

Package ‘wizaRdry’

November 7, 2025

Title A Magical Framework for Collaborative & Reproducible Data Analysis

Version 0.5.0

Description

A comprehensive data analysis framework for NIH-funded research that streamlines workflows for both data cleaning and preparing and modifying NIH Data Archive ('NDA') data structures and submission templates. Provides unified access to multiple data sources ('REDCap', 'MongoDB', 'Qualtrics', 'SQL', 'ORACLE') through interfaces to their APIs, with specialized functions for data cleaning, filtering, merging, and parsing. Features automatic validation, field harmonization, and memory-aware processing to enhance reproducibility in multi-site collaborative research as described in Mittal et al. (2021) <[doi:10.20900/jpbs.20210011](https://doi.org/10.20900/jpbs.20210011)>.

License MIT + file LICENSE

Encoding UTF-8

RoxygenNote 7.3.2

Imports beeper, cli, config, dplyr, future, future.apply, haven, httr, jsonlite, knitr, mongolite, parallel, qualtrics, REDCapR, rlang, stringdist, testthat, rstudioapi, lubridate, DBI, RMariaDB, odbc, R6, openxlsx, openxlsx2

Depends R (>= 4.1.0)

URL <https://github.com/belieflab/wizaRdry>

BugReports <https://github.com/belieflab/wizaRdry/issues>

Suggests rmarkdown, yaml

NeedsCompilation no

Author Joshua G. Kenney [aut, cre],
Trevor F. Williams [aut],
Minerva K. Pappu [aut],
Michael J. Spilka [aut],
Danielle N. Pratt [ctb],
Victor J. Pokorny [ctb],
Santiago Castiello de Obeso [ctb],
Praveen Suthakaran [ctb],
Christian R. Horgan [ctb]

Maintainer Joshua G. Kenney <joshua.kenney@yale.edu>

Repository CRAN

Date/Publication 2025-11-07 18:20:02 UTC

Contents

clean	3
createCsv	4
createRds	4
createSpss	5
dataFilter	6
dataMerge	6
dataRequest	7
getRedcap	8
getSurvey	8
getTask	9
meld	10
mongo	11
mongo.index	12
mongo.rune	13
nda	14
ndaRequest	15
oracle	16
oracle.desc	17
oracle.index	18
oracle.query	18
oracle.test	19
qualtrics	19
qualtrics.dict	20
qualtrics.index	21
qualtrics.rune	21
redcap	22
redcap.dict	24
redcap.index	24
redcap.rune	25
rune	26
scry	27
sift	29
sql	32
sql.desc	33
sql.index	34
sql.query	34
to.csv	35
to.nda	36
to.rds	38
to.sav	39
wizaRdry-deprecated	40

clean	<i>Generate clean dataframes from cleaning scripts created in the ./clean directory</i>
-------	---

Description

This function processes requests for clean data sequentially for specified measures. It makes a request to the appropriate API for the named measure or measures and runs the associated data cleaning routines. It then runs a series of unit tests to verify that the data quality standards are met.

Usage

```
clean(..., csv = FALSE, rdata = FALSE, spss = FALSE, skip_prompt = TRUE)
```

Arguments

...	Strings, specifying the measures to process, which can be a Mongo collection, REDCap instrument, or Qualtrics survey.
csv	Optional; Boolean, if TRUE creates a .csv extract in ./tmp.
rdata	Optional; Boolean, if TRUE creates an .rdata extract in ./tmp.
spss	Optional; Boolean, if TRUE creates a .sav extract in ./tmp.
skip_prompt	Logical. If TRUE (default), skips confirmation prompts. If FALSE, prompts for confirmation unless the user has previously chosen to remember their preference.

Value

Prints the time taken for the data request process.

Author(s)

Joshua Kenney joshua.kenney@yale.edu

Examples

```
## Not run:
clean("prl", csv=TRUE)
clean("rgpts", "kamin", rdata=TRUE)

# Skip confirmation prompts
clean("prl", csv=TRUE, skip_prompt=TRUE)

## End(Not run)
```

createCsv	<i>Alias for 'to.csv' (DEPRECATED)</i>
-----------	--

Description

This function is deprecated. Please use 'to.csv' instead. This is a legacy alias for the 'to.csv' function to maintain compatibility with older code.

Usage

```
createCsv(...)
```

Arguments

... Additional arguments passed through to to.csv().

Value

Invisible TRUE if successful. The function writes a CSV file to the specified path and prints a message indicating the file's location.

Examples

```
## Not run:  
# DEPRECATED - use to.csv() instead  
createCsv(pr101)  
  
## End(Not run)
```

createRds	<i>Alias for 'to.rds' (DEPRECATED)</i>
-----------	--

Description

This function is deprecated. Please use 'to.rds' instead. This is a legacy alias for the 'to.rds' function to maintain compatibility with older code.

Usage

```
createRds(...)
```

Arguments

... Additional arguments passed through to to.rds().

Value

Invisible TRUE if successful. The function writes an RDS file to the specified path and prints a message indicating the file's location.

Examples

```
## Not run:  
# DEPRECATED - use to.rds() instead  
createRds(pr101)  
  
## End(Not run)
```

createSpss

Alias for 'to.sav' (DEPRECATED)

Description

This function is deprecated. Please use 'to.sav' instead. This is a legacy alias for the 'to.sav' function to maintain compatibility with older code.

Usage

```
createSpss(...)
```

Arguments

... Additional arguments passed through to to.sav().

Value

Invisible TRUE if successful. Writes an SPSS file to the designated path and prints a message indicating the file's location.

Examples

```
## Not run:  
# DEPRECATED - use to.sav() instead  
createSpss(pr101)  
  
## End(Not run)
```

dataFilter	<i>Alias for 'sift' (DEPRECATED)</i>
------------	--------------------------------------

Description

This function is deprecated. Please use 'sift' instead. This is a legacy alias for the 'sift' function to maintain compatibility with older code.

Usage

```
dataFilter(...)
```

Arguments

... Additional arguments passed through to sift().

Value

A filtered dataframe based on the provided parameters, and containing only the columns specified in 'cols'. If no columns are specified, returns the entire dataframe with applied row filters.

Examples

```
## Not run:  
# DEPRECATED - use sift() instead  
filtered <- dataFilter(df, sex="F")  
  
## End(Not run)
```

dataMerge	<i>Alias for 'meld' (DEPRECATED)</i>
-----------	--------------------------------------

Description

This function is deprecated. Please use 'meld' instead. This is a legacy alias for the 'meld' function to maintain compatibility with older code.

Usage

```
dataMerge(...)
```

Arguments

... Clean data frames to be merged.

Value

A merged data frame based on the specified or common candidate keys.

Examples

```
## Not run:  
# DEPRECATED - use meld() instead  
merged <- dataMerge(df1_clean, df2_clean)  
  
## End(Not run)
```

dataRequest	<i>Alias for 'clean' (DEPRECATED)</i>
-------------	---------------------------------------

Description

This function is deprecated. Please use 'clean' instead. This is a legacy alias for the 'clean' function to maintain compatibility with older code.

Usage

```
dataRequest(...)
```

Arguments

... Strings, specifying the measures to process, which can be a Mongo collection, REDCap instrument, or Qualtrics survey.

Value

Prints the time taken for the data request process.

Examples

```
## Not run:  
# DEPRECATED - use clean() instead  
pr1 <- dataRequest("pr1")  
  
## End(Not run)
```

`getRedcap`*Alias for 'redcap' (DEPRECATED)*

Description

This function is deprecated. Please use 'redcap' instead. This is a legacy alias for the 'redcap' function to maintain compatibility with older code.

Usage

```
getRedcap(...)
```

Arguments

... Optional column names to filter for. Only rows with non-missing values in ALL specified columns will be returned. This is useful for filtering data to only include complete cases for specific variables of interest.

Value

A data frame containing the requested REDCap data

Examples

```
## Not run:  
# DEPRECATED - use redcap() instead  
survey_data <- getRedcap("demographics")  
  
## End(Not run)
```

`getSurvey`*Alias for 'qualtrics' (DEPRECATED)*

Description

This function is deprecated. Please use 'qualtrics' instead. This is a legacy alias for the 'qualtrics' function to maintain compatibility with older code.

Usage

```
getSurvey(...)
```

Arguments

... Optional column names to filter for. Only rows with non-missing values in ALL specified columns will be returned. This is useful for filtering data to only include complete cases for specific variables of interest.

Value

A cleaned and harmonized data frame containing the survey data with superkeys first.

Examples

```
## Not run:  
# DEPRECATED - use qualtrics() instead  
survey_data <- getSurvey("your_survey_alias")  
  
## End(Not run)
```

getTask

Alias for 'mongo' (DEPRECATED)

Description

This function is deprecated. Please use 'mongo' instead. This is a legacy alias for the 'mongo' function to maintain compatibility with older code.

Usage

```
getTask(...)
```

Arguments

... Optional column names to filter for. Only rows with non-missing values in ALL specified columns will be returned. This is useful for filtering data to only include complete cases for specific variables of interest.

Value

A data frame containing the MongoDB data with superkeys first

Examples

```
## Not run:  
# DEPRECATED - use mongo() instead  
survey_data <- getTask("task_alias")  
  
## End(Not run)
```

meld	<i>Merge two or more data frames magically according to their candidate key</i>
------	---

Description

This function simplifies the process of merging multiple cleaned data frames by automatically determining common merge keys or utilizing user-specified keys. Supports both inner and outer join methods, and offers options for exporting the merged data.

Usage

```
meld(  
  ...,  
  by = NULL,  
  all = TRUE,  
  no.dups = FALSE,  
  csv = FALSE,  
  rdata = FALSE,  
  spss = FALSE  
)
```

Arguments

...	Clean data frames to be merged.
by	A vector of strings specifying the column names to be used as merge keys. If NULL, the function automatically determines common keys from the provided data frames.
all	Logical; if TRUE, performs an OUTER JOIN. If FALSE, performs an INNER JOIN.
no.dups	Logical; if TRUE, duplicates are removed post-merge.
csv	Logical; if TRUE, the merged data frame is exported as a CSV file.
rdata	Logical; if TRUE, the merged data frame is saved as an Rda file.
spss	Logical; if TRUE, the merged data frame is exported as an SPSS file.

Value

A merged data frame based on the specified or common candidate keys.

Author(s)

Joshua Kenney joshua.kenney@yale.edu

Examples

```
## Not run:
# Create sample dataframes for demonstration
df1 <- data.frame(
  src_subject_id = c("S001", "S002", "S003"),
  visit = c(1, 2, 1),
  measure1 = c(10, 15, 12),
  stringsAsFactors = FALSE
)

df2 <- data.frame(
  src_subject_id = c("S001", "S002", "S004"),
  visit = c(1, 2, 2),
  measure2 = c(85, 92, 78),
  stringsAsFactors = FALSE
)

# Perform an OUTER JOIN using default keys:
merged1 <- meld(df1, df2, all = TRUE)

# Perform an INNER JOIN using specified keys:
merged2 <- meld(df1, df2, by = "src_subject_id", all = FALSE)

## End(Not run)
```

mongo

*Fetch data from MongoDB to be stored in a data frame - UPDATED
VERSION*

Description

Fetch data from MongoDB to be stored in a data frame - UPDATED VERSION

Usage

```
mongo(
  collection,
  ...,
  database = NULL,
  identifier = NULL,
  chunk_size = NULL,
  verbose = FALSE,
  interview_date = NULL
)
```

Arguments

collection The name of the MongoDB collection

...	Optional column names to filter for. Only rows with non-missing values in ALL specified columns will be returned. This is useful for filtering data to only include complete cases for specific variables of interest.
database	The database name (optional)
identifier	Field to use as identifier (optional)
chunk_size	Number of records per chunk (optional)
verbose	Logical; if TRUE, displays detailed progress messages. Default is FALSE.
interview_date	Optional; can be either: - A date string in various formats (ISO, US, etc.) to filter data up to that date - A boolean TRUE to return only rows with non-NA interview_date values

Value

A data frame containing the MongoDB data with superkeys first

Examples

```
## Not run:
# Get data from MongoDB collection
data <- mongo("collection")

## End(Not run)
```

mongo.index	<i>Display table of available MongoDB collections</i>
-------------	---

Description

Retrieves a list of all available collections in the configured MongoDB database.

Usage

```
mongo.index(database = NULL)
```

Arguments

database	Optional; the name of the database to connect to. If NULL, uses the database specified in the configuration file.
----------	---

Value

A character vector containing the names of all available collections in the configured MongoDB database.

mongo.rune	<i>Parse composite MongoDB collection into component data frames by variable prefix</i>
------------	---

Description

This function fetches a MongoDB collection containing multiple collections and separates it into individual data frames for each collection detected in the data. It identifies the appropriate identifier column (e.g., participantId, workerId) and splits the data based on column name prefixes.

Usage

```
mongo.rune(collection, db_name = NULL, lower = TRUE)
```

Arguments

collection	Character string specifying the Mongo collection
db_name	Character string specifying the Mongo database
lower	default TRUE convert prefixes to lower case

Details

The function performs the following steps:

- Retrieves the raw Qualtrics data using the getSurvey() function
- Identifies which identifier column to use (participantId, workerId, PROLIFIC_PID, or src_subject_id)
- Determines survey prefixes by analyzing column names
- Creates separate dataframes for each survey prefix found
- Assigns each dataframe to the global environment with names matching the survey prefixes

Value

Creates multiple dataframes in the global environment, one for each survey detected in the data. Each dataframe is named after its survey prefix.

Examples

```
## Not run:  
# Parse a Qualtrics export containing multiple surveys  
mongo.rune("combined_surveys", label = FALSE)  
  
# After running, access individual survey dataframes directly:  
head(pss) # Access the PSS survey dataframe  
head(cesd) # Access the CESD survey dataframe  
  
## End(Not run)
```

nda	<i>Generate validated NDA submission templates created in the .nda directory</i>
-----	--

Description

This function processes requests for clean data sequentially for specified measures. It makes a request to the NIH NDA API for the named data structures and runs the associated data remediation routines. It then runs a series of unit tests to verify that the data quality standards are met.

Usage

```
nda(
  ...,
  csv = FALSE,
  rdata = FALSE,
  spss = FALSE,
  limited_dataset = FALSE,
  skip_prompt = TRUE
)
```

Arguments

...	Strings, specifying the measures to process, which can be a Mongo collection, REDCap instrument, or Qualtrics survey.
csv	Optional; Boolean, if TRUE creates a .csv extract in ./tmp.
rdata	Optional; Boolean, if TRUE creates an .rdata extract in ./tmp.
spss	Optional; Boolean, if TRUE creates a .sav extract in ./tmp.
limited_dataset	Optional; Boolean, if TRUE does not perform date-shifting of interview_date or age-capping of interview_age
skip_prompt	Logical. If TRUE (default), skips confirmation prompts unless preferences aren't set yet. If FALSE, prompts for confirmation unless the user has previously chosen to remember their preference.

Value

Prints the time taken for the data request process.

Author(s)

Joshua Kenney joshua.kenney@yale.edu

Examples

```
## Not run:
nda("pr1", csv=TRUE)
nda("rgpts", "kamin", rdata=TRUE)

# Skip confirmation prompts
nda("pr1", csv=TRUE, skip_prompt=TRUE)

## End(Not run)
```

ndaRequest	<i>Alias for 'nda' (DEPRECATED)</i>
------------	-------------------------------------

Description

This function is deprecated. Please use 'nda' instead. This is a legacy alias for the 'nda' function to maintain compatibility with older code.

Usage

```
ndaRequest(...)
```

Arguments

... Strings, specifying the measures to process, which can be a Mongo collection, REDCap instrument, or Qualtrics survey.

Value

Prints the time taken for the data request process.

Examples

```
## Not run:
# DEPRECATED - use nda() instead
pr101 <- ndaRequest("pr101")

## End(Not run)
```

oracle

*Fetch data from Oracle database to be stored in a data frame***Description**

Retrieves data from an Oracle table or view and optionally joins it with a primary keys table as specified in the configuration.

Usage

```
oracle(
  table_name = NULL,
  ...,
  fields = NULL,
  where_clause = NULL,
  join_primary_keys = TRUE,
  custom_query = NULL,
  max_rows = NULL,
  date_format = NULL,
  batch_size = 1000,
  pii = FALSE,
  interview_date = NULL,
  all = FALSE,
  schema = NULL
)
```

Arguments

<code>table_name</code>	Name of the SQL table or view to query
<code>...</code>	Optional column names to filter for. Only rows with non-missing values in ALL specified columns will be returned.
<code>fields</code>	Optional vector of specific fields to select
<code>where_clause</code>	Optional WHERE clause to filter results (without the "WHERE" keyword)
<code>join_primary_keys</code>	Boolean, whether to join with the primary keys table (default: TRUE)
<code>custom_query</code>	Optional custom SQL query to execute instead of building one
<code>max_rows</code>	Optional limit on the number of rows to return
<code>date_format</code>	Optional format for date fields (default uses ISO format)
<code>batch_size</code>	Number of records to retrieve per batch for large datasets
<code>pii</code>	Logical; if FALSE (default), remove fields marked as PII. TRUE keeps PII.
<code>interview_date</code>	Optional; can be either: - A date string in various formats (ISO, US, etc.) to filter data up to that date - A boolean TRUE to return only rows with non-NA interview_date values
<code>all</code>	Logical; if TRUE, use LEFT OUTER JOIN instead of INNER JOIN (default: FALSE), similar to the 'all' parameter in base R's merge() function
<code>schema</code>	Optional schema name to use for table qualification

Value

A data frame containing the requested SQL data

Examples

```
## Not run:
# Get data from a specific table
data <- oracle("participants")

# Get data with a where clause
survey_data <- oracle("vw_surveyquestionresults",
                      where_clause = "resultidentifier = 'NRS'")

# Get all records, including those without matching primary key
all_data <- oracle("candidate", all = TRUE)

# Specify schema explicitly
schema_data <- oracle("survey_results", schema = "STUDY_DATA")

## End(Not run)
```

oracle.desc

Get Oracle table columns/metadata

Description

Get Oracle table columns/metadata

Usage

```
oracle.desc(table_name, schema = NULL)
```

Arguments

table_name	Name of the table to get metadata for
schema	Optional schema name

Value

A data frame with column information

`oracle.index`*Get a list of tables from the Oracle database*

Description

Get a list of tables from the Oracle database

Usage

```
oracle.index(schema = NULL)
```

Arguments

`schema` Optional schema name to filter tables

Value

A data frame with table information

`oracle.query`*Perform a direct Oracle query with minimal processing*

Description

Perform a direct Oracle query with minimal processing

Usage

```
oracle.query(query, pii = FALSE, schema = NULL)
```

Arguments

`query` The SQL query to execute
`pii` Logical; if FALSE (default), remove fields marked as PII. TRUE keeps PII.
`schema` Optional schema name to qualify table names in the query

Value

A data frame with the query results

oracle.test	<i>Test Oracle database connection</i>
-------------	--

Description

Tests the connection to the Oracle database using the configured DSN and credentials. This is a simple connectivity test that doesn't perform any data operations.

Usage

```
oracle.test()
```

Value

A logical value indicating whether the connection was successful

Examples

```
## Not run:
# Test the Oracle connection
if (oracle.test()) {
  message("Oracle connection successful!")
} else {
  message("Oracle connection failed!")
}

## End(Not run)
```

qualtrics	<i>Retrieve Survey Data from Qualtrics</i>
-----------	--

Description

Retrieve Survey Data from Qualtrics

Usage

```
qualtrics(
  qualtrics_alias,
  ...,
  institution = NULL,
  label = FALSE,
  interview_date = NULL
)
```

Arguments

qualtrics_alias	The alias for the Qualtrics survey to be retrieved.
...	Optional column names to filter for. Only rows with non-missing values in ALL specified columns will be returned. This is useful for filtering data to only include complete cases for specific variables of interest.
institution	Optional. The institution name (e.g., "temple" or "nu"). If NULL, all institutions will be searched.
label	Logical indicating whether to return coded values or their associated labels (default is FALSE).
interview_date	Optional; can be either: - A date string in various formats (ISO, US, etc.) to filter data up to that date - A boolean TRUE to return only rows with non-NA interview_date values

Value

A cleaned and harmonized data frame containing the survey data with superkeys first.

Examples

```
## Not run:
# Get survey by alias (will search all institutions)
survey_data <- qualtrics("rgpts")

## End(Not run)
```

qualtrics.dict	<i>Fetch Qualtrics survey metadata to be stored in data frame</i>
----------------	---

Description

This function extracts column mappings from the metadata of a Qualtrics survey data frame. It can accept either a data frame containing Qualtrics data, a variable name as string, or a survey alias string.

Usage

```
qualtrics.dict(survey_alias, exclude_embedded = TRUE)
```

Arguments

survey_alias	Can either be an existing dataframe, variable name as string, or survey alias string
exclude_embedded	Only select QIDs

Value

A list containing the mappings of column names to survey questions.

qualtrics.index	<i>Display table of available Qualtrics surveys</i>
-----------------	---

Description

Retrieves a list of all available surveys from the Qualtrics API. Shows all surveys pulled down from Qualtrics, with alias and institution information merged from config.yml where available.

Usage

```
qualtrics.index(institution = NULL, all = FALSE)
```

Arguments

institution	Optional; the institution identifier to use. If NULL, uses all institutions specified in the configuration file (or all available credentials if no config).
all	Logical; deprecated parameter kept for backward compatibility. All surveys are now shown by default. Default is FALSE.

Value

A data frame containing the IDs and names of all available surveys from the Qualtrics API. Surveys with aliases configured in config.yml will show the alias and institution; unmapped surveys will show NA for these fields.

qualtrics.rune	<i>Parse composite Qualtrics survey into component data frames by variable prefix</i>
----------------	---

Description

This function fetches a Qualtrics data frame containing multiple surveys and separates it into individual data frames for each survey detected in the data. It identifies the appropriate identifier column (e.g., participantId, workerId) and splits the data based on column name prefixes.

Usage

```
qualtrics.rune(
  qualtrics_alias,
  institution = NULL,
  label = FALSE,
  interview_date = NULL,
  lower = TRUE
)
```

Arguments

<code>qualtrics_alias</code>	Character string specifying the Qualtrics survey alias to retrieve.
<code>institution</code>	Character string; default NULL, specify location
<code>label</code>	Logical; default TRUE, returns coded values as labels instead of raw values.
<code>interview_date</code>	Logical or Date String, returns all data before date
<code>lower</code>	default TRUE convert prefixes to lower case

Details

The function performs the following steps:

- Retrieves the raw Qualtrics data using the `getSurvey()` function
- Identifies which identifier column to use (`participantId`, `workerId`, `PROLIFIC_PID`, or `src_subject_id`)
- Determines survey prefixes by analyzing column names
- Creates separate dataframes for each survey prefix found
- Assigns each dataframe to the global environment with names matching the survey prefixes

Value

Creates multiple dataframes in the global environment, one for each survey detected in the data. Each dataframe is named after its survey prefix.

Examples

```
## Not run:
# Parse a Qualtrics export containing multiple surveys
qualtrics.rune("combined_surveys", label = FALSE)

# After running, access individual survey dataframes directly:
head(pss) # Access the PSS survey dataframe
head(cesd) # Access the CESD survey dataframe

## End(Not run)
```

redcap

Fetch data from REDCap to be stored in a data frame

Description

Retrieves data from a REDCap instrument and ensures subject identifiers are propagated across all events

Usage

```
redcap(
  instrument_name = NULL,
  ...,
  raw_or_label = "raw",
  redcap_event_name = NULL,
  batch_size = 1000,
  records = NULL,
  fields = NULL,
  pii = FALSE,
  interview_date = NULL,
  date_format = "ymd",
  complete = NULL
)
```

Arguments

<code>instrument_name</code>	Name of the REDCap instrument
<code>...</code>	Optional column names to filter for. Only rows with non-missing values in ALL specified columns will be returned. This is useful for filtering data to only include complete cases for specific variables of interest.
<code>raw_or_label</code>	Whether to return raw or labeled values
<code>redcap_event_name</code>	Optional event name filter
<code>batch_size</code>	Number of records to retrieve per batch
<code>records</code>	Optional vector of specific record IDs
<code>fields</code>	Optional vector of specific fields
<code>pii</code>	Logical; if FALSE (default), remove fields marked as PII. TRUE keeps PII.
<code>interview_date</code>	Optional; can be either: - A date string in various formats (ISO, US, etc.) to filter data up to that date - A boolean TRUE to return only rows with non-NA interview_date values
<code>date_format</code>	Default ymd define date format for interview_date
<code>complete</code>	Option boolean TRUE will return only forms marked as complete in REDCap

Value

A data frame containing the requested REDCap data

Examples

```
## Not run:
# Get data from a specific instrument
data <- redcap("demographics")

## End(Not run)
```

redcap.dict

Fetch REDCap data dictionary to be stored in data frame

Description

This function extracts metadata/dictionary information from REDCap. It can accept either an instrument name to fetch new data, an existing data frame with instrument attributes, or a variable name as string.

Usage

```
redcap.dict(instrument_name)
```

Arguments

instrument_name

Can either be an instrument name to fetch new data, a data frame returned by redcap(), or a variable name as string

Value

A data frame containing the data dictionary/metadata for the specified instrument

redcap.index

Display table of available REDCap instruments and their labels

Description

Retrieves a list of all available REDCap forms as a formatted table

Usage

```
redcap.index()
```

Value

A formatted table (kable) of available REDCap instruments/forms

redcap.rune	<i>Parse composite REDCap instrument into component data frames by variable prefix</i>
-------------	--

Description

This function fetches a REDCap instrument and separates it into individual data frames for each survey/collection detected in the data based on column name prefixes. It identifies the appropriate identifier column and splits the data accordingly.

Usage

```
redcap.rune(  
  instrument_name,  
  raw_or_label = "raw",  
  redcap_event_name = NULL,  
  batch_size = 1000,  
  records = NULL,  
  fields = NULL,  
  pii = FALSE,  
  interview_date = NULL,  
  date_format = "ymd",  
  lower = TRUE  
)
```

Arguments

instrument_name	Name of the REDCap instrument
raw_or_label	Whether to return raw or labeled values
redcap_event_name	Optional event name filter
batch_size	Number of records to retrieve per batch
records	Optional vector of specific record IDs
fields	Optional vector of specific fields
pii	Logical; if FALSE (default), remove fields marked as PII. TRUE keeps PII.
interview_date	Optional; date filtering parameter
date_format	Default ymd define date format for interview_date
lower	default TRUE convert prefixes to lower case

Value

Creates multiple dataframes in the parent environment, one for each survey detected in the data. Each dataframe is named after its survey prefix.

rune	<i>Parse composite data frame into component data frames by variable prefix</i>
------	---

Description

This function takes a data frame containing multiple measures and separates it into individual data frames for each measure detected in the data. It identifies the appropriate identifier column (e.g., participantId, workerId) and splits the data based on column name prefixes.

Usage

```
rune(df, lower = TRUE)
```

Arguments

df	a dataframe containing multiple, prefixed measures
lower	default TRUE convert prefixes to lower case

Details

The function performs the following steps:

- Identifies which identifier column to use (participantId, workerId, PROLIFIC_PID, or src_subject_id)
- Determines survey prefixes by analyzing column names
- Creates separate dataframes for each survey prefix found
- Assigns each dataframe to the global environment with names matching the survey prefixes

Value

Creates multiple dataframes in the global environment, one for each survey detected in the data. Each dataframe is named after its survey prefix.

Examples

```
# Parse a data frame containing multiple surveys
combined_df <- data.frame(
  record_id = c("REC001", "REC002", "REC003", "REC004"),
  src_subject_id = c("SUB001", "SUB002", "SUB003", "SUB004"),
  subjectkey = c("KEY001", "KEY002", "KEY003", "KEY004"),
  site = c("Yale", "NU", "Yale", "NU"),
  phenotype = c("A", "B", "A", "C"),
  visit = c(1, 2, 2, 1),
  state = c("complete", "completed baseline", "in progress", NA),
  status = c(NA, NA, NA, "complete"),
  lost_to_followup = c(FALSE, FALSE, TRUE, NA),
  interview_date = c("2023-01-15", "2023/02/20", NA, "2023-03-10"),
  foo_1 = c(1, 3, 5, 7),
```

```

foo_2 = c("a", "b", "c", "d"),
bar_1 = c(2, 4, 6, 8),
bar_2 = c("w", "x", "y", "z")
)
rune(combined_df)

# After running, access individual survey dataframes directly:
head(foo) # Access the foo dataframe
head(bar) # Access the bar dataframe

```

scry

Initialize the wizaRdry directory structure inside an R project

Description

Creates the standard directory structure required for the wizaRdry package to function properly. This includes folders for data cleaning scripts, NDA submission templates, and temporary outputs. It can detect and repair incomplete directory structures, and optionally create an R project.

Usage

```

scry(
  study_alias = NULL,
  path = ".",
  overwrite = FALSE,
  repair = FALSE,
  show_tree = NULL,
  create_project = FALSE,
  examples = FALSE,
  skip_prompt = TRUE
)

```

Arguments

study_alias	Character string specifying the short name for the study e.g. impact, capr, sing
path	Character string specifying the directory path where the structure should be created. Defaults to the current working directory.
overwrite	Logical. If TRUE, will overwrite existing files. If FALSE (default), will not replace existing files.
repair	Logical. If TRUE, will attempt to repair an incomplete directory structure. If FALSE, will abort with an error message when encountering an incomplete structure.
show_tree	Logical. If TRUE (default on first run), will display a visual file tree. Set to FALSE to suppress the tree view.
create_project	Logical. If TRUE, will create an R project file if one doesn't exist. If FALSE (default), will not create an R project.

examples	Logical. If TRUE (default when not repairing), will create example script templates. If FALSE (default when repairing), will skip creating example scripts.
skip_prompt	Logical. If TRUE (default), will skip the initial confirmation prompt if y/n preferences are not set yet. FALSE if specified.

Details

The function creates the following directory structure:

- clean/
 - csv/
 - mongo/
 - qualtrics/
 - redcap/
 - oracle/
 - sql/
- nda/
 - csv/
 - mongo/
 - qualtrics/
 - redcap/
 - oracle/
 - sql/
- tmp/

It also creates template config.yml and secrets.R files, and optionally an R project file.

Value

Invisible TRUE if successful.

Examples

```
## Not run:
# Initialize in current directory
scry()

# Repair structure in current directory
scry(repair = TRUE)

# Initialize in a specific directory with an R project
scry("path/to/project", create_project = TRUE, repair = TRUE)

# Skip the tree display
scry(repair = TRUE, show_tree = FALSE)

# Explicitly create example scripts when repairing
scry(repair = TRUE, examples = TRUE)
```

```
# Skip the confirmation prompt
scry(skip_prompt = TRUE)

## End(Not run)
```

sift

Filter data frame by superkey parameters, rows, and columns

Description

Filter data frame by superkey parameters, rows, and columns

Usage

```
sift(
  df,
  rows = NULL,
  cols = NULL,
  record_id = NULL,
  src_subject_id = NULL,
  subjectkey = NULL,
  site = NULL,
  subsiteid = NULL,
  sex = NULL,
  race = NULL,
  ethnic_group = NULL,
  phenotype = NULL,
  phenotype_description = NULL,
  status = NULL,
  lost_to_followup = NULL,
  twins_study = NULL,
  sibling_study = NULL,
  family_study = NULL,
  sample_taken = NULL,
  visit = NULL,
  week = NULL,
  arm = NULL,
  interview_date = NULL
)
```

Arguments

df	Dataframe to be filtered and trimmed based on the provided parameters.
rows	Optional; either a single row name or a vector of row names to be retained in the final output. If NULL or empty, all rows in the dataframe are retained.

cols	Optional; either a single column name or a vector of column names to be retained in the final output. If NULL or empty, all columns in the dataframe are retained.#' Data Filter
record_id	Optional; either a single record_id or a vector of record_ids to filter the dataframe by
src_subject_id	Optional; either a single subject ID or a vector of subject IDs to filter the dataframe by
subjectkey	Optional; either a single subjectkey or a vector of subjectkeys to filter the dataframe by
site	Optional; either a single site value or a vector of site values to filter the dataframe by (e.g., Yale, NU)
subsiteid	Optional; either a single subsiteid or a vector of subsiteids to filter the dataframe by
sex	Optional; either a single sex value or a vector of sex values at birth to filter the dataframe by (e.g., 'M', 'F')
race	Optional; either a single race value or a vector of race values to filter the dataframe by
ethnic_group	Optional; either a single ethnic_group value or a vector of ethnic_group values to filter the dataframe by
phenotype	Optional; either a single phenotype value or a vector of phenotype values to filter the dataframe by
phenotype_description	Optional; either a single phenotype_description or a vector of phenotype_descriptions to filter the dataframe by
status	Optional; either a single status string or a vector of status conditions to filter the dataframe by. Used if either 'state' or 'status' column exists in the dataframe. Can include values like 'complete', 'completed baseline', 'completed 12m', 'completed 24m', etc.
lost_to_followup	Optional; either a single value or a vector of values to filter the dataframe by (checks both 'lost_to_followup' and 'lost_to_follow-up' columns)
twins_study	Optional; either a single twins_study value or a vector of twins_study values to filter the dataframe by
sibling_study	Optional; either a single sibling_study value or a vector of sibling_study values to filter the dataframe by
family_study	Optional; either a single family_study value or a vector of family_study values to filter the dataframe by
sample_taken	Optional; either a single sample_taken value or a vector of sample_taken values to filter the dataframe by
visit	Optional; either a single visit value or a vector of visit values to filter the dataframe by. Only used if 'visit' column exists in the dataframe.
week	Optional; either a single week value or a vector of week values to filter the dataframe by. Only used if 'week' column exists in the dataframe.

arm Optional; either a single arm value or a vector of arm values to filter the dataframe by (e.g., drug, placebo)

interview_date Optional; can be either: - A date string in various formats (ISO, US, etc.) to filter data up to that date - A boolean TRUE to return only rows with non-NA interview_date values

Value

A filtered dataframe based on the provided parameters, and containing only the columns specified in 'cols'. If no columns are specified, returns the entire dataframe with applied row filters.

Examples

```
# Create a sample dataframe
sample_df <- data.frame(
  record_id = c("REC001", "REC002", "REC003", "REC004"),
  src_subject_id = c("SUB001", "SUB002", "SUB003", "SUB004"),
  subjectkey = c("KEY001", "KEY002", "KEY003", "KEY004"),
  site = c("Yale", "NU", "Yale", "NU"),
  phenotype = c("A", "B", "A", "C"),
  visit = c(1, 2, 2, 1),
  state = c("complete", "completed baseline", "in progress", NA),
  status = c(NA, NA, NA, "complete"),
  lost_to_followup = c(FALSE, FALSE, TRUE, NA),
  interview_date = c("2023-01-15", "2023/02/20", NA, "2023-03-10")
)

# Set row names for demonstration
rownames(sample_df) <- c("foo", "bar", "baz", "qux")

# Filter by specific date
filtered1 <- sift(sample_df,
  cols = c("src_subject_id", "phenotype"),
  visit = 2,
  interview_date = "01/31/2023")

# Filter to include only rows with non-NA interview dates
filtered2 <- sift(sample_df,
  interview_date = TRUE)

# Filter by status (works with either state or status column)
filtered3 <- sift(sample_df,
  status = c("complete", "completed baseline"))

# Filter with specific row names
filtered4 <- sift(sample_df,
  rows = c("foo", "qux"))

# Filter with vector of visit values
filtered6 <- sift(sample_df,
  visit = c(1, 2))
```

```
# Filter by lost_to_followup
filtered10 <- sift(sample_df,
  lost_to_followup = FALSE)

# Filter by src_subject_id
filtered11 <- sift(sample_df,
  src_subject_id = c("SUB001", "SUB004"))

# Multiple filters combined
filtered12 <- sift(sample_df,
  site = "Yale",
  visit = 1,
  cols = c("record_id", "src_subject_id", "site"))
```

sql

Fetch data from SQL database to be stored in a data frame

Description

Retrieves data from a SQL table and optionally joins it with a primary keys table as specified in the configuration.

Usage

```
sql(
  table_name = NULL,
  ...,
  fields = NULL,
  where_clause = NULL,
  join_primary_keys = TRUE,
  custom_query = NULL,
  max_rows = NULL,
  date_format = NULL,
  batch_size = 1000,
  pii = FALSE,
  interview_date = NULL,
  all = FALSE
)
```

Arguments

table_name	Name of the SQL table or view to query
...	Optional column names to filter for. Only rows with non-missing values in ALL specified columns will be returned.
fields	Optional vector of specific fields to select
where_clause	Optional WHERE clause to filter results (without the "WHERE" keyword)

join_primary_keys	Boolean, whether to join with the primary keys table (default: TRUE)
custom_query	Optional custom SQL query to execute instead of building one
max_rows	Optional limit on the number of rows to return
date_format	Optional format for date fields (default uses ISO format)
batch_size	Number of records to retrieve per batch for large datasets
pii	Logical; if FALSE (default), remove fields marked as PII. TRUE keeps PII.
interview_date	Optional; can be either: - A date string in various formats (ISO, US, etc.) to filter data up to that date - A boolean TRUE to return only rows with non-NA interview_date values
all	Logical; if TRUE, use LEFT OUTER JOIN instead of INNER JOIN (default: FALSE), similar to the 'all' parameter in base R's merge() function

Value

A data frame containing the requested SQL data

Examples

```
## Not run:
# Get data from a specific table
data <- sql("participants")

# Get data with a where clause
survey_data <- sql("vw_surveyquestionresults",
  where_clause = "resultidentifier = 'NRS'")

# Get all records, including those without matching primary key
all_data <- sql("candidate", all = TRUE)

## End(Not run)
```

sql.desc	<i>Get SQL table columns/metadata</i>
----------	---------------------------------------

Description

Get SQL table columns/metadata

Usage

```
sql.desc(table_name)
```

Arguments

table_name	Name of the table to get metadata for
------------	---------------------------------------

Value

A data frame with column information

sql.index	<i>Get a list of tables from the SQL database</i>
-----------	---

Description

Get a list of tables from the SQL database

Usage

```
sql.index(schema = NULL)
```

Arguments

schema Optional schema name to filter tables

Value

A data frame with table information

sql.query	<i>Perform a direct SQL query with minimal processing</i>
-----------	---

Description

Perform a direct SQL query with minimal processing

Usage

```
sql.query(query, pii = FALSE)
```

Arguments

query The SQL query to execute
pii Logical; if FALSE (default), remove fields marked as PII. TRUE keeps PII.

Value

A data frame with the query results

to.csv	Create .csv file from a data frame
--------	------------------------------------

Description

This function exports a given R data frame to a CSV file format. The resulting file is saved in the "tmp" directory. If a filename is not specified, the function uses the name of the data frame variable. The ".csv" extension is appended automatically to the filename. The function will prompt for confirmation before creating the file, with an option to remember the user's preference for future calls.

Usage

```
to.csv(df, df_name = NULL, path = ".", skip_prompt = TRUE)
```

Arguments

df	Data frame to be exported to CSV format.
df_name	Optional; a custom file name for the saved CSV file. If not provided, the name of the data frame variable is used. The function adds the ".csv" extension automatically.
path	Character string specifying the directory path where the "tmp" folder and CSV file should be created. Defaults to the current working directory.
skip_prompt	Logical. If TRUE (default), skips the confirmation prompt. If FALSE, will prompt for confirmation unless the user has previously chosen to remember their preference.

Value

Invisible TRUE if successful. The function writes a CSV file to the specified path and prints a message indicating the file's location.

Author(s)

Joshua Kenney joshua.kenney@yale.edu

Examples

```
## Not run:
# Create a sample data frame
sample_df <- data.frame(
  id = 1:3,
  name = c("Alice", "Bob", "Charlie")
)

# Basic usage with prompt
to.csv(sample_df)
```

```

# Custom filename
to.csv(sample_df, "participants_data")

# Skip the confirmation prompt
to.csv(sample_df, skip_prompt = TRUE)

# Save in a different directory
to.csv(sample_df, path = "path/to/project")

## End(Not run)

```

to.nda

Create NDA Submission Template

Description

This function creates a CSV template file for National Data Archive (NDA) submissions. It extracts the data from a specified data frame and formats it according to NDA requirements, with the structure name split into base name and suffix in the first line. The function will prompt for confirmation before creating the file, with an option to remember the user's preference for future calls.

This function creates a CSV template file for National Data Archive (NDA) submissions. It extracts the data from a specified data frame and formats it according to NDA requirements, with the structure name split into base name and suffix in the first line. The function will prompt for confirmation before creating the file, with an option to remember the user's preference for future calls.

Usage

```
to.nda(df, path = ".", skip_prompt = TRUE)
```

```
to.nda(df, path = ".", skip_prompt = TRUE)
```

Arguments

df	Data frame to be used as template or character string naming a data frame in the global environment.
path	Character string specifying the directory path where the "tmp" folder and template file should be created. Defaults to the current working directory.
skip_prompt	Logical. If TRUE (default), skips the confirmation prompt. If FALSE, will prompt for confirmation unless the user has previously chosen to remember their preference.

Details

The function will:

1. Create a 'tmp' directory if it doesn't exist
2. Parse the structure name into base and suffix components (e.g., "eefrt01" → "eefrt" and "01")
3. Write the structure name components as the first line
4. Write column headers as the second line
5. Write the data rows below

The function will:

1. Create a 'tmp' directory if it doesn't exist
2. Parse the structure name into base and suffix components (e.g., "eefrt01" → "eefrt" and "01")
3. Write the structure name components as the first line
4. Write column headers as the second line
5. Write the data rows below

Value

Invisible TRUE if successful. Creates a CSV file at the specified path and prints a message with the file location.

Invisible TRUE if successful. Creates a CSV file at the specified path and prints a message with the file location.

Examples

```
## Not run:
# First create some sample data
eefrt01 <- data.frame(
  src_subject_id = c("SUB001", "SUB002"),
  interview_age = c(240, 360),
  interview_date = c("01/01/2023", "02/15/2023"),
  response_time = c(450, 520)
)

# Create the NDA template using the data frame directly
to.nda(eefrt01)

# Or using the name as a string
to.nda("eefrt01")

# Skip the confirmation prompt
to.nda(eefrt01, skip_prompt = TRUE)

## End(Not run)

## Not run:
# First create some sample data
```

```
eefrt01 <- data.frame(
  src_subject_id = c("SUB001", "SUB002"),
  interview_age = c(240, 360),
  interview_date = c("01/01/2023", "02/15/2023"),
  response_time = c(450, 520)
)

# Create the NDA template using the data frame directly
to.nda(eefrt01)

# Or using the name as a string
to.nda("eefrt01")

# Skip the confirmation prompt
to.nda(eefrt01, skip_prompt = TRUE)

## End(Not run)
```

to.rds

Create .rds file from a data frame

Description

This function exports a given R data frame to an RDS file format. The resulting file is saved in the "tmp" directory. If a filename is not specified, the function uses the name of the data frame variable. The ".rds" extension is appended automatically to the filename. The function will prompt for confirmation before creating the file, with an option to remember the user's preference for future calls.

Usage

```
to.rds(df, df_name = NULL, path = ".", skip_prompt = TRUE)
```

Arguments

df	Data frame to be exported to RDS format.
df_name	Optional; a custom file name for the saved RDS file. If not provided, the name of the data frame variable is used. The function adds the ".rds" extension automatically.
path	Character string specifying the directory path where the "tmp" folder and RDS file should be created. Defaults to the current working directory.
skip_prompt	Logical. If TRUE (default), skips the confirmation prompt. If FALSE, will prompt for confirmation unless the user has previously chosen to remember their preference.

Value

Invisible TRUE if successful. The function writes an RDS file to the specified path and prints a message indicating the file's location.

Examples

```
## Not run:
# Create a sample data frame
sample_df <- data.frame(
  id = 1:3,
  name = c("Alice", "Bob", "Charlie")
)

# Basic usage with prompt
to.rds(sample_df)

# Custom filename
to.rds(sample_df, "participants_data")

# Skip the confirmation prompt
to.rds(sample_df, skip_prompt = TRUE)

# Save in a different directory
to.rds(sample_df, path = "path/to/project")

## End(Not run)
```

to.sav

Create .sav SPSS file from a data frame

Description

This function takes a R data frame and writes it to an SPSS file using the Haven package. The resulting file will be stored in the "tmp" directory with a default name derived from the data frame variable name, but can be customized if desired. The function will prompt for confirmation before creating the file, with an option to remember the user's preference for future calls.

Usage

```
to.sav(df, df_name = NULL, path = ".", skip_prompt = TRUE)
```

Arguments

df	Data frame to be exported to SPSS format.
df_name	Optional; custom file name for the saved SPSS file. If not provided, the name of the data frame variable will be used. The ".sav" extension will be appended automatically.

path	Character string specifying the directory path where the "tmp" folder and SPSS file should be created. Defaults to the current working directory.
skip_prompt	Logical. If TRUE (default), skips the confirmation prompt. If FALSE, will prompt for confirmation unless the user has previously chosen to remember their preference.

Value

Invisible TRUE if successful. Writes an SPSS file to the designated path and prints a message indicating the file's location.

Examples

```
## Not run:
# Create a sample data frame
sample_df <- data.frame(
  id = 1:3,
  score = c(85, 92, 78),
  group = c("A", "B", "A")
)

# Basic usage with prompt
to.sav(sample_df)

# Custom filename
to.sav(sample_df, "participants_data")

# Skip the confirmation prompt
to.sav(sample_df, skip_prompt = TRUE)

# Save in a different directory
to.sav(sample_df, path = "path/to/project")

## End(Not run)
```

wizaRdry-deprecated *Deprecated functions in wizaRdry*

Description

Deprecated functions in wizaRdry

Details

These functions are deprecated and may be removed in a future release. Prefer the suggested replacements.

Deprecated functions

`createCsv(...)` Use `to.csv(...)` instead.
`createRds(...)` Use `to.rds(...)` instead.
`createSpss(...)` Use `to.sav(...)` instead.
`dataFilter(...)` Use `sift(...)` instead.
`dataMerge(...)` Use `meld(...)` instead.
`dataRequest(...)` Use `clean(...)` instead.
`getRedcap(...)` Use `redcap(...)` instead.
`getSurvey(...)` Use `qualtrics(...)` instead.
`getTask(...)` Use `mongo(...)` instead.
`ndaRequest(...)` Use `nda(...)` instead.

See Also

`help("Deprecated")`

Index

`clean`, [3](#)
`createCsv`, [4](#)
`createRds`, [4](#)
`createSpss`, [5](#)

`dataFilter`, [6](#)
`dataMerge`, [6](#)
`dataRequest`, [7](#)

`getRedcap`, [8](#)
`getSurvey`, [8](#)
`getTask`, [9](#)

`meld`, [10](#)
`mongo`, [11](#)
`mongo.index`, [12](#)
`mongo.rune`, [13](#)

`nda`, [14](#)
`ndaRequest`, [15](#)

`oracle`, [16](#)
`oracle.desc`, [17](#)
`oracle.index`, [18](#)
`oracle.query`, [18](#)
`oracle.test`, [19](#)

`qualtrics`, [19](#)
`qualtrics.dict`, [20](#)
`qualtrics.index`, [21](#)
`qualtrics.rune`, [21](#)

`redcap`, [22](#)
`redcap.dict`, [24](#)
`redcap.index`, [24](#)
`redcap.rune`, [25](#)
`rune`, [26](#)

`scry`, [27](#)
`sift`, [29](#)
`sql`, [32](#)

`sql.desc`, [33](#)
`sql.index`, [34](#)
`sql.query`, [34](#)

`to.csv`, [35](#)
`to.nda`, [36](#)
`to.rds`, [38](#)
`to.sav`, [39](#)

`wizaRdry-deprecated`, [40](#)