Package: gophr (via r-universe)

November 14, 2024

Title Utility functions related to working with the MER Structured Dataset

Version 4.1.6

Description This packages contains a number of functions for working with the PEPFAR MSD.

Depends R (>= 3.4.2)

License MIT + file LICENSE

Imports crayon, curl, getPass, dplyr (>= 1.0.0), glue, googlesheets4, httr, lifecycle, magrittr, purrr, readr, stringr, tibble, tidyr, tools, usethis, vroom, jsonlite

Suggests arrow, aws.s3, knitr, rmarkdown, testthat (>= 3.0.0)

Remotes USAID-OHA-SI/glamr, USAID-OHA-SI/gagglr, USAID-OHA-SI/grabr

Encoding UTF-8

LazyData true

RoxygenNote 7.3.2

URL `https://usaid-oha-si.github.io/gophr/`

Config/testthat/edition 3

Config/testthat/parallel true

VignetteBuilder knitr

Config/pak/sysreqs git make libgit2-dev libicu-dev libssl-dev libx11-dev

Repository https://usaid-oha-si.r-universe.dev

RemoteUrl https://github.com/USAID-OHA-SI/gophr

RemoteRef HEAD

RemoteSha 5bdb0ca3e9ae44d4923f0bbbf19705e4bc134e4d

Contents

`achv_color_map` *OHA Achievement Colors*

Description

A tibble of the OHA Colors associated with different target achievement thresholds - 'At Risk', 'Concerned', 'On Target', and 'Above Target'. This table includes the current mapping as well as the original/classic one. These colors are used in association with ‘adorn_achievement‘.

Usage

```
data(achv_color_map)
```

Format

A data frame with three variables and 4 rows

achv_desc description of each achievement level

achv_color hex code of the associated color to the achievement level

achv_color_classic classic/original hex code of the associated color to the achievement level

---

`adorn_achievement` *Adorn Achievement - Percent and Color*

Description

‘adorn_achievement‘ calculate target achievement (cumulative and/or quarterly) for a standard MSD or one reshaped using ‘reshape_msd‘()‘ as well as to apply achievement group labels and colors. It will run ‘calc_achievement‘ if an achievement column does not exist in the dataset.

Usage

```
adorn_achievement(df, qtr = NULL, classic = FALSE)
```

Arguments

`df`	data frame as standrd MSD or one from reshape_msd()
`qtr`	if using standard MSD, need to provide the most recent quarter, ideally using identifypd(df_msd, pd_type = "quarter")
`classic`	use the original OHA achievement color palette (pre July 2024), default = FALSE

Value

data frame with achievement values, labels, and colors

See Also

Other achievement: `calc_achievement()`

Examples

```
## Not run:
df_msd <- read_msd(path)
df_msd_agg <- df_msd %>%
 filter(operatingunit == "Jupiter"
        indicator %in% c("TX_NEW", "TX_CURR"),
        funding_agency != "Dedup",
        standardizeddisaggregate == "Total Numerator") %>%
 group_by(operatingunit, funding_agency, fiscal_year, indicator) %>%
 summarise(across(where(is.double), sum, na.rm = TRUE)) %>%
 ungroup()
```

```
adorn_achievement(df_msd_agg)

df_msd_agg %>%
 reshape_msd("quarters") %>%
 adorn_achievement()

df_msd_agg %>%
 reshape_msd("quarters", qtrs_keep_cumulative = TRUE) %>%
 adorn_achievement()
## End(Not run)
```

`apply_funding_type` *Apply Funding Type*

Description

When working with financial or HRH data, its often useful to determine whether funding or staffing are counted as "service delivery" or "non-service delivery"

Usage

```
apply_funding_type(df)
```

Arguments

`df`	Financial Structured Dataset data frame

Value

a new column, funding_type

`apply_partner_type` *Apply partner type (local/international)*

Description

‘apply_partner_type‘ pulls from USAID managed Google Sheet maintained by the Local Partner team that designates all USAID mechanisms by their type, local or international. The designations include the updates for regional partners, which in FY21 are classified as local. These adjustments can be found on the adjusted column.

Usage

```
apply_partner_type(df)
```

Arguments

`df`	data frame, MER or financial dataset

Value

a df with the partner types provided by USAID.

`browse_knownissues` *Browse Known Issues*

Description

Launches the USAID managed, Known Data Issues Tracker.

The Known Data Issues Tracker is a table that summarizes different issues that are known in DATIM but cannot be resolved. For example, when a treatment partner start/ends mid-year, there will be duplicated targets that cannot be adjusted in DATIM.

The list of known issues is maintained by USAID/SIEI division. To access the table requires having a USAID email account.

Usage

```
browse_knownissues()
```

Value

Launches The Known Data Issues Tracker Google Sheet.

See Also

[resolve_knownissues()] to exclude Known Issues from data set; [set_email()] to store USAID email; [load_secrets()] to load credentials into session.

Examples

```
 ## Not run:
 load_secrets()
 browse_knownissues()
## End(Not run)
```

calc_achievement *Calculate Achievement*

Description

'calc_achievement' creates a target achievement column standard to many PEPFAR analyses. It can calculate achievement from a normal MSD or one reshaped using 'reshape_msd'.

Usage

```
calc_achievement(df)
```

Arguments

df	MSD based dataframe

Value

one or two additional columns that calculate achievement and/or quarterly achievement

See Also

[reshape_msd()]

Other achievement: adorn_achievement()

Examples

```
## Not run:
df_msd <- read_msd(path)
df_msd_agg <- df_msd %>%
 filter(operatingunit == "Jupiter"
        indicator %in% c("TX_NEW", "TX_CURR"),
        funding_agency != "Dedup",
        standardizeddisaggregate == "Total Numerator") %>%
 group_by(operatingunit, funding_agency, fiscal_year, indicator) %>%
 summarise(across(where(is.double), sum, na.rm = TRUE)) %>%
 ungroup()

calc_achievement(df_msd_agg)

df_msd_agg %>%
 reshape_msd("quarters") %>%
 calc_achievement()
## End(Not run)
```

`cascade_ind` *MER Clinical Cascade indicators*

Description

List of indicators in the MER that comprise the clinical cascade. These indicators are needed for calculating the 90s/95s plus linkage and viral load coverage.

Usage

```
data(cascade_ind)
```

Format

A list of clinical cascade indicators

cascade_ind indicator names

`clean_agency` *Clean data from funding agency column*

Description

This function converts all funding agency names to upper case removes the HHS prefix for those agencies and moves State and USAID subsidiaries under their parent agencies

Usage

```
clean_agency(.data)
```

Arguments

`.data`	MSD Datasets

Value

Cleaned MSD DataFrame

See Also

Other column munging: `clean_column()`, `clean_indicator()`, `clean_psnu()`

Examples

```
## Not run:
 df_msd %>% clean_agency()
## End(Not run)
```

`clean_column` *Clean column data*

Description

Clean column data

Usage

```
clean_column(.data, colname = "psnu")
```

Arguments

`.data`	MSD Datasets
`colname`	Name of the column(s)

Value

Cleaned MSD DataFrame

See Also

Other column munging: `clean_agency()`, `clean_indicator()`, `clean_psnu()`

Examples

```
## Not run:
 df_msd %>% clean_column(colname = "psnu")
## End(Not run)
```

`clean_indicator` *Clean indicators (apply _D suffix)*

Description

‘clean_indicator‘ applies a ’_D’ suffix to any indicators that are a denominator. This is particularly useful when aggregating data or reshaping.

Usage

```
clean_indicator(df)
```

Arguments

`df`	MSD data frame

Value

indicators with denominator have _D suffix

See Also

Other column munging: clean_agency(), clean_column(), clean_psnu()

Examples

```
## Not run:
df <- df %>%
filter(indicator == "TX_PVLS",
       standardizeddisaggregate %in% c("Total Numerator", "Total Denominator")) %>%
clean_indicator()
## End(Not run)
```

`clean_psnu` *Clean PSNU column data*

Description

Clean PSNU column data

Usage

```
clean_psnu(.data)
```

Arguments

.data	MSD Datasets

Value

Cleaned MSD DataFrame

See Also

Other column munging: clean_agency(), clean_column(), clean_indicator()

Examples

```
## Not run:
 df_msd %>% clean_psnu()
## End(Not run)
```

get_metadata *Extract MSD Meta Data*

Description

This function is used to extract meta data as a list from the source file of a MER Structured Dataset, MER NAT_SUBNAT Structured Dataset, Financial Structure Dataset, HRH Structured Dataset, or DATIM Genie export. It creates a list object, metadata, in the global environment containing the source, current fiscal year, current period, current quarter, as well as a caption.

Usage

```
get_metadata(path, type, caption_note)
```

Arguments

`path`	path to the folder containing MSDs or specific MSD file (default relies on glamr::si_path() if available)
`type`	PSD type: "OU_IM", PSNU_IM", "NAT_SUBNAT", "Financial"; default = "OU_IM_FY2*"
`caption_note`	additional information to include in a viz caption footer

Value

list of meta data information about the source dataset

See Also

Other metadata: `extract_metadata()`, `source_info()`

Examples

```
## Not run:
 meta <- get_metadata() #works if you have stored path to the MSD folder via glamr::set_paths()
  meta$curr_fy
## End(Not run)

## Not run:
library(tidyverse)
library(glamr)
library(gophr)
library(glue)

ref_id <- "1bdf4c4e"

meta <- get_metadata(caption_note = "Created by: The Dream Team")

cntry <- "Saturn"
```

```
df <- si_path() %>%
  return_latest("OU_IM") %>%
  read_msd()

df_viz <- df %>%
  filter(operatingunit == cntry,
         fiscal_year == meta$curr_fy,
         indicator == "TX_NEW",
         standardizeddisaggregate == "Total Numerator")

df_viz <- df_viz %>%
  group_by(fiscal_year, indicator, mech_code) %>%
  summarise(across(c(targets, starts_with("qtr")), sum, na.rm = TRUE),
            .groups = "drop")

df_viz <- reshape_msd(df_viz, "quarters")

df_viz %>%
  ggplot(aes(period, results_cumulative)) +
  geom_col() +
  geom_text(data = . %>% filter(., period == meta$curr_pd),
            aes(label = results_cumulative),
            vjust = -.5) +
  facet_wrap(~fct_reorder2(mech_code, period, targets)) +
 labs(title = glue("Upward trend in TX_NEW results thru {meta$curr_qtr} quarters") %>% toupper,
      subtitle = glue("{cntry} | {meta$curr_fy_lab} cumulative mechanism results"),
      x = NULL, y = NULL,
      caption = glue("{meta$caption}"))
## End(Not run)
```

`identifypd` *Extract Current Reporting Period*

Description

'identifypd' uses the information from the raw MSD structure to identify the current reporting period (fiscal year and/or quarter). This function can be used to make other inputs in your code more dynamic. Originally, 'identifypd' was developed for use in achafetz/PartnerProgress

Usage

```
identifypd(df, pd_type = "full", pd_prior = FALSE)
```

Arguments

`df`	dataset to use to find latest period
`pd_type`	what is returned? (a) full, eg fy2018q1; (b)year, eg 2018; (c) quarter, eg 1
`pd_prior`	do you want the last period returned (instead of the current); default = FALSE

Examples

```
## Not run:
#return full, current period, eg "fy2018q3"
identifypd(df_mer)
#return the current quarter, eg 3
identifypd(df_mer, "quarter")
#return the prior quarter, eg "fy2018q2"
identifypd(df_mer, pd_prior = TRUE)

## End(Not run)
```

---

identify_psd *Identify PSD Data Stream and Level*

Description

This function is useful as a utility function in this and other packages to determine the type of file that is read in and determine certain types of munging handling.

Usage

```
identify_psd(df)
```

Arguments

df	PSD dataframe

Value

character, PSD data stream and the level

Examples

```
## Not run:
#read in file for use
 path <- "~/Data/ICPI_MER_Structured_Dataset_PSNU_20180323_v2_1.txt"
 df <- read_psd(path)
#identify data stream
 identify_psd(df_psnu)
## End(Not run)
```

`pluck_totals` *Filter for Only Totals in MSD*

Description

Often times we want just want to work with totals when performing an analysis or visualization. This function is a simple filter for Total Numerator and Denominator in a MSD dataframe

Usage

```
pluck_totals(df)
```

Arguments

`df`	MER Structured Dataset (MSD) dataframe

Value

MSD with only total numerator and denominator

Examples

```
df_msd <- read_msd(path)
df_totals <- df_msd %>%
  pluck_totals() %>%
  clean_indicator() %>%
  filter(indicator %in% cascade_ind)
```

`read_msd` *Import PEPFAR Structured Datasets .txt into R and covert to .rds*

Description

Deprecated. Use ‘read_psd‘ instead.

Usage

```
read_msd(
  file,
  save_rds = FALSE,
  remove_txt = FALSE,
  convert_to_old_names = FALSE
)
```

Arguments

`file`	enter the full path to the PEPFAR structured dataset file
`save_rds`	save the Structured Dataset as an rds file, default = FALSE
`remove_txt`	should the txt file be removed, default = FALSE
`convert_to_old_names`	replace FY22Q2 naming convention with old? default = FALSE

`read_psd` *Import PEPFAR Structured Datasets to R*

Description

'read_psd' imports a stored PEPFAR Structured Datasets (.zip, .txt, or .parquet). The function will read in a MSD, Genie, Financial, HRH, or DHI PEPFAR dataset, ensuring the column types are correct. The user has the ability to store the txt file as a rds or parquet file, significantly saving storage space on the computer (and can then remove the txt file after importing). Most of USAID/OHA processes and analyses rely on the use of the MSD file being read in via 'read_psd'. This function can be used in the PDAP space in addition to working locally.

Usage

```
read_psd(
  file,
  export_format = "none",
  remove_base_file = FALSE,
  retain_genie_cols = FALSE
)
```

Arguments

`file`	enter the full path to the PEPFAR structured dataset file
`export_format`	if desired, save the PSD in another compressed format, either "rds" or "parquet", default = "none"
`remove_base_file`	should original base file be removed if exporting in another compressed format? default = FALSE
`retain_genie_cols`	should Genie specific columns ('dataelementuid', 'categoryoptioncombouid', 'approvallevel', 'approvalleveldescription') be retained in the output dataset? default = FALSE

Examples

```
## Not run:
#convert Q1 clean PSNU file from txt to Rds
#read in file for use
 path <- "~/Data/MER_Structured_Datasets_OU_IM_FY22-24_20240315_v2_1.zip"
 df_psnu <- read_psd(path)
## End(Not run)
```

`remove_centralsupport` *Remove Central Support Reporting*

Description

Central Support is often desired to be removed from MER analysis. As of FY21Q4, "CS" is a field under ‘indicatortype‘, which can be used to identify and exclude this sort of reporting. This function is run by default in ‘resolve_knownissues‘, but can be run separately if desired.

Usage

```
remove_centralsupport(df)
```

Arguments

`df`	MSD dataframe (must include indicatortype)

Value

df with central support reporting removed

`remove_mo` *Remove M&O funding*

Description

When working with financial data, its often useful to remove M&O funding from the data.

Usage

```
remove_mo(df)
```

Arguments

`df`	financial or comprehensive budget dataset

Value

a dataset excluding M&O

remove_sch *Remove Supply Chain Funding*

Description

When working with financial data, its often useful to remove Supply Chain mechanism and funding from the data.

Usage

```
remove_sch(df, poc = c("SCH", "SGAC"), flag_only = FALSE)
```

Arguments

`df`	this can be either a financial structured dataset or an MSD
`poc`	you can choose to filter either the SCH list or the SGAC list; default is both
`flag_only`	allows you keep full dataset, but identify the mechanisms that are supply chain, mech_sch as a logical; default = FALSE

Details

The list of SCH mechanisms is maintained by the EA team on a Google Sheet. A USAID email is required to access the dataset.

Value

a df without supply chain mechanisms

See Also

[set_email()] to store USAID email; [load_secrets()] to load credentials into session

Examples

```
## Not run:
#authenticate
load_secrets()
#remove SCh using SGAC list
df <- remove_shc(df, poc = "SGAC")
## End(Not run)
```

`rename_official` *Apply the latest mechanism and partner names from DATIM*

Description

Some mechanisms and partners are recorded in FACTSInfo with multiple names over different time period. This function replaces all partner and mechanism names the most recent name for each mechanism ID pulling from a DATIM SQL View. The ‘mech_code‘ variable is required in your dataset, and having ‘operatingunit‘ and ‘fiscal_year‘ or ‘period‘ are highly recommended as they will limit the size of the DATIM queries.With an DHIS2 update to DATIM in 2021, the DATIM mechanism tables requires a password to access. We would recommend using ‘glamr::set_datim()‘ to store your DATIM credentials securely on your local machine. If you don’t have them stored, you will be prompted each time to enter your password to acces DATIM.

Usage

```
rename_official(df, datim_user, datim_pwd)
```

Arguments

`df`	identify the PEPFAR Structured Data Set to clean
`datim_user`	DATIM username; if missing will look for stored credentials first and then prompt for them if not found
`datim_pwd`	DATIM password; if missing will look for stored credentials first and then prompt for them if not found

Examples

```
## Not run:
df_psnu_im <- read_psd(file) %>% filter(country == "Saturn")
df_psnu_im <- rename_official(df_psnu_im)
## End(Not run)
```

`reshape_msd` *Reshape MSD*

Description

‘reshape_msd‘ transforms the structure into a tidy format. The default reshape, makes the whole dataset long by period, but other options include wide to match the original MSD, semi-wide for a separate column for targets, cumulative, and results, and then a quarterly one which keeps the targets (and cumulative) in their own columsn but makes the quarters long. Reshaping tidy is key for much of OHA/SI processes and analysis.

Usage

```
reshape_msd(
  df,
  direction = c("long", "wide", "semi-wide", "quarters"),
  include_type = TRUE,
  qtrs_keep_cumulative = FALSE
)
```

Arguments

`df`	MSD dataset in the semi-wide format
`direction`	direction of reshape, "long" (default), "wide" (original MSD structure), "semi-wide" (one column for targets, cumulative, results) or "quarters" (quarters pivoted, but not targets - useful for quarterly achievement)).
`include_type`	whether a period_type column (targets, results, cumulative) should be included, default = TRUE
`qtrs_keep_cumulative`	whether to keep the cumulative column when using quarters for direction, default = FALSE

Examples

```
## Not run:
 #read in data
  df_genie <- match_msd("~/Downloads/PEPFAR-Data-Genie-PSNUByIMs-2018-08-15.zip")
 #reshape long
  df_genie_long <- reshape_msd(df_genie)
 #reshape wide (to look like old format)
  df_genie_long <- reshape_msd(df_genie, direction = "wide")
 #reshape semi-wide (one column for targets, cumulative, results)
  df_genie_wide <- reshape_msd(df_genie, direction = "semi-wide")
 #reshape quarters (quarters pivoted, but not targets - useful for quarterly achievement)
  df_genie_wide <- reshape_msd(df_genie, direction = "semi-wide")

## End(Not run)
```

`resolve_knownissues` *Resolve Known Issues*

Description

This function handles known issues in the MSD. For example, a mechanism starting mid-year and reporting on TX_CURR will duplicate TX_CURR targets. This function resolves that by removing those known cases and stores cases relevant to your data frame.

The best workflow is to filter your dataset down to a country and/or technical area of interest before running ‘resolve_knownissues()‘. When you run the function, it will print out any known issues to the console (and can) even store them to your Global Environment, so it makes sense to limit the data first to what you are using/care about.

The list of known issues is maintained by USAID/SIEI division. To access the table requires having a USAID email account and can be accessed via ‘browse_knownissues()‘.

Usage

```
resolve_knownissues(df, remove_cs = TRUE, store_excl = FALSE)
```

Arguments

`df`	standard MSD data frame, typically after its been filtered
`remove_cs`	remove data flagged as central support (CS), default = TRUE
`store_excl`	should the known exclusions be store in the Global Envir?

Value

df excluding known targets/results issues

See Also

[browse_knownissues()] to view table in Google Sheets; [set_email()] to store USAID email; [load_secrets()] to load credentials into session.

Examples

```
## Not run:

library(tidyverse)
library(glamr)

load_secrets() # or googlesheets4::gs4_auth()

df_msd <- si_path() %>%
  return_latest("OU_IM") %>%
  read_rds()

df_mwi <- df_msd %>%
  filter(operatingunit == "Malawi",
         indicator == "TX_CURR")

df_mwi_resolved <- df_mwi %>%
  resolve_knownissues()
## End(Not run)
```

`snapshot_ind` *MER Snapshot indicators*

Description

List of indicators in the MER that are snapshot indicators, eg TX_CURR, as opposed to a cumulative one, like HTS_TST. Snapshot indicators are handled differently for calculating cumulative values and target achievement.

Usage

```
data(snapshot_ind)
```

Format

A list of all snapshot indicators

snapshot_ind indicator names

`source_info` *Extract MSD Source Information*

Description

This function is used primarily to extract the data from the source file of a MER Structured Dataset, MER NAT_SUBNAT Structured Dataset, Financial Structure Dataset, or DATIM Genie export. It can also be used to extact information from the filename about the fiscal year, quarter, or period.

Usage

```
source_info(path, type, return = "source")
```

Arguments

`path`	path to the folder containing MSDs or specific MSD file
`type`	not required unless providing a folder in ‘path‘; default = "OU_IM_FY21"; other examples include: "PSNU_IM", "NAT_SUBNAT", "PSNU", "Financial", "HRH"
`return`	from the info, what should be returned; default = "source" other options are: "period", "fiscal_year", "fiscal_year_label","quarter"

Value

vector of information related to what is being asked in ‘return‘

See Also

Other metadata: extract_metadata(), get_metadata()

Examples

```
## Not run:
 source_info() #works if you have stored path to the MSD folder via glamr::set_paths()
 source_info("../Data", type = "PSNUxIM")
 source_info("../Data", type = "PSNUxIM", return = "period")
 source_info("../Downloads/Genie_PSNU_IM_Jupiter_Daily_c9f5889f-86c9-44e7-ab63-fa86c587d251.zip")
 source_info("../Data/MER_Structured_Datasets_NAT_SUBNAT_FY15-21_20210618_v2_1.rds")
## End(Not run)

## Not run:
library(tidyverse)
library(glamr)
library(glue)

df <- si_path() %>%
  return_latest("OU_IM") %>%
  read_msd()

df_viz <- df %>%
  filter(operatingunit == "Saturn",
         indicator == "TX_NEW",
         standardizeddisaggregate == "Total Numerator") %>%
  count(fiscal_year, wt = targets, name = "targets")

df_viz %>%
  ggplot(aes(fiscal_year, targets)) +
  geom_col() +
  labs(caption = glue("Source: {source_info()}"))
## End(Not run)
```

---

split_save *Export a single dataset into multiple files by group*

Description

'split_save' breaks a dataset by the groups provided and then exports those individual frames as separate csv files. This process can be useful when working with data across multiple partners and then creating datasets to be sent to each of them with their own data.

Usage

```
split_save(df, group_var, folderpath, filename_stub, include_date = FALSE)
```

Arguments

`df`	dataframe to split
`group_var`	grouping variable to split the dataset by, eg operatingunit, funding_agency
`folderpath`	directory where you want to store the files
`filename_stub`	generic stub for naming all the files
`include_date`	include date after filenamestub? default = FALSE, eg "20180913"

Examples

```
## Not run: 
#create country specific files for TX_NEW
 df_mer %>%
  filter(indicator == "TX_NEW",
         standardizeddisaggregate == "Total Numerator") %>%
  split_save(operatingunit, "~/CountryFiles", "FY18Q3_TX")

## End(Not run)
```

Index