Package 'duawranglr'

October 13, 2022

Title Securely Wrangle Dataset According to Data Usage Agreement

Version 0.6.7

URL https://github.com/btskinner/duawranglr

BugReports https://github.com/btskinner/duawranglr/issues

Description Create shareable data sets from raw data files that contain protected elements. Relying on master crosswalk files that list restricted variables, package functions warn users about possible violations of data usage agreement and prevent writing protected elements.

Depends R (>= 3.1.2)

License MIT + file LICENSE

Encoding UTF-8

Imports haven, readxl, readr, digest, dplyr

RoxygenNote 7.1.1

Suggests testthat, knitr, rmarkdown

VignetteBuilder knitr

NeedsCompilation no

Author Benjamin Skinner [aut, cre] (<https://orcid.org/0000-0002-0337-7415>)

Maintainer Benjamin Skinner <btskinner@coe.ufl.edu>

Repository CRAN

Date/Publication 2021-04-15 14:20:03 UTC

R topics documented:

`check_dua_restrictions` *Check data frame columns against currently set restrictions*

Description

Once the DUA crosswalk and level have been set, a working data frame can be checked against active data element restrictions. The data frame must pass before it can be writen using `write_dua_df()`.

Usage

```
check_dua_restrictions(df)
```

Arguments

`df`	Data frame to check against set DUA restriction level.

Examples

```
## --------------
## Setup
## --------------
## set DUA crosswalk
dua_cw <- system.file('extdata', 'dua_cw.csv', package = 'duawranglr')
set_dua_cw(dua_cw)
## read in data
admin <- system.file('extdata', 'admin_data.csv', package = 'duawranglr')
df <- read_dua_file(admin)
## --------------

## set restriction level
set_dua_level('level_iii')

## show restrictions
see_dua_level(show_restrictions = TRUE)

## see variables in administrative data file
names(df)

## remove restrictive variables
df <- dplyr::select(df, -c(sid,sname,tname))

## confirm
```

```
check_dua_restrictions(df)
```

`deid_dua` *Convert identifying variable to unique hash*

Description

Convert a column of unique but restricted IDs into a set of new IDs using secure (SHA-2) hashing algorithm. Users have the option of saving a crosswalk between the old and new IDs in case observations need to reidentified at a later date.

Usage

```
deid_dua(
  df,
  id_col = NULL,
  new_id_name = "id",
  id_length = 64,
  existing_crosswalk = NULL,
  write_crosswalk = FALSE,
  crosswalk_filename = NULL
)
```

Arguments

`df`	Data frame
`id_col`	Column name with IDs to be replaced. By default it is `NULL` and uses the value set by the `id_column` argument in `set_dua_level()` function.
`new_id_name`	New hashed ID column name, which must be different from old name.
`id_length`	Length of new hashed ID; cannot be fewer than 12 characters (default is 64 characters).
`existing_crosswalk`	File name of existing crosswalk. If existing crosswalk is used, then `new_id_name`, `id_length`, `id_length`, and `crosswalk_name` will be determined by the already existing crosswalk. Arguments given for these values will be ignored.
`write_crosswalk`	Write crosswalk between old ID and new hash ID to console (unless `crosswalk_name` is given value).
`crosswalk_filename`	Name of crosswalk file with path; defaults to generic name with current date (YYYYMMDD) appended.

Examples

```
## --------------
## Setup
## --------------
## set DUA crosswalk
dua_cw <- system.file('extdata', 'dua_cw.csv', package = 'duawranglr')
set_dua_cw(dua_cw)
## read in data
admin <- system.file('extdata', 'admin_data.csv', package = 'duawranglr')
df <- read_dua_file(admin)
## --------------

## show identified data
df

## deidentify
df <- deid_dua(df, id_col = 'sid', new_id_name = 'id', id_length = 12)

## show deidentified data
df

## Not run:
## save crosswalk between old and new ids for future
deid_dua(df, write_crosswalk = TRUE)

## use existing crosswalk (good for panel datasets that need link)
deid_dua(df, existing_crosswalk = './crosswalk/master_crosswalk.csv')

## End(Not run)
```

`duawranglr` *duawranglr: Securely Wrangle Dataset According to Data Usage Agreement*

Description

The guiding principle behind `duawranglr` is to make it easier for organizations to share data with protected elements and/or personally idenfiable information (PII) with researchers. There are two key problems this package attempts to solve:

Details

1. Data owners and reseachers may collaborate on multiple projects under a single data usage agreement (DUA), each with a different level of data security required.
2. Administrators tasked with approving data requests do not always have the time or technical proficiency to review the code that reads, subsets, filters, and deidentifies data files according to a data usage agreement.

The duawranglr package uses a simple crosswalk file that lists restricted variables according to security levels prespecified in a DUA and a suite of functions that warn users about possible violations of data usage agreement to prevent writing protected elements. The DUA crosswalk can be an Excel spreadsheet, which means that restricted data elements can be easily added and approved by administrators.

Functions in the package do not replace existing data wrangling functions nor guarantee data security. But if used correctly, data administrators can more easily participate in the data sharing process and have more surety that data are being properly secured before they are transferred to researchers.

See the package vignette for more details about the motivation for the package and an extended example use case.

`make_dua_template` *Interactive function to create template file*

Description

Use this function to create a template script that puts package functions in order and, based on question answers, prepopulates some arguments. By default, this function is run in interactive mode, meaning that it will not work in a script unless a list of answers is given to `answer_list` argument. Note that the saved template file is not intended to be run as is, but only to provide a starting structure for a cleaning script.

Usage

```
make_dua_template(file_name, include_notes = TRUE, answer_list = NULL)
```

Arguments

`file_name`	Name with path of template script.
`include_notes`	If `TRUE`, the template file will include notes and suggestions for completing the script; default value is `TRUE`.
`answer_list`	List of answer strings to provide if you don't want to answer questions interactively. See details for questions and expected input type. Leave as default `NULL` for interactive mode.

Details

Questions to answer if using the `answer_list` argument:

1. Do you want to set the DUA crosswalk file? `'Yes'` or `'No'`
   (a) DUA crosswalk file (with path): '< file name with path >'
2. Do the data need to be deidentified? `'Yes'` or `'No'`
   (a) Would like to select the ID column now? `'Yes'` or `'No'`
   (b) ID column name: '< column name string >'

If answers to questions (1) and (2) are `No`, then strings for 1(a), 2(a), and 2(b) can be left empty since they will be ignored.

Examples

```
## Not run:
## run interactively
make_dua_template('data_clean.R')

## ...and don't include extra notes
make_dua_template('data_clean.R', include_notes = FALSE)

## End(Not run)

## make template to be filled in
file <- file.path(tempdir(), 'data_clean.R')
make_dua_template(file, answer_list = list('No','','No','',''))

## show
writeLines(readLines(file))
```

---

read_dua_file *Read in raw data file*

Description

This function is a wrapper for that will read a variety of file types. The primary reason to use it rather than base R or tidyverse functions is that every new file read will reset the `check_pass` environment variable to `FALSE`. This is a security feature in that it requires a new data check each time a new file is read into memory.

Usage

```
read_dua_file(file, ...)
```

Arguments

`file`	File name to be read into memory
`...`	Arguments to pass to read function based on the input type; see details for more information.

Details

The following input types are supported (with the underlying read function and default arguments accompanying):

- **rds**: `readRDS()`
- **rdata**: `load()`
- **csv**: `readr::read_delim(...,row.names = FALSE, sep = ,)`
- **tsv**: `read::read_delim(...,row.names = FALSE, sep = '\t')`

- **delimited**: `readr::read_delim(...,row.names = FALSE)`
- **excel**: `read_xl::read_excel(...,sheet = 1)`
- **stata**: `haven::read_dta()`
- **sas**: `haven::read_sas()`
- **spss**: `haven::read_sav()`

All arguments for these internal write functions, including those with default values, can be modified by adding them to the top-level `read_dua_file()` function.

Examples

```
## --------------
## Setup
## --------------
## set DUA crosswalk
dua_cw <- system.file('extdata', 'dua_cw.csv', package = 'duawranglr')
set_dua_cw(dua_cw)
## --------------

## read in data
file <- system.file('extdata', 'admin_data.csv', package = 'duawranglr')
df <- read_dua_file(file)

## show
df

## Not run:
## read in other file types
read_dua_file('admin_data.rds')
read_dua_file('admin_data.txt', sep = '|')
read_dua_file('admin_data.dta')
read_dua_file('admin_data.xlsx', sheet = 2)

## End(Not run)
```

---

see_dua_level *Show current DUA restriction level setting*

Description

After setting the DUA restriction level, check the setting and restricted data elements.

Usage

```
see_dua_level(show_restrictions = FALSE, sort_vars = TRUE, ...)
```

Arguments

show_restrictions	Show the names of the variables that are restricted by the current level if TRUE.
sort_vars	Sort variables alphabetically when printing restrictions; if FALSE, prints in the order saved in the crosswalk file
...	For debugging.

Examples

```
## --------------
## Setup
## --------------
## set DUA crosswalk
dua_cw <- system.file('extdata', 'dua_cw.csv', package = 'duawranglr')
set_dua_cw(dua_cw)
## --------------

## set restriction level
set_dua_level('level_i')

## show name of current restriction level
see_dua_level()

## ...include names of restricted elements
see_dua_level(show_restrictions = TRUE)

## ...show variable names in order saved in crosswalk file
see_dua_level(show_restrictions = TRUE, sort_vars = FALSE)
```

---

see_dua_options *Show DUA crosswalk options*

Description

Once the DUA crosswalk has been loaded, show the available restriction levels with associated data element names.

Usage

```
see_dua_options(level = NULL, sort_vars = TRUE, ...)
```

Arguments

level	String name or vector of string names of levels to show.
sort_vars	Sort variables alphabetically when printing restrictions; if FALSE, prints in the order saved in the crosswalk file
...	For debugging.

Examples

```
## --------------
## Setup
## --------------
## set DUA crosswalk
dua_cw <- system.file('extdata', 'dua_cw.csv', package = 'duawranglr')
set_dua_cw(dua_cw)
## --------------

## see level i options
see_dua_options(level = 'level_i')

## compare two levels of options
see_dua_options(level = c('level_i','level_ii'))

## show all option levels
see_dua_options()
```

`set_dua_cw` *Function to set data usage agreement data crosswalk*

Description

Initial function to read in and set the working DUA crosswalk.

Usage

```
set_dua_cw(
  dua_cw,
  delimiter = NULL,
  sheet = NULL,
  ignore_columns = NULL,
  remap_list = NULL
)
```

Arguments

`dua_cw`	Data frame object or file with columns representing security levels (header equalling name of level) and rows in each column representing restricted variables
`delimiter`	Set the delimiter if reading in a delimited file that is neither a comma separated value (CSV) nor tab separated value (TSV).
`sheet`	Set the sheet name or number if reading in a DUA crosswalk from Excel file with values not on the first sheet.
`ignore_columns`	**(Experimental)** Columns to ignore when reading in DUA crosswalk.
`remap_list`	**(Experimental)** If raw variable names should be remapped to new names, provide list with mappings from old names column to new names column.

Details

The crosswalk file can be in a variety of formats. It will be read automatically without additional arguments if it's in the following formats:

- **R**: `*.rdata`, `*.rda`, `*.rds`
- **delimited**: if `*.csv` or `*.tsv`
- **Stata**: `*.dta`
- **SAS**: `*.sas7bdat`
- **SPSS**: `*.sav`
- **Excel**: `*.xls`, `*.xlsx` if on first sheet

If a **delimited** file other than comma- or tab-separated values or an `Excel` file with information on a sheet other than the first, use the appropriate arguments to set that correct values.

Examples

```
## path to DUA crosswalk file
dua_cw <- system.file('extdata', 'dua_cw.csv', package = 'duawranglr')

## set DUA restrictions using crosswalk file
set_dua_cw(dua_cw)

## Not run:
## set using crosswalks stored in other file types
set_dua_cw('dua_cw.dta')
set_dua_cw('dua_cw.txt', delimiter = '|')
set_dua_cw('dua_cw.csv', remap_list = list('level_i_new' = 'level_i_old'))

## End(Not run)
```

---

set_dua_level *Set data restriction level*

Description

Set data restrictions to one of the levels in the DUA crosswalk.

Usage

```
set_dua_level(level, deidentify_required = FALSE, id_column = NULL)
```

Arguments

`level`	String value of the data restriction level
`deidentify_required`	Set to `TRUE` if ID column must be changed to protect unique identifier.
`id_column`	Column with unique IDs that must be identified if `deidentify_required == TRUE`.

Examples

```
## --------------
## Setup
## --------------
## set DUA crosswalk
dua_cw <- system.file('extdata', 'dua_cw.csv', package = 'duawranglr')
set_dua_cw(dua_cw)
## --------------

## set restrictions at first level
set_dua_level('level_i')

## ...same, but set unique ID column to be deidentified
set_dua_level('level_i', deidentify_required = TRUE, id_column = 'sid')
```

`write_dua_df` *Write DUA approved data set*

Description

This function is a wrapper for a variety of write functions that also checks whether the data set has been cleared for writing based on the DUA level restrictions chosen by the user. If restricted variables remain in the data set, the function will return an error and will not write the data set.

Usage

```
write_dua_df(
  df,
  file_name,
 output_type = c("rds", "rdata", "csv", "tsv", "delimited", "stata", "sas", "spss"),
  ...
)
```

Arguments

`df`	Data frame object to save.
`file_name`	Name and path for saved file, with or without file type ending.
`output_type`	Output data file type; options include `rds` (DEFAULT), `rdata`, `csv`, `tsv`, `delimited`, `stata`, `sas`, and `spss`.
`...`	Arguments to pass to write function based on the selected `output_type`; see details for more information.

Details

The following output types are supported (with the underlying write function and default arguments accompanying):

- **rds**: `saveRDS()`
- **rdata**: `save()`
- **csv**: `write.table(...,row.names = FALSE, sep = ,)`
- **tsv**: `write.table(...,row.names = FALSE, sep = '\t')`
- **delimited**: `write.table(...,row.names = FALSE)`
- **stata**: `haven::write_dta()`
- **sas**: `haven::write_sas()`
- **spss**: `haven::write_sav()`

All arguments for these internal write functions, including those with default values, can be modified by adding them to the top-level `write_dua_df()` function.

Examples

```
## --------------
## Setup
## --------------
## set DUA crosswalk
dua_cw <- system.file('extdata', 'dua_cw.csv', package = 'duawranglr')
set_dua_cw(dua_cw)
## read in data
admin <- system.file('extdata', 'admin_data.csv', package = 'duawranglr')
df <- read_dua_file(admin)
## set restriction level
set_dua_level('level_iii')
## remove restrictive variables
df <- dplyr::select(df, -c(sid,sname,tname))
## --------------

## check restrictions
check_dua_restrictions(df)

## able to write since restrictions check passed
file <- file.path(tempdir(), 'clean_data.csv')
write_dua_df(df, file_name = file, output_type = 'csv')

## Not run:
 write_dua_df(df, 'clean_data', output_type = 'delimited', sep = '|')
 write_dua_df(df, 'clean_data', output_type = 'stata', version = 11)

## End(Not run)
```

Index