Package 'DDIwR'

September 15, 2019

Version 0.4

Date 2019-09-15

Title DDI with R

Depends R (>= 3.3.0)

Imports admisc (>= 0.4), haven, readr, tibble, xml2

Description
Useful functions for various DDI (Data Documentation Initiative) related inputs and outputs.

License GPL (>= 2)

NeedsCompilation no

Author Adrian Dusa [aut, cre]

Maintainer Adrian Dusa <dusa.adrian@unibuc.ro>

Repository CRAN

Date/Publication 2019-09-15 21:10:02 UTC

R topics documented:

`DDIwR-package` *Useful functions for various DDI (Data Documentation Initiative) related outputs.*

Description

This package provides various functions to read DDI based metadata documentation, and write dedicated setup files for R, SPSS, Stata and SAS to read an associated .csv file containing the raw data, apply labels for variables and values and also deal with the treatment of missing values.

It can also generate a DDI metadata file out of an R information object, which can be used to export directly to the standard statistical packages files (such as SPSS, Stata and SAS), using the versatile package **haven**. For R, the default object to store data and matadata is a `tibble`.

The research leading to the initial functions in this package has received funding from the European Union's Seventh Framework Programme (FP7/2007-2013) under grant agreement no. 262608 (DwB - Data without Boundaries)

Details

Package:	DDIwR
Type:	Package
Version:	0.4
Date:	2019-09-15
License:	GPL (>= 2)

Author(s)

Authors:
Adrian Dusa
Department of Sociology
University of Bucharest
`<dusa.adrian@unibuc.ro>`

Maintainer:
Adrian Dusa

`convert` *Convert a dataset from one statistical software to another*

Description

This function converts (or transfers) between R, Stata, SPSS, SAS, and DDI XML files. Unlike the regular import / export functions from packages **haven** or **rio**, this function uses the DDI standard as an exchange platform and also attempts a conversion of the missing values.

Usage

```
convert(from, to, embed = FALSE, binpath = "", ...)
```

Arguments

`from`	A path to a file, or a tibble object
`to`	Character, the name of a software package or a path to a specific file
`embed`	Boolean, embed the data when generating a DDI XML file
`binpath`	Path to the binary executable file, to run a recoding script
`...`	Additional parameters passed to exporting functions, see the Details section

Details

When the argument `to` specifies a certain statistical package (`"R"`, `"Stata"`, `"SPSS"` or `"SAS"`), the name of the destination file will be the same as the name of the input file from the argument `to`, with an automatically added software specific extension.

Alternatively, the argument `to` can be specified as a path to a specific file, in which case the software package is determined from its file extension. The following extentions are currently recognized: `.xml` for DDI, `.rds` for R, `.dta` for Stata, `.sav` for SPSS and `.sas7bdat` for SAS.

The argument `binpath` is used only for Stata (if installed on the local machine), to coerce regular missing values to their specific missing values using letters from `a` to `z`, given that package **haven** does not convert Stata missing values by default. Specifying the path to the binary executable file is also a Boolean signal to attempt converting the missing values via an automatic script that recodes all unique missing values to the same letters, the lowest numerical value being assigned to the letter `a`.

Additional parameters can be specified via the three dots argument `...`, that are passed to the respective functions from package **haven**. For instance the function `write_dta()` has an additional argument called `version` (from 8 to a maximum and default value of 14) when writing a Stata file.

Note that this function creates a target file in the same directory as the source file, which is different from importing the source file into R. To import a file, users should refer to the specific functions from package **haven**, such as `read_sav()` or `read_dta()` etc., and be aware the result object is a `tibble`.

The current version reads and creates DDI Codebook version 2.5, with future versions to extend the functionality for DDI Lifecycle versions 3.x and link to the future package **DDI4R** for the UML model based version 4. It extends the standard DDI Codebook by offering the possibility to embed a CSV version of the raw data into the XML file containing the Codebook, into a `notes` child of the `fileDscr` component. This type of Codebook is unique to this package and automatically detected when converting to another statistical software.

Future versions will attempt to extend converting the missing values to SAS types, but otherwise users can also use a setup file produced by function `setupfile()` and run the commands manually.

When importing a file, the R object of choice is a tibble because is the only type of object in R that allows specifying multiple (coded) missing values. It also plays nicely with the SPSS types of variables, which are the most commonly used in the social sciences.

Author(s)

Adrian Dusa

References

DDI - Data Documentation Initiative, see https://www.ddialliance.org/

See Also

`setupfile`, `getMetadata`, `tibble`, `labelled`, `labelled_spss`

Examples

```
## Not run:
# Assuming an SPSS file called test.sav is located in the working directory
# the following command will extract the metadata in a DDI Codebook and
# produce a test.xml file in the same directory
convert("test.sav", to = "DDI")

# It is possible to include the data in the XML file, using:
convert("test.sav", to = "DDI", embed = TRUE)

# To produce a Stata file:
convert("test.sav", to = "Stata")

# Since Stata has different types of missing values than SPSS, it is
# possible to transform these missing values via an automatically run script
# using the argument "binpath", assuming that Stata is installed

# The paths to the binaries differ in various operating systems. A possible
# path for Windows, for Stata version 12 could be:
binpath <- "C:/Progra~1/Stata12/Stata.exe"

# For MacOS, the path could be:
binpath <- "/Applications/Stata/Stata.app/Contents/MacOS/Stata"

# The final command, which also converts to Stata types of missing values
convert("test.sav", to = "Stata", binpath = binpath)

## End(Not run)
```

`exportDDI` *Export to a DDI metadata file*

Description

This function creates a DDI version 2.5, XML file structure.

Usage

```
exportDDI(codebook, file = "", embed = TRUE, OS = "", indent = 4)
```

Arguments

`codebook`	A list object containing the metadata, or a path to a directory where these objects are located, for batch processing
`file`	either a character string naming a file or a connection open for writing. "" indicates output to the console.
`embed`	Embed the CSV datafile in the XML file, if present.
`OS`	The target operating system, for the eol - end of line character(s)
`indent`	Indent width, in number of spaces

Details

The information object is essentially a list having two main list components:

- `fileDscr`, if the data is provided in a subcomponent named `datafile`

- `dataDscr`, having as many components as the number of variables in the (meta)data. For each variable, there should a mandatory subcomponent called `label` (that contains the variable's label) and, if the variable is of a categorical type, another subcomponent called `values`.

Additional informations about the variables can be specified as further subcomponents, combining DDI specific data but also other information that might not be covered by DDI:

- `measurement` is the equivalent of the specific DDI attribute `nature` of the `var` element, and it accepts these values: `"nominal"`, `"ordinal"`, `"interval"`, `"ratio"`, `"percent"`, and `"other"`.

- `type` is useful for multiple reasons. A first one, if the variable is numerical, is to differentiate between `discrete` and `contin` values of the attribute `intrvl` from the same DDI element `var`. Another reason is to help identifying pure string variables (containing text), when the subcomponent `type` is equal to `"char"`. It is also used for the subelement `varFormat` of the element `var`. Finally, another reason is to differentiate between pure categorical (`"cat"`) and pure numerical (`"num"`) variables, as well as mixed ones, among which `"numcat"` referring to a numerical variable with very few values (such as the number of children), for which it is possible to also produce a table of frequencies along the numerical summaries. There are also categorical variables that can be interpreted as numeric (`"catnum"`), such as a Likert type response scale with 7 values, where numerical summaries are also routinely performed along with the usual table of frequencies.

- `missing` is an important subcomponent, indicating which of the values in the variable are going to be treated as missing values, and it is going to be exported as the attribute `missing` of the DDI subelement `catgry`.

There are many more possible attributes and DDI elements to be added in the information object, future versions of this function will likely expand.

For the moment, only DDI codebook version 2.5 is exported, but DDI Lifecycle is also possible.

The argument `OS` can be either:
`"windows"` (default), or `"Windows"`, `"Win"`, `"win"`,
`"MacOS"`, `"Darwin"`, `"Apple"`, `"Mac"`, `"mac"`,
`"Linux"`, `"linux"`.

The end of line separator changes only when the target OS is different from the running OS.

The argument `indent` controls how many spaces will be used in the XML file, to indent the different subelements.

Value

An XML file containing a DDI version 2.5 metadata.

See Also

http://www.ddialliance.org/Specification/DDI-Codebook/2.5/XMLSchema/field_level_documentation.html

Examples

```
codeBook <- list(dataDscr = list(
ID = list(
    label = "Questionnaire ID",
    type = "num",
    measurement = "interval"
),
V1 = list(
    label = "Label for the first variable",
    values = c(
        "No"             =  0,
        "Yes"            =  1,
        "Not applicable" = -7,
        "Not answered"   = -9),
    missing = c(-9, -7),
    type = "cat",
    measurement = "nominal"
),
V2 = list(
    label = "Label for the second variable",
    values = c(
        "Very little"    =  1,
        "Little"         =  2,
        "So, so"         =  3,
        "Much"           =  4,
        "Very much"      =  5,
        "Don't know"     = -8),
    missing = c(-8),
    type = "cat",
    measurement = "ordinal"
),
V3 = list(
    label = "Label for the third variable",
    values = c(
        "First answer"   = "A",
        "Second answer"  = "B",
        "Don't know"     = -8),
    missing = c(-8),
    type = "cat",
    measurement = "nominal"
),
V4 = list(
    label = "Number of children",
    values = c(
        "Don't know"     = -8,
        "Not answered"   = -9),
    missing = c(-9, -8),
    type = "numcat",
    measurement = "ratio"
),
V5 = list(
    label = "Political party reference",
    type = "char",
   txt = "When the respondent indicated his political party reference, his/her open response
was recoded on a scale of 1-99 with parties with a left-wing orientation coded on the low end
of the scale and parties with a right-wing orientation coded on the high end of the scale.
Categories 90-99 were reserved miscellaneous responses."
)))

## Not run:
exportDDI(codeBook, file = "codebook.xml")

## End(Not run)
```

getMetadata *Extract metadata information*

Description

Extract a list containing the variable labels, value labels and any available information about missing values.

Usage

```
getMetadata(x, save = FALSE, OS = "Windows", ...)
```

Arguments

x	A path to a file, or a tibble object
save	Boolean, save an .R file in the same directory
OS	The target operating system, for the eol - end of line separator, if saving the file
...	Additional arguments for this function (internal uses only)

Details

This function reads an XML file containing a DDI codebook version 2.5, or an SPSS file and returns a list containing the variable labels, value labels, plus some other useful information.

It is also possible to extract a limited information from a Stata file, but especially the missing values are not yet imported by package **haven**.

It additionally attempts to automatically detect a type for each variable:

`cat:`	categorical
`num:`	numerical
`numcat:`	numerical variable with very few values (ex. number of children) for which a table of frequencies is possible in addition to frequencies

By default, this function extracts the metadata into an R list object, but when the argument `save` is activated, the argument `OS` (case insensitive) can be either:
`"Windows"` (default), or `"Win"`,
`"MacOS"`, `"Darwin"`, `"Apple"`, `"Mac"`,
`"Linux"`.

The end of line separator changes only when the target OS is different from the running OS.

For the moment, only DDI version 2.5 (Codebook) is supported, but DDI version 3.2 (Lifecycle) is planned to be implemented.

Value

A list containing all variables, their corresponding variable labels and value labels, and (if applicable) missing values if imported and found.

Author(s)

Adrian Dusa

`setupfile` *Create setup files for SPSS, Stata, SAS and R*

Description

This function creates a setup file, based on a list of variable and value labels.

Usage

```
setupfile(codeBook, file = "", type = "all", csv = "", OS = "", ...)
```

Arguments

`codeBook`	A list object containing the metadata, or a path to a directory where these objects are located, for batch processing
`file`	Character, the (path to the) setup file to be created
`type`	The type of setup file, can be: "SPSS", "Stata", "SAS", "R", or "all" (default)
`csv`	The original dataset, used to create the setup file commands, or a path to the directory where the .csv files are located, for batch processing
`OS`	The target operating system, for the eol - end of line character(s)
`...`	Other arguments, see Details below

Details

When the a path to a metadata directory is specified for the argument codebook, then next argument file is silently ignored and all created setup files are saved in a directory called "Setup Files" that (if not already found) is created in the working directory.

The argument file expects the name of the final setup file being saved on the disk. If not specified, the name of the object provided for the codebook argument will be used as a filename.

If file is specified, the argument type is automatically determined from the file's extension, otherwise when type = "all", the function produces one setup file for each supported type.

Missing values are expected to be supplied with the information object obj, otherwise the argument missing expects either:
- a vector of missing values (e.g. -1, -2, -3, if such values refer to missings throughout the entire dataset), or
- a vector of labels that should be interpreted as missings

If batch processing multiple files, the function will inspect all files in the provided directory, and retain only those with the extension .R or .r or DDI versions with the extension .xml or .XML (it will subsequently generate an error if the .R files do not contain an object list, or if the .xml files do not contain a DDI structured metadata file).

If the metadata directory contains a subdirectory called "data" or "Data", it will match the name of the metadata file with the name of the .csv file (their names have to be *exactly* the same, irrespective of their extension).

The csv argument can provide a data frame object produced by reading the .csv file, or a path to the directory where the .csv files are located. If the user doesn't provide something for this argument, the function will check the existence of a subdirectory called data in the directory where the metadata files are located.

In batch mode, the code starts with the argument delim = ",", but if the .csv file is delimited differently it will also try hard to find other delimiters that will match the variable names in the metadata file. At the initial version 0.1-0, the automatically detected delimiters include ";" and "\t".

The argument OS (case insensitive) can be either:
"Windows" (default), or "Win",
"MacOS", "Darwin", "Apple", "Mac",
"Linux".

The end of line character(s) changes only when the target OS is different from the running OS.

Value

A setup file to complement the imported raw dataset.

Examples

```
codeBook <- list(dataDscr = list(
ID = list(
    label = "Questionnaire ID",
    type = "num",
    measurement = "interval"
),
V1 = list(
    label = "Label for the first variable",
    values = c(
        "No"             =  0,
        "Yes"            =  1,
        "Not applicable" = -7,
        "Not answered"   = -9),
    missing = c(-9, -7),
    type = "cat",
    measurement = "nominal"
),
V2 = list(
    label = "Label for the second variable",
    values = c(
        "Very little"    =  1,
        "Little"         =  2,
        "So, so"         =  3,
        "Much"           =  4,
        "Very much"      =  5,
        "Don't know"     = -8),
    missing = c(-8),
    type = "cat",
    measurement = "ordinal"
),
V3 = list(
    label = "Label for the third variable",
    values = c(
        "First answer"   = "A",
        "Second answer"  = "B",
        "Don't know"     = -8),
    missing = c(-8),
    type = "cat",
    measurement = "nominal"
),
V4 = list(
    label = "Number of children",
    values = c(
        "Don't know"     = -8,
        "Not answered"   = -9),
    missing = c(-9, -8),
    type = "numcat",
    measurement = "ratio"
)))


## Not run:
# IMPORTANT:
# make sure to set the working directory to a directory with read/write permissions
# setwd("/path/to/read/write/directory")

# path.to.csv <- file.path(system.file(package = "DDIwR"), "data", "test.csv.gz")
```

```
setupfile(codeBook)


# if the csv data file is available
setupfile(codeBook, csv="/path/to/csv/file.csv")


# generating a specific type of setup file
setupfile(codeBook, file = "codeBook.do") # type = "Stata" is unnecessary


# other types of possible utilizations, using paths to specific files
# an XML file containing a DDI metadata object

setupfile("/path/to/the/metadata/file.xml", csv="/path/to/csv/file.csv")


# or in batch mode, specifying entire directories
setupfile("/path/to/the/metadata/directory", csv="/path/to/csv/directory")

## End(Not run)
```

Index