Package 'wizaRdry'

June 18, 2025

Title A Magical Framework for Collaborative & Reproducible Data Analysis

Version 0.2.6

Description A comprehensive data analysis framework for NIH-funded research that streamlines workflows for both data cleaning and preparing NIH Data Archive ('NDA') submission templates. Provides unified access to multiple data sources ('REDCap', 'MongoDB', 'Qualtrics') 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>.

License MIT + file LICENSE

Encoding UTF-8

RoxygenNote 7.3.2

Imports beepr, cli, config, dplyr, future, future.apply, haven, httr, jsonlite, knitr, mongolite, parallel, qualtRics, R6, REDCapR, rlang, stringdist, testthat, rstudioapi, lubridate

Depends R (>= 4.1.0)

URL `https://github.com/belieflab/wizaRdry`

BugReports `https://github.com/belieflab/wizaRdry/issues`

Suggests rmarkdown

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 Suthaharan [ctb],
Christian R. Horgan [ctb]

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

Repository CRAN

Date/Publication 2025-06-18 00:30:02 UTC

Contents

clean 2
createCsv 4
createRds 4
createSpss 5
dataFilter 6
dataMerge 8
dataRequest 9
getRedcap 10
getSurvey 11
getTask 12
meld 13
mongo 14
mongo.index 15
mongo.rune 16
nda 17
ndaRequest 18
qualtrics 19
qualtrics.dict 20
qualtrics.index 20
qualtrics.rune 21
redcap 22
redcap.dict 23
redcap.index 24
redcap.rune 24
rune 25
scry 26
sift 28
to.csv 31
to.nda 32
to.rds 34
to.sav 35

Index **37**

`clean` *Generate clean data frames from cleaning scripts created in the ./clean directory*

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` *Alias for 'to.csv'*

Description

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

Usage

```
createCsv(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.

Examples

```
## Not run:
createCsv(prl01)

## End(Not run)
```

`createRds` *Alias for 'to.rds'*

Description

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

Usage

```
createRds(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:
createRds(prl01)

## End(Not run)
```

`createSpss` *Alias for 'to.sav'*

Description

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

Usage

```
createSpss(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:
createSpss(prl01)

## End(Not run)
```

---

dataFilter *Alias for 'sift'*

Description

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

Usage

```
dataFilter(
  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

```
## Not run:
filtered <- dataFilter(df, sex="F")

## End(Not run)
```

`dataMerge` *Alias for 'meld'*

Description

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

Usage

```
dataMerge(
  ...,
  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.

Examples

```
## Not run:
merged <- dataMerge(df1_clean, df2_clean)

## End(Not run)
```

`dataRequest` *Alias for 'clean'*

Description

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

Usage

```
dataRequest(..., 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.

Examples

```
## Not run:
prl <- clean("prl")

## End(Not run)
```

`getRedcap` *Alias for 'redcap'*

Description

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

Usage

```
getRedcap(
  instrument_name = NULL,
  ...,
  raw_or_label = "raw",
  redcap_event_name = NULL,
  batch_size = 1000,
  records = NULL,
  fields = NULL,
  exclude_pii = TRUE,
  interview_date = NULL,
  date_format = "ymd",
  complete = NULL
)
```

Arguments

`instrument_name`	Name of the REDCap instrument
`...`	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.
`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
`exclude_pii`	Default TRUE remove all fields marked as identifiable
`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

`date_format`	Default ymd define date format for interview_date
`complete`	Option boolean TRUE will return only forms marked as complete in REDCap

Value

A data frame containing the requested REDCap data

Examples

```
## Not run:
survey_data <- getRedcap("demographics")

## End(Not run)
```

`getSurvey` *Alias for 'qualtrics'*

Description

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

Usage

```
getSurvey(
  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:
survey_data <- getSurvey("your_survey_alias")

## End(Not run)
```

---

getTask *Alias for 'mongo'*

Description

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

Usage

```
getTask(
  collection_name,
  ...,
  db_name = NULL,
  identifier = NULL,
  chunk_size = NULL,
  verbose = FALSE,
  interview_date = NULL
)
```

Arguments

`collection_name`	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.
`db_name`	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:
survey_data <- getTask("task_alias")

## End(Not run)
```

`meld`	*Merge two or more data frames magically according to their candidate key*

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*

Description

Fetch data from MongoDB to be stored in a data frame

Usage

```
mongo(
  collection_name,
  ...,
  db_name = NULL,
  identifier = NULL,
  chunk_size = NULL,
  verbose = FALSE,
  interview_date = NULL
)
```

Arguments

`collection_name`	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.
`db_name`	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_name")

## End(Not run)
```

`mongo.index` *Display table of available MongoDB collections*

Description

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

Usage

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

Arguments

`db_name`	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` *Parse composite MongoDB collection into component data frames by variable prefix*

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 — *Generate validated NDA submission templates created in the ./nda directory*

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("prl", csv=TRUE)
  nda("rgpts", "kamin", rdata=TRUE)

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

## End(Not run)
```

`ndaRequest` *Alias for 'nda'*

Description

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

Usage

```
ndaRequest(
  ...,
  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.

Examples

```
## Not run:
prl01 <- ndaRequest("prl01")

## End(Not run)
```

`qualtrics` *Retrieve Survey Data from Qualtrics*

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` *Fetch Qualtrics survey metadata to be stored in data frame*

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` *Display table of available Qualtrics surveys*

Description

Retrieves a list of all available surveys in the configured Qualtrics account.

Usage

```
qualtrics.index(institution = NULL)
```

Arguments

`institution`	Optional; the institution identifier to use. If NULL, uses all institutions specified in the configuration file.

Value

A data frame containing the IDs and names of all available surveys in the configured Qualtrics account. Can be used to identify surveys for further data retrieval.

`qualtrics.rune` *Parse composite Qualtrics survey into component data frames by variable prefix*

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

`qualtrics_alias`	Character string specifying the Qualtrics survey alias to retrieve.
`institution`	Character string; default NULL, specify location
`label`	Logical; default TRUE, returns coded values as labels instead of raw values.
`interview_date`	Logical or Date String, returns all data before date
`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
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,
  exclude_pii = TRUE,
  interview_date = NULL,
  date_format = "ymd",
  complete = NULL
)
```

Arguments

`instrument_name`	Name of the REDCap instrument
`...`	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.

`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
`exclude_pii`	Default TRUE remove all fields marked as identifiable
`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
`date_format`	Default ymd define date format for interview_date
`complete`	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` *Parse composite REDCap instrument into component data frames by variable prefix*

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,
  exclude_pii = TRUE,
  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
`exclude_pii`	Default TRUE remove all fields marked as identifiable
`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` *Parse composite data frame into component data frames by variable prefix*

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/
  - sql/
- nda/
  - csv/
  - mongo/
  - qualtrics/
  - redcap/
  - 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"))
```

`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.

Usage

```
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" $\rightarrow$ "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.

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)
```

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)
```

Index

clean, 2
createCsv, 4
createRds, 4
createSpss, 5

dataFilter, 6
dataMerge, 8
dataRequest, 9

getRedcap, 10
getSurvey, 11
getTask, 12

meld, 13
mongo, 14
mongo.index, 15
mongo.rune, 16

nda, 17
ndaRequest, 18

qualtrics, 19
qualtrics.dict, 20
qualtrics.index, 20
qualtrics.rune, 21

redcap, 22
redcap.dict, 23
redcap.index, 24
redcap.rune, 24
rune, 25

scry, 26
sift, 28

to.csv, 31
to.nda, 32
to.rds, 34
to.sav, 35