Package: crosstabser (via r-universe)

February 19, 2025

Title Generate Crosstabs of Labelled Data Sets with Excel Files

Version 0.0.0.9000

Description An R package to use commands in Excel files to generate crosstabs of labelled data sets (usually survey data). The crosstabs can be printed to the console, and serve as an input for an app to plot them interactively.

License AGPL (>= 3)

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.2

Imports openxlsx2, R6, tibble, dplyr, tidyr, stringr, purrr, rlang, matrixStats, Hmisc, tidyselect, pillar, cli, datadaptor, vctrs, S7, xml2, jsonlite

Remotes gitlab::urswilke/datadaptor

Suggests testthat (>= 3.0.0), haven, ggplot2, withr, knitr, rmarkdown, janitor, glue

Config/testthat/edition 3

VignetteBuilder knitr

Language en-US

URL https://urswilke.gitlab.io/crosstabser, https://gitlab.com/urswilke/crosstabser, https://urswilke.r-universe.dev/crosstabser

Config/pak/sysreqs make libicu-dev libxml2-dev libx11-dev zlib1g-dev

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

RemoteUrl https://gitlab.com/urswilke/crosstabser

RemoteRef HEAD

RemoteSha 7a195cd9ed0b1e6a0c2cde368e585fb069bbc0f1

Contents

`create_tabula` *Create an Excel mapping file based on a labelled dataframe*

Description

This function first calls `datadaptor::create_mapping_workbook()`. Additionally to the sheets "Variables", "Label", "Verbatims" & "Free", it will insert 2 more sheets "Macro" & "Questions". The Excel workbook is then written to the file `mapping_file`. Please refer to `vignette("crosstabser")` & `vignette("questions-parameters")` for details how to use the mapping.

Usage

```
create_tabula(df_raw, mapping_file, mapping_type = "excel")
```

Arguments

`df_raw`	dataframe with labelled variables, e.g. resulting from haven::read_sav
`mapping_file`	name of the Excel file to be created
`mapping_type`	String specifying the mapping type. Either "excel" or "list". Defaults to "excel".

Examples

```
spss_file <- system.file(
  "extdata",
  "fruit_survey.sav",
  package = "datadaptor"
)
df <- haven::read_sav(spss_file)
# The next command creates an empty mapping file `mapping.xlsx`:
## Not run:
create_tabula(df, "mapping.xlsx")

## End(Not run)
```

gen_data_json *Generate data json string*

Description

This helper function generates a json string containing the data produced by `Tabula$get_crosstabs_data()`, and which then can be put as the `"data"` attribute of the `<table-charter>` app. It is also used by `Tabula$save_html_app()`

Usage

```
gen_data_json(l)
```

Arguments

`l`	list object generated by `Tabula$get_crosstabs_data()`.

Value

A character string

Examples

```
df <- tibble::tibble(
  q1 = c(1, 2, 1) |> haven::labelled(c(Yes = 1, No = 2), label = "Super important question"),
   age = c(2, 1, 1) |> haven::labelled(c("18-39" = 1, "40+" = 2), label = "age")
)
mapping_file = list(
  Questions = data.frame(
    Type  = "cat",
    RowVar = "q1",
    Title = "The crosstab's title"
  ),
  Macro = list(ColVar = "age")
)
m <- Tabula$new(df, mapping_file)
m$get_crosstabs_data() |> gen_data_json()
```

get_tabula_options *Generate list of options for a* `Tabula` *object*

Description

Generate list of options for a `Tabula` object

Usage

```
get_tabula_options(tabula, book_no = NULL, ...)
```

Arguments

tabula	An object generated with Tabula$new().
book_no	An integer skalar to identify the
...	Arguments passed to methods

Examples

```
## Not run:
# TODO: document!
- for this we should add an example excel mapping file to the crosstabser package
- and then add a docs example, something like this:
# Only for documentation purposes:
# (`get_mapping_options()` isn't supposed to be be called directly).
mapping_file <- system.file(
  "extdata",
  "<file-to-be-created-mapping.xlsx>",
  package = "crosstabser"
)
m <- Tabula$new(mapping_file = mapping_file)
# Result of datadaptor::get_mapping_options() in `da` field:
m$opts$da
# Result of get_tabula_options() in `da` field:
m$opts$ct

## End(Not run)
```

---

print.Qtab *Print object of class "Qtab"*

Description

Print object of class "Qtab"

Usage

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

Arguments

x	Qtab object
...	Arguments passed to print()

Examples

```
# see `?Tabula`
```

Qrow *Questions row class*

Description

This is not supposed to be used directly. When creating a "Tabula" object, this will generate a list of `Qrow` objects in its `$qrows` field.

Public fields

`p` parameters extracted from `df_qrow`

`m` `Tabula` object

`qtabs` list of `Qtabs` objects

`log` log entries

`ditw` This is the "dust in the wind" list object field that stores data that didn't make it into their own field. For developers only! For reproducible code you should NEVER rely on this field as it might be subject to change without any warning.

Methods

Public methods:

- `Qrow$new()`
- `Qrow$clone()`

Method `new()`:

Usage:

```
Qrow$new(df_qrow, mapping, ...)
```

Arguments:

`df_qrow` row of the Questions dataframe

`mapping` `Tabula` object

`...` Not used at the moment.

Method `clone()`: The objects of this class are cloneable with this method.

Usage:

```
Qrow$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
# see `?Tabula`
```

Qtab *Qtab class*

Description

This is not supposed to be used directly. When creating a "Tabula" object, this will generate a list of `Qrow` objects in its `$qrows` field, themselves each containing a list of `Qtab` objects in their `$qtabs` fields.

Details

`Qtab` objects have a custom print method `print.Qtab` (see examples in `?Tabula`).

Public fields

`p` parameters

`d` data

`m` `Tabula` object

Methods

Public methods:

- `Qtab$new()`
- `Qtab$clone()`

Method `new()`:

Usage:

```
Qtab$new(params, mapping, ...)
```

Arguments:

`params` Parameters from `Qrow` object

`mapping` `Tabula` object

`...` Not used at the moment.

Method `clone()`:

The objects of this class are cloneable with this method.

Usage:

```
Qtab$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
# see `?Tabula`
```

Tabula *Tabulation class*

Description

The class Tabula can be used to calculate the crosstabs specified on the Questions sheet of the Excel mapping file.

Super class

datadaptor::Mapping -> Tabula

Public fields

dat_mod dat_mod modified data field of the super-class datadaptor::Mapping.

mapping_file mapping_file file path field of the super-class datadaptor::Mapping.

dat dat input data field of the super-class datadaptor::Mapping. If this is specified, dat_mod will be ignored, and instead generated with datadaptor::Mapping$modify_data()

qrows A list() of Qrow objects

ditw This is the "dust in the wind" list object field that stores data that didn't make it into their own field. For developers only! For reproducible code you should NEVER rely on this field as it might be subject to change without any warning. This overwrites the datadaptor::Mapping$ditw field; the list field additionally contains the ct element.

Methods

Public methods:

- Tabula$new()
- Tabula$set_options()
- Tabula$calc_qtabs()
- Tabula$save_html_app()
- Tabula$get_crosstabs_data()
- Tabula$print()
- Tabula$clone()

Method new(): Initialize a Tabula object

Usage:

```
Tabula$new(
  dat_mod = NULL,
  mapping_file = NULL,
  row = NULL,
  dat = NULL,
  tabulate = TRUE,
  ...
)
```

Arguments:

`dat_mod` `dat_mod` modified data field of the super-class `datadaptor::Mapping`.

`mapping_file` `mapping_file` file path field of the super-class `datadaptor::Mapping`.

`row` Numeric vector with the row numbers in the Questions sheet, where crosstabs should be calculated, when calling `Tabula$calc_qtabs()`. Or `NULL` (the default) resulting in the selection of all row numbers where `Type` is specified.

`dat` `dat` input data field of the super-class `datadaptor::Mapping`. If this is specified, `dat_mod` will be ignored, and instead generated with `datadaptor::Mapping$modify_data()`

`tabulate` Logical, whether to call the `Tabula$calc_qtabs()` method when initializing (defaults to `TRUE`).

`...` Arguments passed to `Tabula$set_options()`

Method `set_options()`**:** Set `Tabula` options. This overwrites `datadaptor::Mapping$set_options()`

Usage:

```
Tabula$set_options(...)
```

Arguments:

`...` Arguments passed to `get_tabula_options()`.

Method `calc_qtabs()`**:** Calculate the crosstabs

Usage:

```
Tabula$calc_qtabs(row = NULL)
```

Arguments:

`row` Numeric vector with the row numbers in the Questions sheet, where crosstabs should be calculated, when calling `Tabula$calc_qtabs()`. Or `NULL` (the default) resulting in the selection of all row numbers where `Type` is specified.

Method `save_html_app()`**:** Write a table_charter app html file of the crosstab data

Usage:

```
Tabula$save_html_app(
  template_file =
   "https://gitlab.com/urswilke/table_charter/-/raw/main/example_dashboard.html",
  output_file = "dashboard.html",
  project_data = NULL
)
```

Arguments:

`template_file` Path to the template file (see description).

`output_file` File path to the table_charter app html file.

`project_data` Either a `list()` object to modify the default: `list(logo_base64 = "", logo_url = "https://gitlab.com/urswilke/table_charter/-/raw/main/img/logo_small.svg", title = "Dashboard", date = Sys.Date())`, or `NULL` (the default). If `NULL`, nothing is done. The fields will modify the elements in the header of the dashboard.

Details: This needs a valid html `template_file`, i.e. one of:

- The file example_dashboard.html which is directly scraped from the table_charter repo by default (no installation of table_charter needed).

- For deploying it in the web or running it on a dev server, you need to install table_charter first, and then use the file index.html on your machine.
- After installing, you can also generate a standalone html file (without the need to download javascript libraries) by running:

  ```
  npm run standalone-build
  ```

  and then using the template file created in the `dist/` sub-directory.

Method `get_crosstabs_data()`**:** Return the crosstabs data of the `Tabula` object

This method returns a list of dataframes containing all the crosstabs information. Thus it's not chainable.

Usage:

```
Tabula$get_crosstabs_data()
```

Returns: A list of dataframes with the data of the crosstabs; see `vignette("data-format")`.

Method `print()`**:** Print the crosstabs of the `Tabula` object

This method is called under the hood, if you `print()` a `Tabula` object. This will call the print method of all `Qrow` elements in the `Tabula$qrows` field.

Usage:

```
Tabula$print(...)
```

Arguments:

`...` Not used for now.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
Tabula$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
df <- tibble::tibble(
  q1 = c(1, 2, 1) |> haven::labelled(c(Yes = 1, No = 2), label = "Super important question"),
  age = c(2, 1, 1) |> haven::labelled(c("18-39" = 1, "40+" = 2), label = "age")
)
mapping_file = list(
  Questions = data.frame(
    Type  = "cat",
    RowVar = "q1",
    Title = "The crosstab's title"
  ),
  Macro = list(ColVar = "age")
)
m <- Tabula$new(df, mapping_file)
m
# The previous line prints the "Tabula" object.
# Under the hood, a list of `Qrow` objects were generated.
# Printing `m` prints the list of `Qtab` elements of each `Qrow`:
```

```
m$qrows
# For instance, this prints the list of `Qtab` elements
# of the first `Qrow` element:
m$qrows[[1]]$qtabs |> print()
```

Index