name = waiver(),
  breaks = waiver(),
  minor_breaks = waiver(),
  guide = waiver(),
  n.breaks = NULL,
  labels,
  limits = NULL,
  expand = c(0.01, 0),
  oob = scales::censor,
  na.value = NA_real_,
  trans = "identity",
  position = "bottom",
  sec.axis = waiver(),
  accuracy = 1,
  scale = 100,
  prefix = "",
  suffix = " %",
  big.mark = " ",
  decimal.mark = ",",
  trim = TRUE,
  ...
)

scale_y_percent_cz(
  name = waiver(),
  breaks = waiver(),
```

```
  minor_breaks = waiver(),
  guide = waiver(),
  n.breaks = NULL,
  labels,
  limits = NULL,
  expand = c(0.01, 0),
  oob = scales::censor,
  na.value = NA_real_,
  trans = "identity",
  position = "left",
  sec.axis = waiver(),
  accuracy = 1,
  scale = 100,
  prefix = "",
  suffix = " %",
  big.mark = " ",
  decimal.mark = ",",
  trim = TRUE,
  ...
)

scale_x_number_cz(
  name = waiver(),
  breaks = waiver(),
  minor_breaks = waiver(),
  guide = waiver(),
  n.breaks = NULL,
  labels,
  limits = NULL,
  expand = c(0.01, 0),
  oob = scales::censor,
  na.value = NA_real_,
  trans = "identity",
  position = "bottom",
  sec.axis = waiver(),
  accuracy = 1,
  scale = 1,
  prefix = "",
  suffix = "",
  big.mark = " ",
  decimal.mark = ",",
  trim = TRUE,
  ...
)

scale_y_number_cz(
  name = waiver(),
  breaks = waiver(),
```

```
  minor_breaks = waiver(),
  guide = waiver(),
  n.breaks = NULL,
  labels,
  limits = NULL,
  expand = c(0.01, 0),
  oob = scales::censor,
  na.value = NA_real_,
  trans = "identity",
  position = "left",
  sec.axis = waiver(),
  accuracy = 1,
  scale = 1,
  prefix = "",
  suffix = "",
  big.mark = " ",
  decimal.mark = ",",
  trim = TRUE,
  ...
)
```

Arguments

name	The name of the scale. Used as axis or legend title. If `waiver()`, the default, the name of the scale is taken from the first mapping used for that aesthetic. If `NULL`, the legend title will be omitted.
breaks	One of: • `NULL` for no breaks • `waiver()` for the default breaks computed by the transformation object • A numeric vector of positions • A function that takes the limits as input and returns breaks as output
minor_breaks	One of: • `NULL` for no minor breaks • `waiver()` for the default breaks (one minor break between each major break) • A numeric vector of positions • A function that given the limits returns a vector of minor breaks.
guide	guide A function used to create a guide or its name. See `guides()` for more information.
n.breaks	An integer guiding the number of major breaks. The algorithm may choose a slightly different number to ensure nice break labels. Will only have an effect if `breaks = waiver()`. Use NULL to use the default number of breaks given by the transformation.
labels	Specifying overrides the default format (i.e. you really don't want to do that). `NULL` means no labels.

limits	A numeric vector of length two providing limits of the scale. Use NA to refer to the existing minimum or maximum.
expand	same as in ggplot2
oob	Function that handles limits outside of the scale limits (out of bounds). The default replaces out of bounds values with NA.
na.value	If na.translate = TRUE, what value aesthetic value should missing be displayed as? Does not apply to position scales where NA is always placed at the far right.
trans	Either the name of a transformation object, or the object itself. Built-in transformations include "asn", "atanh", "boxcox", "exp", "identity", "log", "log10", "log1p", "log2", "logit", "probability", "probit", "reciprocal", "reverse" and "sqrt".
position	The position of the axis. "left" or "right" for vertical scales, "top" or "bottom" for horizontal scales
sec.axis	specify a secondary axis
accuracy, scale, prefix, suffix, big.mark, decimal.mark, trim	See [scales::comma_format()] or [scales::percent_format()]
...	passed on to [scales::comma_format()] or [scales::percent_format()]

Examples

```
# ADD_EXAMPLES_HERE
library(reschola)
library(ggplot2)
ggplot(mpg, aes(hwy * 100, cty / max(cty))) +
  geom_point() +
  theme_schola(family = "sans", title_family = "sans", gridlines = "scatter") +
  scale_y_percent_cz() +
  scale_x_number_cz()
```

---

schola_barplot *Plot standard Schola likert-like barplot with groupwise comparison per items*

Description

[Maturing]

Usage

```
schola_barplot(
  .data,
  vars,
  group,
  dict = dict_from_data(.data),
```

```
  escape_level = "nevím",
  n_breaks = 11,
  desc = TRUE,
  labels = TRUE,
  min_label_width = 0.09,
  absolute_counts = TRUE,
  fill_cols = NULL,
  fill_labels = waiver(),
  facet_label_wrap = 115,
  reverse = FALSE,
  order_by = "chi-square differences",
  drop = FALSE,
  drop_na = TRUE,
  show.legend = TRUE,
  ...
)
```

Arguments

`.data`	data with items and group variable
`vars`	vector of items, supports `{tidyselect}` syntax (i.e, non-standard evaluation)
`group`	group variable used to split the results, have to be logical, where `TRUE` is gonna be considered as "focal" group and displayed as upper group Supports `{tidyselect}` syntax.
`dict`	item code-label dictionary, if none provided, those are derived from the data.
`escape_level`	*character*, level of item response considered as NA
`n_breaks`	number of breaks displayed at x-axis, outer labels are automatically aligned to face inward. Defaults to 11, which results in 10% wide breaks.
`desc`	sor items in descending order?
`labels`	draw labels?
`min_label_width`	smallest percentage (0-1) to display in the plot, proportions larger than this value are shown, smaller are not.
`absolute_counts`	draw labels and absolute counts in parentheses?
`fill_cols`	colors to be used for item categories, defaults to NULL, meaning standard RdYlBu palette will be used
`fill_labels`	character vector or function taking breaks and returning labels for fill aesthetic
`facet_label_wrap`	width of facet label to wrap
`reverse`	if TRUE, reverse colors
`order_by`	how to order the items. chi-square differences (default) computes chi-square test for every item and sort them by largest X2 statistic to smallest (if desc = TRUE)
`drop`	Drop unobserved levels form the legend? Defaults to `FALSE`. See ggplot2::discrete_scale for more details. To always show the legend key, make sure you have `show.legend` set to `TRUE` as well (as done by default).

drop_na	Drop NAs from every item (a.k.a. "pairwise")? Defaults to TRUE. Note that the number of observations per item may differ, because NA in one item does not mean the respondent row is discarded completely (listwise).
show.legend	logical. Should this layer be included in the legends? NA, the default, includes if any aesthetics are mapped. FALSE never includes, and TRUE always includes. It can also be a named logical vector to finely select the aesthetics to display.
...	Arguments passed on to fct_nanify negate *logical*, whether to return non-matching elements. Defaults to FALSE. ignore_case *logical*, ignore case when matching? Defaults to TRUE.

Value

Object of class "gg", "ggplot".

See Also

Other Making charts: flush_axis, plot_lollipop(), prepare_lollipop_data(), theme_schola()

---

schola_labeller *Labeller with width wrapping and item code-label dictionary*

Description

Turns item codes into labels by looking up provided dictionary and applies them as facet labels.

Usage

```
schola_labeller(dict, width = 80)
```

Arguments

dict	code-label dictionary as named character vector, see dict_from_data()
width	maximal string length per line, if NULL, this is off

Value

object of class labeller, to be used in facet_* ggplot2 functions.

schola_pdf *Schola Empirica PDF Format*

Description

Function called in the output section of the YAML header (see examples below), instructing `knitr` to use the standard Schola PDF document format. If you run into any problem using the Czech language, consult this function details first.

Usage

```
schola_pdf(
  num_format = NULL,
  number_sections = FALSE,
  toc = TRUE,
  template = find_resource("schola_pdf", "schola_template.tex"),
  latex_engine = "xelatex",
  fig_crop = FALSE,
  document_class = "report",
  ...
)
```

Arguments

`num_format`	*Character*, if `cs`, Czech number formatting is used. Other values are currently ignored and will result in default options.
`number_sections`	*Logical*, `TRUE` to number headings. Defaults to `FALSE`.
`toc`	*Logical*, `TRUE` (default) to include a table of contents. The title is set with `pandoc` variable `toc-title` in YAML header.
`template`	*Character*, path to the `.tex` template used by the format. Defaults to standard Schola template bundled in the package. **Changes discouraged.**
`latex_engine`	*Character*, engine used for `.md` to `.pdf` conversion. Only `xelatex` (the default) and `lualatex` are supported because of custom fonts. **Changes discouraged.**
`fig_crop`	*logical*, whether to crop PDF figures. Defaults to `FALSE`. Requires the tools `pdfcrop` and `ghostscript` to be installed, if `TRUE`.
`document_class`	*Character*, one of standard LaTeX document class. Defaults to `report`. **Changes discouraged.**
`...`	Arguments passed on to `rmarkdown::pdf_document` `toc_depth` Depth of headers to include in table of contents `fig_width` Default width (in inches) for figures `fig_caption` `TRUE` to render figures with captions

`df_print` Method to be used for printing data frames. Valid values include "default", "kable", "tibble", and "paged". The "default" method uses a corresponding S3 method of `print`, typically `print.data.frame`. The "kable" method uses the `knitr::kable` function. The "tibble" method uses the **tibble** package to print a summary of the data frame. The "paged" method creates a paginated HTML table (note that this method is only valid for formats that produce HTML). In addition to the named methods you can also pass an arbitrary function to be used for printing data frames. You can disable the `df_print` behavior entirely by setting the option `rmarkdown.df_print` to `FALSE`. See Data frame printing section in bookdown book for examples.

`highlight` Syntax highlighting style passed to Pandoc.
Supported built-in styles include "default", "tango", "pygments", "kate", "monochrome", "espresso", "zenburn", "haddock", and "breezedark".
Two custom styles are also included, "arrow", an accessible color scheme, and "rstudio", which mimics the default IDE theme. Alternatively, supply a path to a '`.theme`' file to use a custom Pandoc style. Note that custom theme requires Pandoc 2.0+.
Pass `NULL` to prevent syntax highlighting.

`keep_tex` Keep the intermediate tex file used in the conversion to PDF. Note that this argument does not control whether to keep the auxiliary files (e.g., '`.aux`') generated by LaTeX when compiling '`.tex`' to '`.pdf`'. To keep these files, you may set `options(tinytex.clean = FALSE)`.

`keep_md` Keep the markdown file generated by knitting.

`citation_package` The LaTeX package to process citations, `natbib` or `biblatex`. Use `default` if neither package is to be used, which means citations will be processed via the command `pandoc-citeproc`.

`includes` Named list of additional content to include within the document (typically created using the `includes` function).

`md_extensions` Markdown extensions to be added or removed from the default definition of R Markdown. See the `rmarkdown_format` for additional details.

`output_extensions` Pandoc extensions to be added or removed from the output format, e.g., `"-smart"` means the output format will be `latex-smart`.

`pandoc_args` Additional command line options to pass to pandoc

`extra_dependencies` A LaTeX dependency `latex_dependency()`, a list of LaTeX dependencies, a character vector of LaTeX package names (e.g. `c("framed", "hyperref")`), or a named list of LaTeX package options with the names being package names (e.g. `list(hyperef = c("unicode=true", "breaklinks=true"), lmodern = NULL)`). It can be used to add custom LaTeX packages to the .tex header.

Value

A modified `bookdown::pdf_document2` with the standard Schola formatting.

Package `babel` hyphenation warning

You may encounter a following warning when your document is being "compiled" to `.pdf` format:

```
Warning: Package babel Warning: No hyphenation patterns were preloaded for
Warning: (babel)                the language 'Czech' into the format.
Warning: (babel)               Please, configure your TeX system to add them and
Warning: (babel)                rebuild the format. Now I will use the patterns
Warning: (babel)             preloaded for \language=0 instead on input line 53.
```

This usually solves by calling `tinytex::tlmgr_install("hyphen-czech")` (if you are a {tinytex} user).

Author(s)

Jan Netik

See Also

Other Report templates and formats: `copy_schola_template()`, `open_schola_template()`, `schola_word2()`, `schola_word()`

Examples

```
## Not run:
# in YAML header ------
output:
reschola::schola_pdf:
toc:false

## End(Not run)
```

---

schola_word *Basic Schola Empirica Word document*

Description

This is a function called in the output of the YAML of the Rmd file to specify using the standard Schola word document formatting.

Usage

```
schola_word(
  reference_docx = find_resource("schola_word", "template.docx"),
  ...
)
```

Arguments

`reference_docx`	Path to custom template. By default, the built-in one is used.
`...`	Arguments passed on to `bookdown::word_document2`

Details

If no template is specified, the function will use the `reschola`'s default template. Path to template is relative to document being compiled. See the examples below, or read the bookdown manual for more details and for a brief guide to Word templating).

Value

A modified `word_document2` with the standard Schola formatting.

Author(s)

Petr Bouchal

Jan Netik

See Also

Other Report templates and formats: `copy_schola_template()`, `open_schola_template()`, `schola_pdf()`, `schola_word2()`

Examples

```
## Not run:
#  # with the default template
output:
reschola::schola_word

# with a user-specified template
output:
reschola::schola_word:
reference_docx:template.docx

## End(Not run)
```

`schola_word2` *Schola Empirica Word document with customisable template*

Description

[Deprecated]

This is a function called in the output part of the YAML section of the Rmd file while using the Word template provided at the same place (see example below).

Usage

```
schola_word2(...)
```

Arguments

...	Arguments to be passed to [bookdown::word_document2]

Details

Compared to schola_word, this "version" comes with no predefined template, so the user can utilize a template stated in YAML header (see the example below, or read the bookdown manual for more details and for a brief guide to Word templating).

Value

A modified word_document2 with the standard Schola formatting, but without hard-coded and unchangeable template.

Author(s)

Jan Netik

See Also

Other Report templates and formats: copy_schola_template(), open_schola_template(), schola_pdf(), schola_word()

Examples

```
## Not run:
output:
reschola::schola_word2:
reference_docx:template.docx

## End(Not run)
```

---

sten *Transform to STEN score (Standard Ten)*

Description

Get your raw total score transformed to stens, i.e., with the mean of 5.5 and a standard deviation of 2. See the Wikipedia article on the topic for more details.

Usage

```
sten(score, standardize = FALSE, bounds = TRUE)
```

Arguments

score	*numeric* vector of scores to transform
standardize	*logical*, center and scale score vector before the transformation? Defaults to FALSE.
bounds	limit result to 1-10 scale, TRUE by default

Value

numeric vector of transformed scores

Examples

```
rnorm(10) %>% sten()
```

---

theme_schola *A Schola Empirica ggplot2 theme*

Description

A wrapper around theme() which provides several shortcuts to setting common options and several defaults. See more in Details.

Usage

```
theme_schola(
  gridlines = c("y", "x", "both", "scatter"),
  base_size = 11,
  family = "Ubuntu Condensed",
  title_family = "Ubuntu",
  margins = TRUE,
  plot.title.position = "plot",
  axis_titles = TRUE,
  multiplot = FALSE,
  ...
)
```

Arguments

gridlines	Whether to display major gridlines along "y" (the default), "x", "both" or draw a "scatter", which has both gridlines and inverted colours.
base_size	Numeric text size in pts, affects all text in plot. Defaults to 11.
family, title_family	font family to use for the (title of the) plot. Defaults to "Ubuntu" for title and "Ubuntu Condensed" for plot.
margins	*logical*, whether to draw margins around the plot or not (the default).

plot.title.position
: where to align the title. Either "plot" (the default, difference from theme() default) or "panel".

axis_titles
: *logical*, draw axis titles? Defaults to TRUE.

multiplot
: if set to TRUE, provides better styling for small multiples created using facet_*.

...
: Arguments passed on to ggplot2::theme

  line all line elements (element_line())

  rect all rectangular elements (element_rect())

  title all title elements: plot, axes, legends (element_text(); inherits from text)

  aspect.ratio aspect ratio of the panel

  axis.text,axis.text.x,axis.text.y,axis.text.x.top,axis.text.x.bottom,axis.text.y.lef tick labels along axes (element_text()). Specify all axis tick labels (axis.text), tick labels by plane (using axis.text.x or axis.text.y), or individually for each axis (using axis.text.x.bottom, axis.text.x.top, axis.text.y.left, axis.text.y.right). axis.text.*.* inherits from axis.text.* which inherits from axis.text, which in turn inherits from text

  axis.ticks,axis.ticks.x,axis.ticks.x.top,axis.ticks.x.bottom,axis.ticks.y,axis.ticks tick marks along axes (element_line()). Specify all tick marks (axis.ticks), ticks by plane (using axis.ticks.x or axis.ticks.y), or individually for each axis (using axis.ticks.x.bottom, axis.ticks.x.top, axis.ticks.y.left, axis.ticks.y.right). axis.ticks.*.* inherits from axis.ticks.* which inherits from axis.ticks, which in turn inherits from line

  axis.ticks.length,axis.ticks.length.x,axis.ticks.length.x.top,axis.ticks.length.x.bo length of tick marks (unit)

  axis.line,axis.line.x,axis.line.x.top,axis.line.x.bottom,axis.line.y,axis.line.y.lef lines along axes (element_line()). Specify lines along all axes (axis.line), lines for each plane (using axis.line.x or axis.line.y), or individually for each axis (using axis.line.x.bottom, axis.line.x.top, axis.line.y.left, axis.line.y.right). axis.line.*.* inherits from axis.line.* which inherits from axis.line, which in turn inherits from line

  legend.background background of legend (element_rect(); inherits from rect)

  legend.margin the margin around each legend (margin())

  legend.spacing,legend.spacing.x,legend.spacing.y the spacing between legends (unit). legend.spacing.x & legend.spacing.y inherit from legend.spacing or can be specified separately

  legend.key background underneath legend keys (element_rect(); inherits from rect)

  legend.key.size,legend.key.height,legend.key.width size of legend keys (unit); key background height & width inherit from legend.key.size or can be specified separately

  legend.text legend item labels (element_text(); inherits from text)

  legend.text.align alignment of legend labels (number from 0 (left) to 1 (right))

legend.title title of legend (element_text(); inherits from title)

legend.title.align alignment of legend title (number from 0 (left) to 1 (right))

legend.position the position of legends ("none", "left", "right", "bottom", "top", or two-element numeric vector)

legend.direction layout of items in legends ("horizontal" or "vertical")

legend.justification anchor point for positioning legend inside plot ("center" or two-element numeric vector) or the justification according to the plot area when positioned outside the plot

legend.box arrangement of multiple legends ("horizontal" or "vertical")

legend.box.just justification of each legend within the overall bounding box, when there are multiple legends ("top", "bottom", "left", or "right")

legend.box.margin margins around the full legend area, as specified using margin()

legend.box.background background of legend area (element_rect(); inherits from rect)

legend.box.spacing The spacing between the plotting area and the legend box (unit)

panel.border border around plotting area, drawn on top of plot so that it covers tick marks and grid lines. This should be used with fill = NA (element_rect(); inherits from rect)

panel.spacing,panel.spacing.x,panel.spacing.y spacing between facet panels (unit). panel.spacing.x & panel.spacing.y inherit from panel.spacing or can be specified separately.

panel.ontop option to place the panel (background, gridlines) over the data layers (logical). Usually used with a transparent or blank panel.background.

plot.background background of the entire plot (element_rect(); inherits from rect)

plot.title.position,plot.caption.position Alignment of the plot title/subtitle and caption. The setting for plot.title.position applies to both the title and the subtitle. A value of "panel" (the default) means that titles and/or caption are aligned to the plot panels. A value of "plot" means that titles and/or caption are aligned to the entire plot (minus any space for margins and plot tag).

plot.subtitle plot subtitle (text appearance) (element_text(); inherits from title) left-aligned by default

plot.caption caption below the plot (text appearance) (element_text(); inherits from title) right-aligned by default

plot.tag upper-left label to identify a plot (text appearance) (element_text(); inherits from title) left-aligned by default

plot.tag.position The position of the tag as a string ("topleft", "top", "topright", "left", "right", "bottomleft", "bottom", "bottomright) or a coordinate. If a string, extra space will be added to accommodate the tag.

strip.clip should strip background edges and strip labels be clipped to the extend of the strip background? Options are "on" to clip, "off" to disable clipping or "inherit" (default) to take the clipping setting from the parent viewport.

`strip.placement` placement of strip with respect to axes, either "inside" or "outside". Only important when axes and strips are on the same side of the plot.

`strip.text,strip.text.x,strip.text.y,strip.text.x.top,strip.text.x.bottom,strip.text` facet labels (`element_text()`; inherits from `text`). Horizontal facet labels (`strip.text.x`) & vertical facet labels (`strip.text.y`) inherit from `strip.text` or can be specified separately. Facet strips have dedicated position-dependent theme elements (`strip.text.x.top`, `strip.text.x.bottom`, `strip.text.y.left`, `strip.text.y.right`) that inherit from `strip.text.x` and `strip.text.y`, respectively. As a consequence, some theme stylings need to be applied to the position-dependent elements rather than to the parent elements

`strip.switch.pad.grid` space between strips and axes when strips are switched (`unit`)

`strip.switch.pad.wrap` space between strips and axes when strips are switched (`unit`)

`complete` set this to `TRUE` if this is a complete theme, such as the one returned by `theme_grey()`. Complete themes behave differently when added to a ggplot object. Also, when setting `complete = TRUE` all elements will be set to inherit from blank elements.

`validate` `TRUE` to run `validate_element()`, `FALSE` to bypass checks.

Details

In particular, the theme: - displays only major gridlines, allowing you to quickly switch which ones; gridlines are thinner, panel has white background

- provides quick option to draw a scatter with grey background - switches defaults for title alignment - turns axis labels off by default: in practice, x axes are often obvious and y axes are better documented in a subtitle - sets backgrounds to a schola-style shade - sets plot title in bold and 120% of base_size

All the changed defaults can be overriden by another call to `theme()`.

See "Making charts" vignette for more complex examples: `vignette('charts', package = 'reschola')`.

Value

a ggtheme object

Note

The default fonts - Ubuntu and Ubuntu Condensed - are contained in this package and can be registered with the system using `register_reschola_fonts()`. You should then install them onto your system like any font, using files in the directories described in the `register_reschola_fonts()` messsage.

See Also

Other Making charts: `flush_axis`, `plot_lollipop()`, `prepare_lollipop_data()`, `schola_barplot()`

Examples

```
library(ggplot2)

# NOTE when `theme_schola()` is used in these examples, fonts
# are set to 'sans' to pass checks on computers without the
# Ubuntu included. If you have these fonts (see Note) you can
# leave these parameters at their default values.

use_reschola_fonts("sans")

# the basic plot for illustration, theme not used

p <- ggplot(mpg) +
  geom_bar(aes(y = class)) +
  labs(title = "Lots of cars", subtitle = "Count of numbers")

# using `theme_schola()` defaults

p +
  theme_schola("x", family = "sans", title_family = "sans")

# in combination with `flush_axis`:

p +
  theme_schola("x", family = "sans", title_family = "sans") +
  scale_x_continuous(expand = flush_axis)

# scatter

ggplot(mpg) +
  geom_point(aes(cty, hwy)) +
  theme_schola("scatter", family = "sans", title_family = "sans") +
  labs(title = "Lots of cars", subtitle = "Point by point")

# Smaller text, flush alignment

ggplot(mpg) +
  geom_point(aes(cty, hwy)) +
  theme_schola("scatter",
    base_size = 9, family = "sans", title_family = "sans"
  ) +
  labs(title = "Lots of cars", subtitle = "Point by point")

# Override defaults changed inside `theme_schola()`

ggplot(mpg) +
  geom_point(aes(cty, hwy)) +
  theme_schola("scatter",
    base_size = 9, family = "sans", title_family = "sans"
  ) +
  labs(title = "Lots of cars", subtitle = "Point by point") +
  theme(panel.background = element_rect(fill = "lightpink"))
```

`use_reschola_fonts` *Make ggplot2 use font(s) in text-based geoms*

Description

Sets {ggplot2} defaults for most text-based geoms in {ggplot2}, {ggtext}, and {ggrepel}.

Usage

```
use_reschola_fonts(
  family = "Ubuntu Condensed",
  face = "plain",
  size = 3.5,
  color = "#2b2b2b"
)

set_reschola_ggplot_fonts(
  family = "Ubuntu Condensed",
  face = "plain",
  size = 3.5,
  color = "#2b2b2b"
)
```

Arguments

`family`	font family, defaults to reschola recommended
`face`	font face
`size`	font size
`color`	font color

Details

Geoms covered: "text", "label", "richtext", "text_box", "text_repel", and "label_repel".

See Also

Other Font helpers and shortcuts: `import_fonts()`, `install_reschola_fonts()`, `register_reschola_fonts()`

`with_clr` *Create HTML span tag with text and color style*

Description

Create HTML span tag with text and color style

Usage

```
with_clr(text, color = "black", alpha = 1, ...)
```

Arguments

`text`	*character*, text that will be colored.
`color`	*character*, color applied to the text, defaults to black.
`alpha`	*numeric*, opacity in interval 0–1, where 1 is no transparency, i.e. full opacity
`...`	Arguments passed on to `htmltools::span` `.noWS` Character vector used to omit some of the whitespace that would normally be written around this tag. Valid options include `before`, `after`, `outside`, `after-begin`, and `before-end`. Any number of these options can be specified. `.renderHook` A function (or list of functions) to call when the `tag` is rendered. This function should have at least one argument (the `tag`) and return anything that can be converted into tags via `as.tags()`. Additional hooks may also be added to a particular `tag` via `tagAddRenderHook()`.

Value

an object of class `shiny.tag`, coercible to character

Examples

```
html <- paste0(
  with_clr("Red", "red"), ", ",
  with_clr("green", "green"),
  " and ",
  with_clr("blue", "blue"),
  " are the basic colors."
)

library(ggplot2)
library(ggtext)

ggplot() +
  geom_richtext(aes(x = 1, y = 1, label = html), size = 8) +
  theme_void()
```

Index