Package: cleaningtools (via r-universe)

October 17, 2024

Title cleaningtools package focuses on data cleaning

Version 0.0.0.9003

Description The cleaningtools package focuses on cleaning, and has three components: **Check**, which includes a set of functions that flag values, such as check_outliers and check_logical. **Create**, which includes a set of functions to create different items for use in cleaning, such as the cleaning log from the checks, clean data, and enumerator performance. **Review**, which includes a set of functions to review the cleaning.

License MIT + file LICENSE

URL https://impact-initiatives.github.io/cleaningtools/

Depends R (>= 4.0)

Imports assertthat, cluster, data.table, dplyr, fs, glue, here, httr, magrittr, openxlsx, purrr, randomcoloR, rlang, snakecase, stats, stringi, stringr, tibble, tidyr, tidyselect, utils

Suggests knitr, riddle, rmarkdown, testthat (>= 3.0.0), tidyverse

VignetteBuilder knitr

Remotes edouard-legoupil/riddle

Config/fusen/version 0.5.2

Config/testthat/edition 3

Encoding UTF-8

LazyData true

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.1

Repository https://humanitarian-user-group.r-universe.dev

RemoteUrl https://github.com/impact-initiatives/cleaningtools

RemoteRef HEAD

RemoteSha 954aa39b07f224ac51644e0801115da837538050

Contents

`add_duration` *Creates a duration variable using the start and end time of the survey*

Description

Creates a duration variable using the start and end time of the survey

Usage

```
add_duration(
  dataset,
  duration_column = "duration",
  uuid_column = "uuid",
  start_column = "start",
  end_column = "end"
)
```

Arguments

`dataset`	dataset to add the duration
`duration_column`	the name of the duration variable that will be created.
`uuid_column`	uuid column in the dataset. Default is uuid.
`start_column`	start time of the survey, in KOBO format (e.g "2021-07-14T14:02:24.955+03:00")
`end_column`	end time of the survey, in KOBO format (e.g "2021-07-14T14:03:28.955+03:00")

Value

the clean dataset with 6 new columns: duration, start_date, start_time, end_date, end_time, days_diff

Examples

```
test_data <- data.frame(
  start = c(
    "2021/07/13 11:25:49", "2021/07/13 12:36:16",
    "2021/07/13 10:21:10", "2021/07/13 10:54:07", "2021/07/13 11:18:45"
  ),
  end = c(
    "2021/07/13 12:02:39", "2021/07/13 13:20:17",
    "2021/07/13 10:53:42", "2021/07/13 11:28:58", "2021/07/13 11:55:24"
  ),
  uuid = 1:5
```

```
)

test_data_w_duration <- test_data %>%
  add_duration()

test_data_kobo_time <- data.frame(
  `X.U.FEFF.start` = c(
    "2021-07-13T11:25:49.543+03:00",
    "2021-07-13T12:36:16.316+03:00",
    "2021-07-13T10:21:10.337+03:00",
    "2021-07-13T10:54:07.394+03:00",
    "2021-07-13T11:18:45.521+03:00"
  ),
  end = c(
    "2021-07-14T12:02:39.269+03:00",
    "2021-07-13T13:20:17.815+03:00",
    "2021-07-13T10:53:42.662+03:00",
    "2021-07-13T11:28:58.295+03:00",
    "2021-07-13T11:55:24.366+03:00"
  ),
  uuid_column = 1:5
)
test_data_kobo_time_w_duration <- test_data_kobo_time %>%
  add_duration(start_column = "X.U.FEFF.start")
```

`add_duration_from_audit` *Adds duration from the audit file*

Description

Wrapper around create_duration_from_audit_with_start_end and create_duration_from_audit_sum_all to add the duration to the dataset.

Usage

```
add_duration_from_audit(
  dataset,
  col_name_prefix = "duration_audit",
  uuid_column = "uuid",
  audit_list,
  start_question = NULL,
  end_question = NULL,
  sum_all = TRUE
)
```

Arguments

dataset	dataset to add the duration
col_name_prefix	string character to be used for the prefix of new columns
uuid_column	uuid column in the dataset. Default is uuid.
audit_list	list of dataframe that are the audit file with the uuid of each interview as name of the dataframe.
start_question	character vector use for the starting question (optional)
end_question	character vector use for the ending question (optional)
sum_all	TRUE or FALSE if to add the time with sum all duration

Value

return the dataset with durations column added

Examples

```
list_audit <- list(
  uuid1 = data.frame(
    event = c("form start", rep("question", 5)),
    node = c("", paste0("/xx/question", 1:5)),
    start = c(
      1661415887295, 1661415887301,
      1661415890819, 1661415892297,
      1661415893529, 1661415894720
    ),
    end = c(
      NA, 1661415890790, 1661415892273,
      1661415893506, 1661415894703,
      1661415896452
    )
  ),
  uuid2 = data.frame(
    event = c("form start", rep("question", 5)),
    node = c("", paste0("/xx/question", 1:5)),
    start = c(
      1661415887295, 1661415887301, 1661415890819,
      1661415892297, 1661415893529, 1661415894720
    ),
    end = c(
      NA, 1661415890790, 1661415892273,
      1661415893506, 1661415894703, 1661415896452
    )
  )
)
some_dataset <- data.frame(
  X_uuid = c("uuid1", "uuid2"),
  question1 = c("a", "b"),
  question2 = c("a", "b"),
  question3 = c("a", "b"),
```

```
  question4 = c("a", "b"),
  question5 = c("a", "b")
)
add_duration_from_audit(some_dataset, uuid_column = "X_uuid", audit_list = list_audit)
add_duration_from_audit(some_dataset,
  uuid_column = "X_uuid", audit_list = list_audit,
  start_question = "question1",
  end_question = "question3",
  sum_all = FALSE
)
add_duration_from_audit(some_dataset,
  uuid_column = "X_uuid", audit_list = list_audit,
  start_question = "question1",
  end_question = "question3",
  sum_all = TRUE
)
```

`add_info_to_cleaning_log` *Add information to the cleaning log*

Description

Add information to the cleaning log

Usage

```
add_info_to_cleaning_log(
  list_of_log = NULL,
  dataset = "checked_dataset",
  cleaning_log = "cleaning_log",
  dataset_uuid_column = "uuid",
  cleaning_log_uuid_column = "uuid",
  information_to_add = c("enumerator_num", "date_assessment")
)
```

Arguments

`list_of_log`	A list file containing the cleaning_log and the dataset
`dataset`	If a list is provided, the element name represents the dataset; otherwise, it should be a dataframe.
`cleaning_log`	If a list is provided, the element name represents the cleaning log; otherwise, it is should be a dataframe.
`dataset_uuid_column`	uuid column in the cleaning dataset, it can take a vector of several binding key. Default is uuid.

cleaning_log_uuid_column
: uuid column in the cleaning log, it can take a vectore of several binding key. Default is uuid.

information_to_add
: Vector of name of the columns to be added.

Value

A list with cleaning log with added info and the dataset.

Examples

```
cleaningtools::cleaningtools_raw_data |>
  check_pii(uuid_column = "X_uuid") |>
  check_duplicate(uuid_column = "X_uuid") |>
  check_value(uuid_column = "X_uuid") |>
  create_combined_log() |>
  add_info_to_cleaning_log(dataset_uuid_column = "X_uuid")
```

---

add_percentage_missing

Adds the percentage of missing values per row

Description

Adds the percentage of missing values per row

Usage

```
add_percentage_missing(
  dataset,
  column_name = "percentage_missing",
  kobo_survey = NULL,
 type_to_include = c("integer", "date", "text", "select_one", "select_multiple")
)
```

Arguments

dataset
: A dataset to add the percentage of missing values

column_name
: string variable with the name of the new column to be created, default is percentage_missing

kobo_survey
: Kobo survey sheet.

type_to_include
: Types (from KOBO) to be included in the columns default are integer, date, text, select_one and select_multiple.

Value

a dataset with the one additional column with the percentage of missing value

See Also

check_percentage_missing()

Examples

```
data_test <- data.frame(
  uuid = c(1:3),
  col_1 = c(1:3),
  col_2 = c(NA, NA, "expenditures"),
  col_3 = c("with need", NA, "with need"),
  col_4 = c("food health school", NA, "food"),
  col_4.food = c(1, NA, 1),
  col_4.health = c(1, NA, 0),
  col_4.school = c(1, NA, 0)
)
kobo_survey <- data.frame(
  type = c(
    "uuid",
    "integer",
    "select_one choice2",
    "select_one choice3",
    "select_multiple choice4"
  ),
  name = c("uuid", "col_1", "col_2", "col_3", "col_4")
)
data_test %>% add_percentage_missing(
  kobo_survey = kobo_survey,
  type_to_include = c("integer", "select_one", "select_multiple")
)
data_test %>% add_percentage_missing()
```

---

auto_detect_sm_parents

Detect select multiple parent columns

Description

auto_detect_sm_parents is mean to detect select multiple parent columns in a way that does not rely on the XLSForm as the input

Usage

```
auto_detect_sm_parents(dataset, sm_separator = ".")
```

Arguments

dataset	dataset to correct
sm_separator	Separator for choice multiple questions. The default is "." If using read_csv to read in data the separator will most likely be '/' where as if using read.csv it will likely be '.'

Value

a list of select multiple parent columns in data set.

---

auto_sm_parent_child *detect and group together select multiple parent and children columns*

Description

auto_sm_parent_children is mean to detect select multiple parent columns & children columns in a way that does not rely on the XLSForm as the input

Usage

```
auto_sm_parent_children(dataset, sm_separator = ".")
```

Arguments

dataset	dataset to correct
sm_separator	Separator for choice multiple questions. The default is "." If using read_csv to read in data the separator will most likely be '/' where as if using read.csv it will likely be '.'

Value

a data frame containing the the child select multiple columns alongside there parents and a log with all changes recorded.

check_duplicate *Checks for duplicated values in columns*

Description

Checks for duplicated values in columns

Usage

```
check_duplicate(
  dataset,
  uuid_column = "uuid",
  columns_to_check = NULL,
  log_name = "duplicate_log"
)
```

Arguments

dataset	dataset to be check as a dataframe or a list with the dataframe stored as "checked_dataset".
uuid_column	uuid column in the dataset. Default is uuid.
columns_to_check	string character with the name of the columns to check. If NULL (default), it will check for the uuid_column
log_name	name of the log of flagged value

Value

return a list with the dataset checked stored as checked_dataset and a dataframe with the duplicate log

Examples

```
testdata <- data.frame(
  uuid = c(letters[1:4], "a", "b", "c"),
  col_a = runif(7),
  col_b = runif(7)
)

check_duplicate(testdata)

testdata2 <- data.frame(
  uuid = letters[c(1:7)],
  village = paste("village", c(1:3, 1:3, 4)),
  ki_identifier = paste0("xx_", c(1:5, 3, 4))
)

check_duplicate(testdata2, columns_to_check = "village")
```

```
check_duplicate(testdata2, columns_to_check = c("village", "ki_identifier"), uuid = "uuid")
```

---

check_duration *Check if duration is outside of a range*

Description

Check if value is strictly inferior of the lower threshold or strictly superior of the higher threshold.

Usage

```
check_duration(
  dataset,
  column_to_check,
  uuid_column = "uuid",
  log_name = "duration_log",
  lower_bound = 25,
  higher_bound = 90
)
```

Arguments

dataset	a dataset to be check as a dataframe or a list with the dataframe stored as "checked_dataset"
column_to_check	string character with the name of the duration column
uuid_column	uuid column in the dataset. Default is uuid.
log_name	character string with name to give to the log
lower_bound	lower value of the range (strictly inferior to)
higher_bound	higher value of the range (strictly superior to)

Value

return a list with the dataset checked stored as checked_dataset and a dataframe with the duration log

Examples

```
testdata <- data.frame(
  uuid = c(letters[1:7]),
  duration_audit_start_end_ms = c(
    2475353, 375491, 2654267, 311585, 817270,
    2789505, 8642007
  ),
  duration_audit_start_end_minutes = c(41, 6, 44, 5, 14, 46, 144)
)
```

```
check_duration(testdata,
  column_to_check = "duration_audit_start_end_minutes"
)

check_duration(
  testdata,
  column_to_check = "duration_audit_start_end_ms",
  lower_bound = 375490,
  higher_bound = 8642000
)

testdata %>%
  check_duration(column_to_check = "duration_audit_start_end_minutes") %>%
  check_duration(
    column_to_check = "duration_audit_start_end_ms",
    log_name = "duration_in_ms",
    lower_bound = 375490,
    higher_bound = 8642000
  )
```

`check_fcs` *FCS component checks*

Description

FCS component checks

Usage

```
check_fcs(
  dataset,
  uuid_column = "uuid",
  cereals_column = "fs_fcs_cerealgrainroottuber",
  pulses_column = "fs_fcs_beansnuts",
  dairy_column = "fs_fcs_dairy",
  meat_column = "fs_fcs_meatfishegg",
  vegetables_column = "fs_fcs_vegetableleave",
  fruits_column = "fs_fcs_fruit",
  oil_column = "fs_fcs_fat",
  sugar_column = "fs_fcs_sugar"
)
```

Arguments

`dataset`	dataset to be check as a dataframe or a list with the dataframe stored as "checked_dataset"
`uuid_column`	uuid column in the dataset. Default is uuid.
`cereals_column`	Variable name for- In the last 7 days, on how many days did your household eat cereals, grains, roots and tubers, including wild roots?

pulses_column	Variable name for- In the last 7 days, on how many days did your household eat any beans / legumes, pulses or nuts?
dairy_column	Variable name for- In the last 7 days, on how many days did your household drink milk or eat other dairy products?
meat_column	Variable name for- In the last 7 days, on how many days did your household eat meat, fish or eggs?
vegetables_column	Variable name for- In the last 7 days, on how many days did your household eat vegetables or leaves, including all wild vegetables and leaves?
fruits_column	Variable name for- In the last 7 days, on how many days did your household eat fruits, including all wild fruits?
oil_column	Variable name for- In the last 7 days, on how many days did your household eat oil, fat, or butter?
sugar_column	Variable name for- In the last 7 days, on how many days did your household eat sugar or sugary foods?

Value

dataset with potential issues

`check_logical` *Check a logical test*

Description

Check a logical test

Usage

```
check_logical(
  dataset,
  uuid_column = "uuid",
  information_to_add = NULL,
  check_id = "logical_xx",
  check_to_perform,
  columns_to_clean = NULL,
  description
)
```

Arguments

dataset	dataset to be check as a dataframe or a list with the dataframe stored as "checked_dataset"
uuid_column	uuid column in the dataset. Default is "uuid".
information_to_add	string vector optional, if to add some information to the log (today, vilagge)

check_id name of the check to perform

check_to_perform
test to perform as R code (in text format)

columns_to_clean
variables to be put in the log. if not provided, it will try to detect variables

description description of the check performed

Value

return a list with the dataset checked stored as checked_dataset, it will have all a column added with the check_id and a dataframe with the logical check log.

Examples

```
test_data <- data.frame(
  uuid = c(1:10) %>% as.character(),
  today = rep("2023-01-01", 10),
  location = rep(c("villageA", "villageB"), 5),
  distance_to_market = c(rep("less_30", 5), rep("more_30", 5)),
  access_to_market = c(rep("yes", 4), rep("no", 6)),
  number_children_05 = c(rep(c(0, 1), 4), 5, 6)
)

check_logical(test_data,
  uuid_column = "uuid",
  check_id = "my_test",
  check_to_perform = "distance_to_market == \"less_30\" & access_to_market == \"no\"",
  columns_to_clean = "distance_to_market, access_to_market",
  description = "distance to market less than 30 and no access"
)

check_logical(test_data,
  uuid_column = "uuid",
  information_to_add = c("today", "location"),
  check_to_perform = "distance_to_market == \"less_30\" & access_to_market == \"no\"",
  columns_to_clean = "distance_to_market, access_to_market",
  description = "distance to market less than 30 and no access"
)
```

---

check_logical_with_list

Check several logical test

Description

This is a wrapper around check_logical to allow several checks to be performed.

Usage

```
check_logical_with_list(
  dataset,
  uuid_column = "uuid",
  information_to_add = NULL,
  list_of_check,
  check_id_column,
  check_to_perform_column,
  columns_to_clean_column = NULL,
  description_column,
  bind_checks = TRUE
)
```

Arguments

dataset	dataset to be check as a dataframe or a list with the dataframe stored as "checked_dataset"
uuid_column	uuid column in the dataset. Default is "uuid".
information_to_add	string vector optional, if to add some information to the log (today, vilagge)
list_of_check	a dataframe with the checks to perform
check_id_column	the column name with the names of each test.
check_to_perform_column	the column name with the checks to perform as R code (in text format)
columns_to_clean_column	the column name with the variables to be put in the log.
description_column	the column name with the description
bind_checks	default is TRUE, to bind the checks into 1 log.

Value

return a list with the dataset checked stored as checked_dataset, it will have all a column added with the check_id and a dataframe with the logical check log.

Examples

```
test_data <- data.frame(
  uuid = c(1:10) %>% as.character(),
  distance_to_market = rep(c("less_30", "more_30"), 5),
  access_to_market = c(rep("yes", 4), rep("no", 6)),
  number_children_05 = c(rep(c(0, 1), 4), 5, 6),
  number_children_618 = c(rep(c(0, 1), 4), 5, 6)
)
check_list <- data.frame(
  name = c("logical_xx", "logical_yy", "logical_zz"),
  check = c(
    "distance_to_market == \"less_30\" & access_to_market == \"no\"",
```

```
    "number_children_05 > 3",
    "rowSums(dplyr::across(starts_with(\"number\")), na.rm = T) > 9"
  ),
  description = c(
    "distance to market less than 30 and no access",
    "number of children under 5 seems high",
    "number of children very high"
  ),
  columns_to_clean = c(
    "distance_to_market, access_to_market",
    "number_children_05",
    ""
  )
)
check_logical_with_list(test_data,
  uuid_column = "uuid",
  list_of_check = check_list,
  check_id_column = "name",
  check_to_perform_column = "check",
  columns_to_clean_column = "columns_to_clean",
  description_column = "description"
)
```

`check_others` *Generate a log for other follow up questions*

Description

Generate a log for other follow up questions

Usage

```
check_others(dataset, uuid_column = "uuid", columns_to_check = NULL)
```

Arguments

`dataset`	dataset to be check as a dataframe or a list with the dataframe stored as "checked_dataset"
`uuid_column`	uuid column in the dataset. Default is "uuid".
`columns_to_check`	Columns to include in the checks

Value

a log with all the values that may require recoding

Examples

```
check_others(
  dataset = cleaningtools::cleaningtools_clean_data,
  uuid_column = "X_uuid",
  columns_to_check = names(cleaningtools::cleaningtools_clean_data |>
    dplyr::select(ends_with("_other")) |>
    dplyr::select(-contains(".")))
)
```

---

check_others_checks *Check if the input passed to the check_others function is correct*

Description

Check if the input passed to the check_others function is correct

Usage

```
check_others_checks(dataset, uuid_column, columns_to_check)
```

Arguments

dataset	dataset to be check as a dataframe or a list with the dataframe stored as "checked_dataset"
uuid_column	uuid column in the dataset. Default is uuid.
columns_to_check	Columns to include in the checks

Value

nothing

---

check_outliers *check outliers over the dataset*

Description

check outliers over the dataset

Usage

```
check_outliers(
  dataset,
  uuid_column = "uuid",
  element_name = "checked_dataset",
  kobo_survey = NULL,
  kobo_choices = NULL,
  cols_to_add_cleaning_log = NULL,
  strongness_factor = 3,
  minimum_unique_value_of_variable = NULL,
  remove_choice_multiple = TRUE,
  sm_separator = ".",
  columns_not_to_check = NULL
)
```

Arguments

dataset	dataset to be check as a dataframe or a list with the dataframe stored as "checked_dataset"
uuid_column	UUID. Default is uuid
element_name	name of the dataset in list
kobo_survey	Kobo survey sheet. Default is NULL.
kobo_choices	Kobo choices sheet. Default is NULL.
cols_to_add_cleaning_log	Variables those must be included in the output
strongness_factor	Strongness factor define how strong your outliers will be. The default is 3.
minimum_unique_value_of_variable	Default is NULL, mean this parameter won't be considered. For example 10 means for any variable where number of unique value is less than 10, then the variable won't be considered for outlier checking.
remove_choice_multiple	TRUE (default) will remove choice multiple questions from the output.
sm_separator	Separator for choice multiple questions. The default is "."
columns_not_to_check	Columns to exclude from the checks even if they are numeric values.

Value

return a list with the dataset checked stored as checked_dataset and a dataframe with the outliers log

Examples

```
dataset_outlier <- data.frame(
  uuid = paste0("uuid_", 1:100),
 one_value = c(round(runif(90, min = 45, max = 55)), round(runif(5)), round(runif(5, 99, 100))),
```

```
  expense = c(sample(200:500, replace = TRUE, size = 95), c(600, 100, 80, 1020, 1050)),
  income = c(c(60, 0, 80, 1020, 1050), sample(20000:50000, replace = TRUE, size = 95)),
  yy = c(rep(100, 99), 10)
)

check_outliers(dataset = dataset_outlier, uuid_column = "uuid")
```

`check_percentage_missing` *Check the percentages of missing value*

Description

The function will flag if a survey for its missing values. The missing values column can be created with add_percentage_missing and the values are flagged with check_outliers.

Usage

```
check_percentage_missing(
  dataset,
  uuid_column = "uuid",
  column_to_check = "percentage_missing",
  strongness_factor = 2,
  log_name = "percentage_missing_log"
)
```

Arguments

`dataset`	a dataset to be check as a dataframe or a list with the dataframe stored as "checked_dataset".
`uuid_column`	uuid column in the dataset. Default is "uuid".
`column_to_check`	string character with the name of the columns to check. Default is "percentage_missing"
`strongness_factor`	Strongness factor define how strong your outliers will be. The default is 3.
`log_name`	name of the log of flagged value, default is percentage_missing_log

Value

return a list with the dataset checked stored as checked_dataset and a dataframe with the flagged values log

See Also

`add_percentage_missing()`, `check_outliers()`

Examples

```
# Adding the percentage missing first
data_example <- data.frame(
  uuid = letters[1:3],
  col_1 = c(1:3),
  col_2 = c(NA, NA, "expenditures"),
  col_3 = c("with need", NA, "with need"),
  col_4 = c("food health school", NA, "food"),
  col_4.food = c(1, NA, 1),
  col_4.health = c(1, NA, 0),
  col_4.school = c(1, NA, 0)
)
data_example <- data_example %>%
  add_percentage_missing()
data_example %>% check_percentage_missing()

# With a dataset that already has a percentage missing
data_example2 <- data.frame(
  uuid = letters,
  any_cols = LETTERS,
  any_number = 1:26,
  percentage_missing = c(rep(.05, 25), .99)
)
data_example2 %>% check_percentage_missing()
```

---

check_pii *Checks for potential PII*

Description

Checks for potential PII

Usage

```
check_pii(
  dataset,
  element_name = "checked_dataset",
  uuid_column = "uuid",
  words_to_look = NULL
)
```

Arguments

dataset	dataset to be check as a dataframe or a list with the dataframe stored as "checked_dataset"
element_name	If the input is a list file, please specify the element name that contains the dataset
uuid_column	uuid column in the dataset. Default is uuid.
words_to_look	Specify the words that might be the PIIs

Value

dataset with potential PII

Examples

```
check_pii(cleaningtools_raw_data, uuid_column = "X_uuid")
```

---

check_soft_duplicates *Checks for survey similarities - Soft Duplicates*

Description

Checks for survey similarities - Soft Duplicates

Usage

```
check_soft_duplicates(
  dataset,
  kobo_survey,
  uuid_column = "uuid",
  idnk_value = "idnk",
  sm_separator = ".",
  log_name = "soft_duplicate_log",
  threshold = 7,
  return_all_results = FALSE
)
```

Arguments

dataset	dataset to be check as a dataframe or a list with the dataframe stored as "checked_dataset" stored as "checked_dataset"
kobo_survey	Kobo survey sheet.
uuid_column	uuid column in the dataset. Default is "uuid".
idnk_value	String character for the value of the "I don't know" value
sm_separator	Separator for choice multiple questions. The default is "."
log_name	Name of the log dataframe flagged in the end of the function
threshold	flag all entries less or equal a specified threshold. The default is 7.
return_all_results	By default, the function will return only the values that are under the threshold. Default is FALSE.

Value

return a list with the dataset checked stored as checked_dataset and a dataframe with the soft duplicate log

Examples

```
soft_duplicates <- check_soft_duplicates(
  dataset = cleaningtools_raw_data,
  kobo_survey = cleaningtools_survey,
  uuid_column = "X_uuid",
  idnk_value = "dont_know",
  sm_separator = ".",
  log_name = "soft_duplicate_log",
  threshold = 7
)

group_by_enum_raw_data <- cleaningtools_raw_data %>%
  dplyr::group_by(enumerator_num)

soft_per_enum <- group_by_enum_raw_data %>%
  dplyr::group_split() %>%
  purrr::map(~ check_soft_duplicates(
    dataset = .,
    kobo_survey = cleaningtools_survey,
    uuid_column = "X_uuid", idnk_value = "dont_know",
    sm_separator = ".",
    log_name = "soft_duplicate_log",
    threshold = 7,
    return_all_results = TRUE
  ))
soft_per_enum %>%
  purrr::map(~ .[["soft_duplicate_log"]]) %>%
  purrr::map2(
    .y = dplyr::group_keys(group_by_enum_raw_data) %>% unlist(),
    ~ dplyr::mutate(.x, enum = .y)
  ) %>%
  do.call(dplyr::bind_rows, .)
```

---

check_value *Check for value(s) in the dataset*

Description

Check for value(s) in the dataset

Usage

```
check_value(
  dataset,
  uuid_column = "uuid",
  element_name = "checked_dataset",
  values_to_look = c(99, 999, 999, 88, 888, 888)
)
```

Arguments

dataset	dataset to be check as a dataframe or a list with the dataframe stored as "checked_dataset"
uuid_column	uuid column in the dataset. Default is "uuid".
element_name	Name of the dataset in list
values_to_look	Values to look. Default are- 99,999,999,88,888,888

Value

A dataframe as cleaning log format

Examples

```
df <- data.frame(
  X_uuid = paste0("uuid_", 1:100),
  age = c(sample(18:80, replace = TRUE, size = 96), 99, 99, 98, 88),
  gender = c("99", sample(c("male", "female"),
    replace = TRUE, size = 95
  ), "98", "98", "88", "888")
)
check_value(
  dataset = df, uuid_column = "X_uuid",
  element_name = "checked_dataset",
  values_to_look = c(99, 98, 88, 888)
)
```

---

cleaningtools_analysis_by_group

Analysis by population group

Description

Survey data

Usage

```
cleaningtools_analysis_by_group
```

Details

These data sets include HH data (raw and clean data) and analysis along with cleaning log.

Examples

```
cleaningtools_analysis_by_group
cleaningtools_raw_data
cleaningtools_cleaning_log
cleaningtools_clean_data
cleaningtools_survey
cleaningtools_choices
cleaningtools_overall_analysis
cleaningtools_sample_frame
```

---

cleaningtools_choices *Choices tab of kobo tool*

Description

Choices tab of kobo tool

Usage

```
cleaningtools_choices
```

---

cleaningtools_cleaning_log *Cleaning log*

Description

Cleaning log

Usage

```
cleaningtools_cleaning_log
```

---

cleaningtools_clean_data *Clean data*

Description

Clean data

Usage

```
cleaningtools_clean_data
```

cleaningtools_food_consumption_df

Dataset with food consumption, household hunger Score component

Description

Dataset with food consumption, household hunger Score component

Usage

```
cleaningtools_food_consumption_df
```

cleaningtools_overall_analysis

Nation/all population level analysis.

Description

Nation/all population level analysis.

Usage

```
cleaningtools_overall_analysis
```

cleaningtools_raw_data

Raw data

Description

Raw data

Usage

```
cleaningtools_raw_data
```

cleaningtools_sample_frame *Sample frame*

Description

Sample frame

Usage

```
cleaningtools_sample_frame
```

cleaningtools_survey *Survey tab of kobo tool*

Description

Survey tab of kobo tool

Usage

```
cleaningtools_survey
```

coerce_to_character *Coerce numeric values to character, without scientific noting and NA are kept as NA.*

Description

Coerce numeric values to character, without scientific noting and NA are kept as NA.

Usage

```
coerce_to_character(x)
```

Arguments

x	a value to convert

Value

x as character

Examples

```
coerce_to_character(c("a", NA))
coerce_to_character(c(1, 10000000, NA))
```

create_audit_list *Read all audit files from a zip*

Description

Read all audit files from a zip

Usage

```
create_audit_list(audit_zip_path, dataset = NULL, uuid_column = "uuid")
```

Arguments

audit_zip_path	location to .zip file. It must contain .zip
dataset	dataset to (optional)
uuid_column	uuid column if a dataset is provided. It will only read the uuid present in the dataset. It will only be used if a dataset is provided.

Value

A list with all the audits file read as they are.

Examples

```
## Not run:
create_audit_list("audit_path.zip")

## End(Not run)
```

create_cleaning_log *Compares 2 dataset and logs differences*

Description

Compares 2 dataset and logs differences

Usage

```
create_cleaning_log(
  raw_dataset,
  raw_dataset_uuid_column = "uuid",
  clean_dataset = clean_dataset,
  clean_dataset_uuid_column = "uuid",
  check_for_deletion_log = T,
  columns_not_to_check = NULL,
  check_for_variable_name = T
)
```

Arguments

raw_dataset	Raw dataset
raw_dataset_uuid_column	uuid column in the raw dataset. Default is "uuid".
clean_dataset	Clean dataset
clean_dataset_uuid_column	uuid column in the raw dataset. Default is "uuid".
check_for_deletion_log	TRUE to flag the removed survey
columns_not_to_check	Columns to exclude from the checks
check_for_variable_name	TRUE to flag the removed variables

Value

Cleaning log

Examples

```
## Not run:
create_cleaning_log(
 raw_dataset = cleaningtools::cleaningtools_raw_data, raw_dataset_uuid_column = "X_uuid",
 clean_dataset = cleaningtools::cleaningtools_clean_data, clean_dataset_uuid_column = "X_uuid",
  check_for_deletion_log = TRUE, check_for_variable_name = TRUE
)

## End(Not run)
```

---

create_cleaning_template

Create a project folder with a cleaning template

Description

Create a project folder with a cleaning template

Usage

```
create_cleaning_template(folder_path, write_template_files, ...)
```

Arguments

folder_path	Path for the new folder
write_template_files	TRUE/FALSE if to write template files
...	Extra information collected from the RStudio wizard

Value

Folder with a copy of the cleaning_template_folder

Examples

```
## Not run:
create_cleaning_template("path/to/folder")

## End(Not run)
```

`create_clean_data` *implement cleaning log on raw data set.*

Description

implement cleaning log on raw data set.

Usage

```
create_clean_data(
  raw_dataset,
  raw_data_uuid_column = "uuid",
  cleaning_log,
  cleaning_log_uuid_column = "uuid",
  cleaning_log_question_column,
  cleaning_log_new_value_column,
  cleaning_log_change_type_column,
  change_response_value = "change_response",
  NA_response_value = "blank_response",
  no_change_value = "no_action",
  remove_survey_value = "remove_survey"
)
```

Arguments

`raw_dataset`	Raw dataset
`raw_data_uuid_column`	uuid column in the raw dataset. Default is uuid.
`cleaning_log`	cleaning + deletion log (data.frame).
`cleaning_log_uuid_column`	uuid column in the cleaning log. Default is "uuid".
`cleaning_log_question_column`	column in cleaning log which specifies data set column to change
`cleaning_log_new_value_column`	cleaning log column specifying the new correct value

cleaning_log_change_type_column
: column in cleaning log which specifies change type to be made

change_response_value
: values in change type column which should be changed to a new value.

NA_response_value
: values in change type column which should be blank (NA).

no_change_value
: values in change type column which should NOT be changed to a new value.

remove_survey_value
: values in change type column which should be deleted from the data.

Value

clean data set

Examples

```
cleaning_log_test <- data.frame(
  uuid = paste0("uuid", 1:4),
  question = c("age", "gender", "pop_group", "strata"),
  change_type = c("blank_response", "no_change", "Delete", "change_res"),
  new_value = c(NA_character_, NA_character_, NA_character_, "st-a")
)
test_data <- data.frame(
  uuid = paste0("uuid", 1:4),
  age = c(180, 23, 45, 67),
  gender = c("male", "female", "male", "female"),
  pop_group = c("idp", "refugee", "host", "idp"),
  strata = c("a", "b", "c", "d")
)


review_cleaning_log(
  raw_dataset = test_data,
  raw_data_uuid_column = "uuid",
  cleaning_log = cleaning_log_test,
  cleaning_log_change_type_column = "change_type",
  change_response_value = "change_res",
  cleaning_log_question_column = "question",
  cleaning_log_uuid_column = "uuid",
  cleaning_log_new_value_column = "new_value"
)

create_clean_data(
 raw_dataset = test_data, raw_data_uuid_column = "uuid", cleaning_log = cleaning_log_test,
  cleaning_log_change_type_column = "change_type",
  change_response_value = "change_res",
  NA_response_value = "blank_response",
  no_change_value = "no_change",
  remove_survey_value = "Delete",
  cleaning_log_question_column = "question",
```

```
  cleaning_log_uuid_column = "uuid",
  cleaning_log_new_value_column = "new_value"
)
```

`create_col_range` *Generate excel range to be used for the data validation formula in excel*

Description

This function returns an Excel cell range for a given column in a dataframe. The range is formatted for use in Excel functions or formulas and is prefixed with the sheet name 'validation_rules'.

Usage

```
create_col_range(variable, data.val)
```

Arguments

`variable`	The name of the column/question for which the Excel range is to be generated.
`data.val`	A dataframe containing the column specified by `variable`.

Value

A string representing the Excel cell range containing all choices for the column `variable` in the sheet 'validation_rules'.

Examples

```
# Assume df is a sample dataframe with a column named 'consent'.
# range_string <- create_col_range('consent', df)
```

`create_combined_log` *Merging the cleaning logs*

Description

Merging the cleaning logs

Usage

```
create_combined_log(list_of_log, dataset_name = "checked_dataset")
```

Arguments

list_of_log	A list file containig all the logs
dataset_name	The dataset's name, if it exists in the list_of_log

Value

Combined cleaning log

Examples

```
list <- cleaningtools::cleaningtools_raw_data |>
  check_pii(uuid_column = "X_uuid") |>
  check_duplicate(uuid_column = "X_uuid") |>
  check_value(uuid_column = "X_uuid")

create_combined_log(list_of_log = list)
```

---

create_duration_from_audit_sum_all

Calculate duration from audit summing all time

Description

Duration is the difference between the sum of all start and the sum of all end. It ignores all node that are empty such as form_start, end screen, form save, etc.

Usage

```
create_duration_from_audit_sum_all(audit_file)
```

Arguments

audit_file	a dataframe with a single audit file, it needs start, end, node column

Value

A dataframe with the duration in ms and duration in minutes.

Examples

```
some_audit <- data.frame(
  event = c("form start", rep("question", 5)),
  node = c("", paste0("/xx/question", 1:5)),
  start = c(
    1661415887295, 1661415887301, 1661415890819,
    1661415892297, 1661415893529, 1661415894720
  ),
```

```
  end = c(
    NA, 1661415890790, 1661415892273,
    1661415893506, 1661415894703, 1661415896452
  )
)
create_duration_from_audit_sum_all(some_audit)
```

`create_duration_from_audit_with_start_end` *Calculate duration from audit between 2 questions*

Description

The function will calculate time difference between the start of the start_question and the end of the end_question.

Usage

```
create_duration_from_audit_with_start_end(
  audit_file,
  start_question,
  end_question
)
```

Arguments

`audit_file`	a dataframe with a single audit file, it needs start, end, node column
`start_question`	character vector use for the starting question
`end_question`	character vector use for the ending question

Details

In case the node appear several time (if the value was changed or with a select multiple) it will take the minimum for the start and the maximum for the end.

If a value is missing (skip logic or question not found), it will return -Inf

Value

A dataframe with the duration in ms and duration in minutes.

Examples

```
some_audit <- data.frame(
  event = c("form start", rep("question", 5)),
  node = c("", paste0("/xx/question", 1:5)),
  start = c(
    1661415887295, 1661415887301, 1661415890819,
    1661415892297, 1661415893529, 1661415894720
```

```
  ),
  end = c(
    NA, 1661415890790, 1661415892273,
    1661415893506, 1661415894703, 1661415896452
  )
)
create_duration_from_audit_with_start_end(some_audit, "question1", "question3")
```

`create_formated_wb` *Creates formatted workbook with openxlsx*

Description

Creates formatted workbook with openxlsx

Usage

```
create_formated_wb(
  write_list,
  column_for_color = NULL,
  header_front_size = 12,
  header_front_color = "#FFFFFF",
  header_fill_color = "#ee5859",
  header_front = "Arial Narrow",
  body_front = "Arial Narrow",
  body_front_size = 11
)
```

Arguments

`write_list`	List of dataframe
`column_for_color`	Column name in the dataframe which should be used for colorizing the cell. The default is null.
`header_front_size`	Header front size (default is 12)
`header_front_color`	Hexcode for header front color (default is white)
`header_fill_color`	Hexcode for header fill color (default is red)
`header_front`	Front name for header (default is Arial Narrow)
`body_front`	Front name for body (default is Arial Narrow)
`body_front_size`	Front size for body (default is 11)

Value

A workbook

Examples

```
checks_list <- cleaningtools::cleaningtools_raw_data |>
  check_pii(uuid_column = "X_uuid") |>
  check_duplicate(uuid_column = "X_uuid") |>
  check_value(uuid_column = "X_uuid")

create_combined_log(list_of_log = checks_list) |>
  create_formated_wb()
```

`create_formatted_choices` *Format and filter Choices for 'select_one' Questions*

Description

This function returns a dataframe containing 'select_one' questions from the `tool` dataframe with their corresponding choices in the `choices` dataframe.

Usage

```
create_formatted_choices(choices, tool)
```

Arguments

`choices`	A dataframe containing the choices sheet from the Kobo tool Expected to have at least the columns `list_name` and `name`.
`tool`	A dataframe containing the survey sheet from the Kobo tool Expected to have at least the columns `type` and `name`.

Value

A dataframe containing 'select_one' questions with their corresponding choices. The dataframe has columns from both the `tool` and `choices` dataframes.

Examples

```
# formatted_choices <- create_formatted_choices(choices, tool)
```

create_logic_for_other *Create logical checks for "other" values.*

Description

Create logical checks for "other" values.

Usage

```
create_logic_for_other(
  kobo_survey,
  sm_separator = ".",
  compare_with_dataset = FALSE,
  dataset = NULL
)
```

Arguments

kobo_survey	Kobo survey sheet.
sm_separator	Separator for choice multiple questions. The default is "."
compare_with_dataset	Default FALSE will not compare the applicability of logics with the dataset whereas TRUE will check the applicability of logics.
dataset	dataset to be checked.

Value

A dataframe will all logical tests to check "others" value. It should be used in combination of review_others

Examples

```
create_logic_for_other(
  kobo_survey = cleaningtools::cleaningtools_survey,
  sm_separator = ".",
  dataset = cleaningtools::cleaningtools_clean_data,
  compare_with_dataset = TRUE
)
```

create_validation_list

Create a Validation List for Data Entry

Description

This function generates a validation list to be used for data entry validation. It combines predefined validation lists with dynamically formatted choices derived from the `choices` and `tool` parameters. The resulting dataframe is intended to provide structured and valid choices for various question types.

Usage

```
create_validation_list(choices, tool)
```

Arguments

`choices`	A dataframe representing the Kobo choices sheet. Expected to have at least the columns `list_name` and `name`.
`tool`	A dataframe representing the Kobo survey sheet. Expected to have at least the columns `type` and `name`.

Value

A dataframe where each column corresponds to a choice list for a specific question. Each row contains a valid choice for the question.

Examples

```
# Assume choices_df and tool_df are sample dataframes that fit the expected structure.
# validation_list <- create_validation_list(choices, tool)
```

create_xlsx_cleaning_log

Creates formatted excel for cleaning log

Description

Creates formatted excel for cleaning log

Usage

```
create_xlsx_cleaning_log(
  write_list,
  cleaning_log_name = "cleaning_log",
  change_type_col = "change_type",
  column_for_color = "check_binding",
  header_front_size = 12,
  header_front_color = "#FFFFFF",
  header_fill_color = "#ee5859",
  header_front = "Arial Narrow",
  body_front = "Arial Narrow",
  body_front_size = 11,
  use_dropdown = F,
  sm_dropdown_type = NULL,
  kobo_survey = NULL,
  kobo_choices = NULL,
  output_path = NULL
)
```

Arguments

write_list	List of dataframe
cleaning_log_name	Name for cleaning log from write_list
change_type_col	Change type column name in the cleaning log
column_for_color	Column name in the dataframe which should be used for colorizing the cell. The default is null.
header_front_size	Header front size (default is 12)
header_front_color	Hexcode for header front color (default is white)
header_fill_color	Hexcode for header fill color (default is red)
header_front	Front name for header (default is Arial Narrow)
body_front	Front name for body (default is Arial Narrow)
body_front_size	Front size for body (default is 11)
use_dropdown	Use drop down lists for data validation in the cleaning log output (default is FALSE)
sm_dropdown_type	Dropdown list options for select multiple questions: numerical (1/0) or logical (TRUE/FALSE) - default is logical
kobo_survey	Kobo survey dataframe

kobo_choices	Kobo choices dataframe
output_path	Output path. Default is NULL which will return a workbook instead of an excel file.

Value

save a .xlsx file or return a workbook object

Examples

```
## Not run:

checks_list <- cleaningtools::cleaningtools_raw_data |>
  check_pii(uuid_column = "X_uuid") |>
  check_duplicate(uuid_column = "X_uuid") |>
  check_value(uuid_column = "X_uuid")
create_combined_log(list_of_log = checks_list) |>
  create_xlsx_cleaning_log()

logical_check_example <- cleaningtools::cleaningtools_raw_data |>
  check_logical(
    check_to_perform = 'treat_cook_water == "always_treat"',
    uuid_column = "X_uuid",
    description = "description",
    check_id = "check_4",
    columns_to_clean = "treat_cook_water"
  )
create_combined_log(logical_check_example) |>
  create_xlsx_cleaning_log(
    output_path = paste0(tempdir(check = TRUE), "/cleaning_log.xlsx"),
    cleaning_log_name = "cleaning_log",
    change_type_col = "change_type",
    kobo_survey = cleaningtools::cleaningtools_survey,
    kobo_choices = cleaningtools::cleaningtools_choices,
    use_dropdown = TRUE,
    sm_dropdown_type = "logical"
  )

## End(Not run)
```

---

detect_variable *detects variables names in code*

Description

detects variables names in code

Usage

```
detect_variable(codeintext)
```

Arguments

codeintext	a string vector to look for variables

Value

a vector with strings

Examples

```
string_to_check <- c(
  "ki_age> 1",
  "noo==\"mar\"",
  "sum(var1, var2, na.rm = T)",
  "oks != \"mar2\"",
  "oknospace!=\"mar2\""
)
detect_variable(string_to_check)
```

---

recreate_parent_column

This function recreates the columns for select multiple questions

---

Description

This function recreates the columns for select multiple questions

Usage

```
recreate_parent_column(
  dataset,
  uuid_column = "uuid",
  kobo_survey = NULL,
  kobo_choices = NULL,
  sm_separator = ".",
  cleaning_log_to_append = NULL
)
```

Arguments

dataset	data frame
uuid_column	uuid column in the dataset. Default is "uuid".
kobo_survey	Kobo survey sheet.
kobo_choices	Kobo choices sheet.
sm_separator	Separator for choice multiple questions. The default is "."

cleaning_log_to_append
: A cleaning log where to add the changes from this functions. Names of the log from this function are "uuid", "question", "change_type", "new_value", "old_value", "comment". If the cleaning_log_to_append names are not matching, the only way is to create without a cleaning_log_to_append, and rename the columns and then bind.

Examples

```
test_data <- dplyr::tibble(
  uuid = paste0("uuid_", 1:6),
  gender = rep(c("male", "female"), 3),
  reason = c(
    "xx,yy", "xx,zy",
    "zy", "xx,xz,zy",
    NA_character_, "xz"
  ),
  reason.xx = c(0, 1, 0, 1, 0, 0),
  reason.yy = c(1, 0, 0, 0, 1, 0),
  reason.xz = c(0, 0, 0, 1, 0, 1),
  reason.zy = c(0, 1, 1, 1, 0, 0),
  reason_zy = c(NA_character_, "A", "B", "C", NA_character_, NA_character_)
)
recreate_parent_column(dataset = test_data, uuid_column = "uuid", sm_separator = ".")
```

---

review_cleaning *Review cleaning*

Description

It will compare the raw dataset, clean dataset, cleaning log and deletion log. Possible flags are :

- UUID found in deletion log.
- No action with different value in new value column.
- Changes were not applied
- This survey should be deleted from the clean dataset but it was not deleted
- This survey should be removed from deletion log as it doesn't exist in the raw data.
- Duplicated entry with different value, please recheck and keep one
- Entry missing in cleaning log
- New value in cleaning log and value in clean dataset not matching
- Survey missing in the raw data

Usage

```
review_cleaning(
  raw_dataset,
  raw_dataset_uuid_column = "uuid",
  clean_dataset,
  clean_dataset_uuid_column = "uuid",
  cleaning_log = cleaning_log_only,
  cleaning_log_uuid_column = "uuid",
  cleaning_log_change_type_column = "change_type",
  cleaning_log_question_column = "question",
  cleaning_log_new_value_column = "new_value",
  cleaning_log_old_value_column = "old_value",
  cleaning_log_added_survey_value = "added_survey",
  cleaning_log_no_change_value = c("no_action", "no_change"),
  deletion_log = NULL,
  deletion_log_uuid_column = NULL,
  check_for_deletion_log = T
)
```

Arguments

raw_dataset	Raw dataset
raw_dataset_uuid_column	uuid column in the raw dataset. Default is "uuid".
clean_dataset	Clean dataset
clean_dataset_uuid_column	uuid column in the raw dataset. Default is "uuid".
cleaning_log	Cleaning log
cleaning_log_uuid_column	uuid column in the raw dataset. Default is "uuid".
cleaning_log_change_type_column	column in cleaning log which specifies which change to be made
cleaning_log_question_column	column in cleaning log which specifies which column to change
cleaning_log_new_value_column	cleaning log column specifying the new correct value
cleaning_log_old_value_column	cleaning log column specifying the old value
cleaning_log_added_survey_value	Value for change type column which defines for new surveys
cleaning_log_no_change_value	Value for change type column which defines for no action needed
deletion_log	deletion log
deletion_log_uuid_column	Unique ID column name of deletion log
check_for_deletion_log	TRUE to flag the removed survey

Value

Discrepancy in cleaning log

Examples

```
## Not run:
deletion_log <- cleaningtools::cleaningtools_cleaning_log |>
  dplyr::filter(change_type == "remove_survey")
cleaning_log <- cleaningtools::cleaningtools_cleaning_log |>
  dplyr::filter(change_type != "remove_survey")

review_cleaning(
  raw_dataset = cleaningtools::raw_dataset, raw_dataset_uuid_column = "X_uuid",
  clean_dataset = cleaningtools::clean_dataset, clean_dataset_uuid_column = "X_uuid",
  cleaning_log = cleaning_log2, cleaning_log_uuid_column = "X_uuid",
  cleaning_log_question_column = "questions",
  cleaning_log_new_value_column = "new_value",
  cleaning_log_old_value_column = "old_value",
  deletion_log = deletaion_log,
  deletion_log_uuid_column = "X_uuid",
  check_for_deletion_log = T
)

## End(Not run)
```

---

review_cleaning_log *Review cleaning log*

Description

Review the cleaning log. It can be used before create_clean_data to check cleaning log. Possible flags are:

- Value in change_response_value not found
- question_does_not_exist: question does not exist
- uuid_does_not_exist: cannot find a UUID
- na_in_change_type: change_type is NA

Usage

```
review_cleaning_log(
  raw_dataset,
  raw_data_uuid_column = "uuid",
  cleaning_log,
  cleaning_log_uuid_column = "uuid",
  cleaning_log_question_column,
  cleaning_log_new_value_column,
  cleaning_log_change_type_column,
```

```
  change_response_value = "change_response"
)
```

Arguments

raw_dataset	Raw dataset
raw_data_uuid_column	uuid column in the raw dataset. Default is "uuid".
cleaning_log	cleaning log (data.frame)
cleaning_log_uuid_column	uuid column in the cleaning log. Default is "uuid".
cleaning_log_question_column	column in cleaning log which specifies data set column to change
cleaning_log_new_value_column	cleaning log column specifying the new correct value
cleaning_log_change_type_column	column in cleaning log which specifies change type to be made
change_response_value	values in change type column which should be changed to a new value.

Value

review of the cleaning log with flagged entries and note explaining the flags.

---

review_others *Review discrepancy between kobo relevancies and the dataset.*

Description

Review discrepancy between kobo relevancies and the dataset.

Usage

```
review_others(
  dataset,
  uuid_column = "uuid",
  kobo_survey,
  sm_separator = ".",
  information_to_add = NULL,
  columns_not_to_check = NULL
)
```

Arguments

dataset	dataset to be check
uuid_column	uuid column in the dataset. Default is "uuid".
kobo_survey	Kobo survey sheet.
sm_separator	Separator for choice multiple questions. The default is "."
information_to_add	string vector optional, if to add some information to the log (today,village etc.)
columns_not_to_check	Columns to exclude from the checks

Value

Any discrepancy between kobo relevancies and the dataset.

Examples

```
review_others(
  dataset = cleaningtools::cleaningtools_clean_data,
  uuid_column = "X_uuid", kobo_survey = cleaningtools_survey
)
```

---

review_sample_frame_with_dataset

Compares the sample frame with the clean data

Description

Compares the sample frame with the clean data

Usage

```
review_sample_frame_with_dataset(
  sample_frame,
  sampling_frame_strata_column,
  sampling_frame_target_survey_column = "Total.no.of.HH",
  clean_dataset,
  clean_dataset_strata_column,
  consent_column = "consent",
  consent_yes_value = "yes"
)
```

Arguments

sample_frame	A sample frame. It mush have a column for strata and another column defining the number of needed survey per strata.
sampling_frame_strata_column	Column name for strata in the sample frame
sampling_frame_target_survey_column	Column name defining number of survey per strata in sample frame.
clean_dataset	Clean data
clean_dataset_strata_column	Column name for strata in clean data
consent_column	Dataset consent column
consent_yes_value	Choice for consent yes

Value

a dataset with the two additional columns with the number of completed surveys and number of remaining survey.

Examples

```
review_sample_frame_with_dataset(
  sample_frame = cleaningtools::cleaningtools_sample_frame,
  sampling_frame_strata_column = "Neighbourhood",
  sampling_frame_target_survey_column = "Total.no.of.HH",
  clean_dataset = cleaningtools::cleaningtools_clean_data,
  clean_dataset_strata_column = "neighbourhood",
  consent_column = "consent_remote",
  consent_yes_value = "yes"
)
```

---

verify_valid_choices *Verify if the Kobo choices dataframe is valid*

Description

This function checks whether the provided Kobo choices dataframe meets certain criteria:

1. It is a dataframe.
2. It is not empty.
3. It contains columns named "list_name" and "name".

Usage

```
verify_valid_choices(kobo_choices)
```

Arguments

kobo_choices A dataframe representing the Kobo choices sheet.

Value

Logical. Returns TRUE if the dataframe meets all the criteria. Otherwise, it returns FALSE.

Examples

```
# Assume df_valid_choices is a valid Kobo choices dataframe
df_valid_choices <- data.frame(list_name = c("ChoiceA", "ChoiceB"), name = c("Option1", "Option2"))
verify_valid_choices(df_valid_choices) # should return TRUE

# Assume df_invalid_choices lacks the required columns
df_invalid_choices <- data.frame(column1 = c("ChoiceA", "ChoiceB"),
                                 column2 = c("Option1", "Option2"))
verify_valid_choices(df_invalid_choices) # should return FALSE
```

---

verify_valid_survey *Verify if the Kobo survey dataframe is valid*

Description

This function checks whether the provided Kobo survey dataframe meets certain criteria:

1. It is a dataframe.
2. It is not empty.
3. It contains columns named "type" and "name".

Usage

```
verify_valid_survey(kobo_survey)
```

Arguments

kobo_survey A dataframe representing the Kobo survey sheet.

Value

Logical. Returns TRUE if the dataframe meets all the criteria. Otherwise, it returns FALSE.

Examples

```
# Assume df_valid is a valid Kobo survey dataframe
df_valid <- data.frame(type = c("integer", "selecte_one yesno"), name = c("age", "consent"))
verify_valid_survey(df_valid) # should return TRUE

# Assume df_invalid lacks the required columns
df_invalid <- data.frame(column1 = c("integer", "selecte_one yesno"), column2 = c("age", "consent"))
verify_valid_survey(df_invalid) # should return FALSE
```

Index