Package ‘DefraUtils’

February 12, 2026

Type Package

Title Utility functions for use by Defra analysts.

Version 1.0.3

Description The DefraUtils package contains a variety of functions that
could be useful for a variety of analysts within Defra. These include
functions to standardise our ways of working (e.g. project templates) and
improve consistency in our approaches (e.g. to rounding). This is an open
project and all Defra users should feel free to contribute any functions
they think will be of wider use to defra.

License MIT

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.3

Suggests futile.logger,
mockery,
srvyr,
testthat (>= 3.0.0),
tidyxl,
withr

Config/testthat/edition 3

Depends R (>= 3.5)

LazyData true

Imports afcharts,
aftables,
arrow,
bestNormalize,
brickster,
cli,
credentials,
dplyr,
fs,
ggplot2,

ggrepel,
gitcreds,
glue,
grDevices,
grid,
gridtext,
here,
httr,
httr2,
janitor,
labeling,
lubridate,
magrittr,
modelr,
moments,
openxlsx,
pacman,
plyr,
png,
polite,
purrr,
quarto,
readODS,
readr,
readxl,
rio,
rlang,
rstudioapi,
rvest,
scales,
stats,
stringr,
survey,
tibble,
tidyr,
tools,
useful,
usethis,
xfun,
xml2,
zoo

Contents

`add_missing_columns` *Add missing columns*

Description

Given a list of columns which should occur in a dataset, adds in the missing columns and fill them with the supplied default value. Will only add columns, not remove them.

Usage

```
add_missing_columns(df, full_column_list, fill_value = NA, reorder = TRUE)
```

Arguments

`df`	A dataset
`full_column_list`	A character vector of all the columns which should appear in the dataset
`fill_value`	The value that the new columns should be populated with
`reorder`	Logical, default = `TRUE`; Should the columns of the output data be reordered using the `full_column_list`?

`calculate_mcib` *Mean Constrained Integration Over Brackets (MCIB)*

Description

Used by `estimate_band_means()`. Run to check how MCIB has been calculated.

Usage

```
calculate_mcib(band_counts_df, bands_df)
```

Arguments

`band_counts_df`	Counts of bands, created using `compute_band_counts()`
`bands_df`	A table of bands, created using `create_bands_df()`

Details

`https://journals.sagepub.com/doi/10.1177/0081175018782579#sec-4`

Author(s)

Farm Business Survey team (fbs.queries@defra.gov.uk)

See Also

Other functions for estimating means of banded data: `calculate_riob()`, `calculate_rpme()`, `compute_band_counts()`, `create_bands_df()`, `estimate_band_means()`

`calculate_riob` *Calculate Robust Integration Over Brackets*

Description

Used by `estimate_band_means()`. You can use this function to check that band labels have been assigned correctly, otherwise use `estimate_band_means()`.

Usage

```
calculate_riob(
  band_counts_df,
  bands_df,
  method = c("Geometric", "Arithmetic", "Harmonic", "Median"),
  min_alpha = 1.11
)
```

Arguments

`band_counts_df`	Counts of bands, created using `compute_band_counts()`; fed into both `calculate_mcib()` and `calculate_rpme()`
`bands_df`	A table of bands, created using `create_bands_df()`; fed into `calculate_mcib()`
`method`	String; one of "Arithmetic", "Harmonic", "Geometric" or "Median", fed into `calculate_rpme()`
`min_alpha`	Numeric; the minimum threshold for the alpha parameter of the Pareto distribution, fed into `calculate_rpme()`

Author(s)

Farm Business Survey team (fbs.queries@defra.gov.uk)

See Also

Other functions for estimating means of banded data: `calculate_mcib()`, `calculate_rpme()`, `compute_band_counts()`, `create_bands_df()`, `estimate_band_means()`

calculate_rpme *Robust Pareto Midpoint Estimator (RPME)*

Description

Used by estimate_band_means(). Run to check how RPME has been calculated.

Usage

```
calculate_rpme(
  band_counts_df,
  method = c("Geometric", "Arithmetic", "Harmonic", "Median"),
  min_alpha = 1.11
)
```

Arguments

band_counts_df	Counts of bands, created using compute_band_counts()
method	What kind of average to return: "Arithmetic", "Harmonic", "Geometric" or "Median"
min_alpha	Numeric; the minimum threshold for the alpha parameter of the Pareto distribution

Author(s)

Farm Business Survey team (fbs.queries@defra.gov.uk)

See Also

Other functions for estimating means of banded data: calculate_mcib(), calculate_riob(), compute_band_counts(), create_bands_df(), estimate_band_means()

compare_two_years *Compare two year's worth of data*

Description

Data must contain the following four columns: survey_year, grouping_factor, group, value

Usage

```
compare_two_years(
  df,
  year_1,
  year_2,
  year_col = "survey_year",
  grouping_col = "grouping_factor",
  group_col = "group",
  group_levels = NULL,
  group_labels = group_levels,
  value_col = "value",
  price_type = NULL,
  extra_vars = NULL,
  diff = "percent",
  ...
)
```

Arguments

`df`	A data frame
`year_1`	First year to include in the comparison
`year_2`	Second year to include in the comparison
`year_col`	Default = `"survey_year"`; The column with the years in
`grouping_col`	Default = `NULL`; The column with the grouping variables in - only required if you want this column returned
`group_col`	Default = `NULL`; The column with the group variables in - only required if you want this column returned
`group_levels`	Default = `NULL`; If you have a 'group' column that you are using in the commentary, you may want to edit the levels of this column so that they read better in text, e.g. "LFA grazing livestock" rather than "Grazing livestock (LFA)"; to do this, set the current levels of the group column here, and the labels you want with `group_labels`
`group_labels`	Default = `group_levels`; The new level labels for the group column
`value_col`	The column with the values in
`price_type`	Default = `NULL`; If the data contains both real and current prices, which should be included? To include both keep as `NULL`. This requires the data to contain a column called 'prices'.
`extra_vars`	String; Extra variable(s) to select, e.g. band
`diff`	String; Either `"percent"` (default; for difference in %) or `"points"` (for difference in percentage points)
`...`	arguments passed to `round_with_commas()` (i.e. `method`, `round_to`, `optimise_to` and `round_zeros`)
`prefix`	Default = `""`; If you are calculating percentage change, you can add a prefix to the returned values, e.g. `"£"`

Author(s)

Farm Business Survey team (fbs.queries@defra.gov.uk)

See Also

Other commentary functions: `create_list()`, `get_address()`, `get_diff_in_words()`, `get_pc_in_words()`

`compute_band_counts` *Count the population in each band using weights*

Description

Used by `estimate_band_means()`

Usage

```
compute_band_counts(
  df,
  bands_df,
  band_col,
  grouping_cols = "year",
  weights = "weight"
)
```

Arguments

`df`	A data frame containing banded incomes
`bands_df`	A table of bands, corresponding with the values in `band_col`, created using `create_bands_df()`
`band_col`	String; The name of the column in `df` with bands in
`grouping_cols`	String (default = "year"); Columns to group the counts by, along with band (if no groups, set to `NULL`)
`weights`	String (default = "weight"); The name of the column containing the weights

Author(s)

Farm Business Survey team (fbs.queries@defra.gov.uk)

See Also

Other functions for estimating means of banded data: `calculate_mcib()`, `calculate_riob()`, `calculate_rpme()`, `create_bands_df()`, `estimate_band_means()`

`connect_github_pat` *Connect RStudio to your GitHub account using a Personal Access Token*

Description

This is a simple function which connects RStudio to GitHub using a Personal Access Token (PAT), allowing you to work on GitHub repositories. This function is for local RStudio installs only, not for on the DASH platform. For the dash platform, use `connect_github_ssh()` instead.

Usage

```
connect_github_pat(username, email)
```

Arguments

`username`	string containing GitHub username
`email`	string containing email address used for GitHub account

Details

This function will set your GitHub credentials and Personal Access Token (PAT) and connect RStudio to GitHub. This is essential if you want to work in RStudio in projects stored as GitHub repositories.

It uses `system()` to run the code in the terminal to set your credentials, and `gitcreds::gitcreds_set()` and `credentials::set_github_pat()` packages to connect your RStudio to GitHub.

`gitcreds::gitcreds_set()` is an interactive function and will prompt users for input. To replace existing credentials and PAT choose option 2. You will then be prompted for you PAT. Your PAT should be changed every 30 days to ensure security.

Note: For this function to work you must:

- have git installed on your local machine
- have a GitHub account
- have created a PAT on GitHub.

An additional feature of this function (not mentioned in the Defra instructions) is to add the `credentials::set_github_pat` function call to an .Rprofile. This will ensure your PAT is set for every R session, meaning you won't need to provide your PAT when running functions such as `devtools::install_github()`. The function will check if an .Rprofile file already exists, if one does it will add the code to the bottom of the existing profile. If the .Rprofile file does not exist, the function will create one and add the code to it.

This function is for local RStudio installs only, not for on the DASH platform.

Value

GitHub credentials and PAT set

Note

This function is interactive, requiring user input. Therefore it is not suitable for automated scripts.

Author(s)

Josh Moatt (Joshua.Moatt@defra.gov.uk)

See Also

`gitcreds::gitcreds_set()`, `credentials::set_github_pat()`, `connect_github_ssh()`

Examples

```
## Not run:
# Set GitHub credentials
connect_github_pat(
  username = "my_github_username",
  email = "my_email"
)

## End(Not run)
```

`connect_github_ssh` *Connect RStudio to your GitHub account using SSH*

Description

Connect RStudio to your GitHub account using SSH - this is the preferred method of connecting RStudio and GitHub on the DASH platform. The user will only need to supply their GitHub username and email address, then the function will set their credentials and generate an SSH key. Once the SSH key has been added to GitHub, the function does the final bits of set-up. All steps and outputs are printed in the RStudio console rather than the terminal, which offers a more user friendly experience. This method of connection is only recommended for the DASH platform - for connecting local RStudio installs to GitHub, see `connect_github_pat()` instead.

Usage

```
connect_github_ssh(username, email)
```

Arguments

`username`	user's GitHub username
`email`	user's email address

Details

The suggested method for connecting RStudio and GitHub on the DASH platform is via SSH. The alternative is via Personal Access Token, but this method can be a bit clunky on the DASH platform. Connecting via SSH provides a much smoother user experience.

This function is designed to simplify the connection process and avoid the user having to interact with the terminal - which many beginners to Git find intimidating. Instead, all prompts and outputs are returned in the RStudio console.

The user calls the function and gives their user credentials (username and email), the function will set these and print an output to confirm they have been set. The function will then create a hidden SSH folder and sub-folder named "id_ed25519" (~/.ssh/id_ed25519). Once done, it will then generate your SSH key and save it to this folder. The SSH key will then be printed in the console. The function then adds GitHub as a known host.

At this point, the function pauses and asks the user to confirm they have added the SSH key to GitHub. The user can then copy the SSH key from the console, go to GitHub -> settings -> SSH and GPG keys and add the SSH key.

Once added, the user can respond to the prompt and the function will finish establishing the connection. If all works as expected, the message "Hi username! You've successfully authenticated, but GitHub does not provide shell access" will be printed in the console.

This method of connection is only recommended for the DASH platform.

Value

No return value. Side effects include setting Git credentials, generating an SSH key, and establishing a GitHub connection.

Note

This function is interactive, pausing for user input before continuing. Therefore, this function is not suitable for automated scripts.

Author(s)

Josh Moatt (Joshua.Moatt@defra.gov.uk)

See Also

`connect_github_pat()`

Examples

```
## Not run:
# Set GitHub credentials
connect_github_ssh(
  username = "my_github_username",
  email = "my_email"
)

## End(Not run)
```

create_bands_df *Create bands*

Description

Creates a tibble of bands with corresponding lower and upper limits and labels.

Usage

```
create_bands_df(
  start = 0,
  upper_limits = NULL,
  open_ended = FALSE,
  codes = NULL,
  prefix = ""
)
```

Arguments

`start`	Numeric (default = 0); the start of the range
`upper_limits`	Numeric (default = `NULL`); the upper limits of each band, only used if `step` is not specified
`open_ended`	Logical (default = `FALSE`); whether to add a open-ended band as the last band (i.e. include all values greater than the upper limit)
`codes`	Numeric (default = `NULL`); the codes used in the dataset for each band - if NULL, will use a numeric sequence from 1 to the total number of bands
`prefix`	String; a prefix to apply to the band values (for example, "£" when the values are in GBP)

Value

A tibble with columns for the band codes, lower limits, upper limits and band labels

Author(s)

Farm Business Survey team (fbs.queries@defra.gov.uk)

See Also

Other functions for estimating means of banded data: `calculate_mcib()`, `calculate_riob()`, `calculate_rpme()`, `compute_band_counts()`, `estimate_band_means()`

Examples

```
create_bands_df(start = 1, upper_limits = c(10, 1000, 10000), prefix = "£")
create_bands_df(start = 0, upper_limits = c(10, 1000, 10000), open_ended = TRUE)
```

`create_dash_dir` *Create a directory in a DASH volume*

Description

Function to streamline creating directories in DASH volumes. This function will help avoid the http2 error which seems to be a frequent problem with `brickster`. In order for this function to work, you must be working on the Defra DASH platform and have set the required `brickster` environmental variables.

Usage

```
create_dash_dir(..., max_tries = 5, interval = 2)
```

Arguments

`...`	Additional arguments passed to `brickster::db_volume_dir_create()`.
`max_tries`	Maximum number of attempts to create directory. Default is 5.
`interval`	Interval between tries to read in data. Default is 2
`path`	A string containing the path on DASH catalog for the directory you wish to create. It should be the full DASH string starting "/Volumes/..."

Details

This function is designed to handle the frequent http2 errors that occur with `brickster`. From testing, these errors are not code or file path errors, but are just minor bugs with the API. Often, if the same code is rerun, the data will be read in fine.

This function uses the `brickster::db_volume_dir_create()` function. It uses a retry loop that catches errors with `tryCatch()`. If no error is thrown by `brickster`, the function will end as the directory will have been created. If an error is thrown by `brickster`, the function will wait a set number of seconds (controlled by the interval argument) before retrying. It will repeat the attempt to create the directory until either the directory is created successfully or the maximum number of attempts (set by max_tries) is reached - after which it throws an error.

In order for this function to work, you must be working on the Defra DASH platform and have set the required brickster environmental variables. These variables include setting a databricks Personal Access Token (PAT).

Value

Directory created in DASH catalog.

Author(s)

Josh Moatt (Joshua.Moatt@defra.gov.uk)

See Also

`brickster::db_volume_dir_create()`

Examples

```
## Not run:
# create directory
create_dash_dir(
  path = "/Volumes/prd_dash_lab/<volume-name>/<new-directory-name>"
)

## End(Not run)
```

`create_defra_stats_infographic`

Create stats infographic for sharing to Defra stats X account

---

Description

Uses grid to output an infographic into the Plots pane. To save it, click 'Export', choose 'Save as Image...' and save as a PNG file. Adjust the height and width to suit. NB: this function could use `grDevices::png()` rather than manually exporting, but testing this has resulted in corrupted files. If you can get it to work (including the option to edit height and width), please feel free to submit your fix in a pull request.

Usage

```
create_defra_stats_infographic(title, subtitle, source, key_points, icons)
```

Arguments

`title`	The filepath to find the icons listed in the `icons` argument
`subtitle`	The filepath to find the icons listed in the `icons` argument
`source`	The source of the data within the infographic
`key_points`	List of key points (use markdown formatting); between 3 and 5 points works best
`icons`	List of icons; the icons will go alongside each key point (for a list of the available icons, run `view_available_icons()`)

Details

Once you have saved your infographic, send it to the user engagement team. Include:

- The name of the publication
- A link to the publication

- The infographic .png file
- Plain text of the content of the infographic, which will be used in the alt text of the image

For example:

Hi User Engagement team, my team have just published 'Energy use on farms in England, 2023/24'. Please find the infographic attached. The text within the infographic reads:

Energy use on farms in England, 2023/24

- Red diesel was used by **98%** of farms
- **32% generate renewable energy** (mostly solar power)
- **84%** of solar-generating farm businesses have panels installed on farm building rooftops
- **20%** of farms had conducted a **carbon audit**

Source: Defra, Farm Business Survey England 2023/24

Author(s)

Farm Business Survey team (fbs.queries@defra.gov.uk)

Examples

```
title <- "Energy use on farms in England"
subtitle <- "2023/24"
source <- "Source: Defra, Farm Business Survey England 2023/24"
icons <- c("tractor", "sun", "barn", "form")
key_points <- list(
  "Red diesel was used<br>by **98%** of farms",
  "**32% generate renewable energy**<br>(mostly solar power)",
 "**84%** of solar-generating farm<br>businesses have panels installed on<br>farm building rooftops",
  "**20%** of farms had<br>conducted a **carbon audit**")

create_defra_stats_infographic(title, subtitle, source, key_points, icons)
```

`create_list` *Create a list to insert into markdown text*

Description

Takes a vector of strings and collapses them into a list; the last item is connected to the list with 'and', the rest of the items are separated by commas

Usage

```
create_list(
  string_vector,
  oxford_comma = FALSE,
  descriptor = NULL,
  last_connector = "and",
  case = c("original", "lower", "upper", "sentence", "title")
)
```

Arguments

`string_vector`	A string vector
`oxford_comma`	Logical (default = `FALSE`); should the last item be connected using an Oxford comma?
`descriptor`	A descriptor to add to the end of the list; if supplied, provide an object of length 1 (if always should be the same word), 2 (if you just require singular and plural options - in that order), or 3 (if you require options for one, two or more than two items - in that order)
`last_connector`	What word should connect the last two items? Default = `"and"`
`case`	Option to change the case of the output using `stringr::case()`

Author(s)

Farm Business Survey team (fbs.queries@defra.gov.uk)

See Also

Other commentary functions: `compare_two_years()`, `get_address()`, `get_diff_in_words()`, `get_pc_in_words()`

Examples

```
create_list(string_vector = c("apple", "banana", "pear"), descriptor = c("fruit", "fruits"))
create_list(string_vector = c("apple", "banana"), descriptor = c("fruit", "fruits"))
create_list(string_vector = c("apple", "banana", "pear"), oxford_comma = T, descriptor = c("fruit", "fruits"))
```

`create_readme` *Create a README using a template*

Description

Generates a README file for a project using a generic Quarto template. The function creates a `.qmd` file and performs an initial render to produce the output in the specified format.

Usage

```
create_readme(format, file_path = NULL, author = NULL, readme_title = NULL)
```

Arguments

`format`	A string specifying the output format. Options are `"markdown"` (default), `"html"`, or `"github"`.
`file_path`	A string specifying the file path where the README should be saved. Defaults to the working directory.
`author`	A string specifying the author's name. Defaults to `"add author"` if not provided.
`readme_title`	A string specifying the README title. Defaults to "README (edit title)" if not provided.

Details

This function helps standardize README creation across projects. It supports output formats `"markdown"` (default), `"html"`, and `"github"` (GitHub Flavored Markdown, or GFM). The README is saved in the working directory by default, but a custom location can be specified using the `file_path` argument.

The template includes sections for project introduction, structure, and instructions on how to run the project. The rendered output is created using `quarto::quarto_render()`.

Value

Creates a `.qmd` file and renders it to the specified format. The output file is saved in the specified location.

Author(s)

Josh Moatt (Joshua.Moatt@defra.gov.uk)

See Also

`quarto::quarto_render()`

Examples

```
## Not run:
# create README in active working directory
create_readme(
  format = "markdown",
  author = "Farming Stats",
  readme_title = "My Project README"
)

# create README in specified location
create_readme(
  format = "markdown",
  file_path = "~/my-project",
  author = "Farming Stats",
  readme_title = "My Project README"
)

## End(Not run)
```

`create_roxygen_script` *Create a new script using the default template.*

Description

This function will create a new R script with a roxygen2 template for use documenting functions. There are multiple options for customisation (see details).

Usage

```
create_roxygen_script(
  file_name = NULL,
  file_path = NULL,
  author = NULL,
  email = NULL
)
```

Arguments

`file_name`	string containing desired name for script.
`file_path`	string containing folder name to save script. This is built on the here function in R, so will follow your root directory. If you want to save in a sub-folder, enter the full folder sequence, e.g. "folder/sub-folder".
`author`	string containing author's name.
`email`	string containing author's email.

Details

This function generates a new R script pre-filled with a standard roxygen2 template.

You can optionally pre-fill the following fields:

- Script name
- Save location
- Author name
- Author email

The script is saved in your project's root directory (via `here::here()`) or a specified folder.

Value

An R script will be saved in the root directory or in the specified folder.

Author(s)

Josh Moatt (Joshua.Moatt@defra.gov.uk)

Examples

```
## Not run:
# Create a script with default settings
create_roxygen_script()

# Create a script with custom metadata
create_roxygen_script(
  file_name = "my_function_name",
  file_path = "R",
  author = "Farming Stats",
  email = "Farming.Stats@defra.gov.uk"
)

## End(Not run)
```

`create_script` *Create a new script using the default template.*

Description

This function will create a new R script using the default header template. There are multiple options for customisation (see details).

Usage

```
create_script(
  file_name = NULL,
  file_path = NULL,
  author = NULL,
  email = NULL,
  date = format(Sys.Date(), "%d/%m/%Y")
)
```

Arguments

`file_name`	string containing desired name for script.
`file_path`	string containing folder name to save script. This is built on the here function in R, so will follow your root directory. If you want to save in a sub-folder, enter the full folder sequence, e.g. "folder/sub-folder".

`author`	string containing author's name.
`email`	string containing author's email.
`date`	string containing a date. By default, this will be set as today's date.

Details

This function generates a new R script pre-filled with a standard header template. By default, all fields are left blank except for the creation date, which is set to today's date.

You can optionally pre-fill the following fields:

- Script name
- Save location
- Author name
- Author email
- Date created

The script is saved in your project's root directory (via `here::here()`) or a specified sub-folder.

Value

An R script will be saved in the root directory or in the specified folder.

Author(s)

Josh Moatt (Joshua.Moatt@defra.gov.uk)

Examples

```
## Not run:
# Create a script with default settings
create_script()

# Create a script with custom metadata
create_script(
  file_name = "analysis_script",
  file_path = "scripts",
  author = "Farming Stats",
  email = "Farming.Stats@defra.gov.uk"
)

## End(Not run)
```

`create_script_template` *Create a default R script template*

Description

Creates a default R script template to be applied as a header to all newly opened scripts. Supports default, custom, manual, and blank templates to help enforce consistent documentation practices.

Usage

```
create_script_template(format = "default", template = NULL, dash = FALSE)
```

Arguments

`format`	Character. One of `"default"`, `"custom"`, `"manual_edit"`, or `"blank"`. Determines the type of template to apply.
`template`	Character vector or NULL. Required if `format = "custom"`. Specifies the custom template content.
`dash`	Logical. Default is `FALSE`. If `TRUE`, uses the DASH platform path.

Details

This function sets a default header template for new R scripts, helping enforce best practices in documentation. Depending on the environment, the template is saved in:

- Local RStudio: `~/AppData/Roaming/RStudio/templates`
- DASH platform: `~/.config/rstudio/templates`

The function supports four modes:

- `"default"`: Applies a suggested header template with key metadata.
- `"custom"`: Applies a user-defined template provided as a string.
- `"manual_edit"`: Opens the template file for manual editing.
- `"blank"`: Removes the template, reverting to blank scripts.

Value

A new .R file containing the script template is created or modified.

Author(s)

Josh Moatt (Joshua.Moatt@defra.gov.uk)

Examples

```
## Not run:
# Set to default template on local install
create_script_template()

# create custom template
my_template <- c(
  "## Script name: ",
  "##",
  "## Purpose of script: ",
  "##",
  "## Author: ",
  "##",
  "## Date Created: "
)

# apply custom template in dash
create_script_template(
  format = "custom",
  template = my_template,
  dash = TRUE
)

# remove template
create_script_template(format = "blank")

## End(Not run)
```

`cut_long_scale` *Adapt scales::cut_long_scale to allow lower case*

Description

Adapt scales::cut_long_scale to allow lower case

Usage

```
cut_long_scale(lower = T, space = FALSE)
```

See Also

`scales::cut_long_scale()`

`decode_multi_choice_column` *Decode a multiple choice column*

Description

When a survey, such as the Farm Business Survey, requires all answers to be input as numbers, a geometric sequence (1, 2, 4, 8, 16, etc.) is used to code multiple choice answers. For example, the respondent choosing the first (code 1) and third (code 4) options would be input as 5. This method means that any combination of choices can be decoded into its constituent parts. Currently the maximum number of options for multiple choice is 12, 13 when including no answer. This function decodes these answers by taking an input dataset with a column of values and returning separate columns for each answer.

Usage

```
decode_multi_choice_column(
  input_data,
  var,
  id_cols,
  codes = NULL,
  colnames_option = "code_numbers",
  data_option = "code_numbers"
)
```

Arguments

`input_data`	Dataset
`var`	String; Column to decode
`id_cols`	vector of strings, columns with unique identifiers
`codes`	List; a list containing dataframes, the dataframes must be in the same order as the columns to decode, each dataframe must have a 2 columns one named code and the other named `description`. Default is null.
`colnames_option`	String; one of two options `code_numbers` or `code_descriptions`, default is `code_numbers`, the outputted colnames either be the code number or the names from the description column from imputed code dataframes.
`data_option`	String; one of three options `binary`, `code_numbers` or `code_descriptions`, default is `code_numbers`, determines the format of outputted decoded columns, they can be either in binary format, the original codes or the names from the description from the code dataframes.

Author(s)

Farm Business Survey team (fbs.queries@defra.gov.uk)

See Also

Other functions for decoding multiple choice answers: `decompose_multi_choice_column()`, `decompose_multi_choice_v`

`decompose_multi_choice_column`
Apply decompose_multi_choice_value to a column

Description

When a survey, such as the Farm Business Survey, requires all answers to be input as numbers, a geometric sequence (1, 2, 4, 8, 16, etc.) is used to code multiple choice answers. For example, the respondent choosing the first (code 1) and third (code 4) options would be input as 5. This method means that any combination of choices can be decoded into its constituent parts. This function decodes these answers by taking an input column of values and returning separate values for each answer.

Usage

```
decompose_multi_choice_column(column)
```

Arguments

`column`	Column to decompose

Author(s)

Farm Business Survey team (fbs.queries@defra.gov.uk)

See Also

Other functions for decoding multiple choice answers: `decode_multi_choice_column()`, `decompose_multi_choice_valu`

`decompose_multi_choice_value`
Decompose multiple choice value

Description

When a survey, such as the Farm Business Survey, requires all answers to be input as numbers, a geometric sequence (1, 2, 4, 8, 16, etc.) is used to code multiple choice answers. For example, the respondent choosing the first (code 1) and third (code 4) options would be input as 5. This method means that any combination of choices can be decoded into its constituent parts. This function decodes these answers by taking an input value and returning separate values for each answer.

Usage

```
decompose_multi_choice_value(n)
```

Arguments

`n`	Value to decompose

Author(s)

Farm Business Survey team (fbs.queries@defra.gov.uk)

See Also

Other functions for decoding multiple choice answers: `decode_multi_choice_column()`, `decompose_multi_choice_colu`

`easy_plot` *Use ggplot and afcharts to create accessible static charts*

Description

Has a large number of options so that the need to modify using ggplot code is minimised. Requires the y-axis to be continuous, but the x-axis can be of any scale.

Usage

```
easy_plot(
  plot_data,
  aesthetics,
  chart_type = c("stacked", "grouped", "line", "distribution", "boxplot"),
  chart_direction = c("vertical", "horizontal"),
  labels = F,
  label_size = 8,
  label_colour = "white",
  label_position = position_stack(0.5),
  error_bars = F,
  series_breaks = NULL,
  x_axis_title = NULL,
  y_axis_title = NULL,
  y_label_function = scales::label_comma(),
  y_axis_breaks = 8,
  y_min = 0,
  y_max = NA,
  wrap_vars = NULL,
  wrap_type = c("axis", "plot"),
  wrap_cols = NULL,
  wrap_rows = NULL,
  wrap_char_width = 25,
  font_family = "GDS Transport Website",
  font_size = 24,
  af_palette = names(afcharts::af_colour_palettes),
  custom_palette = NULL,
  gridlines = NULL,
  zero_line = TRUE,
  legend_position = NULL,
  legend_justification = NULL,
  legend_cols = NULL,
  annotate_lines = TRUE,
  top_margin = 10,
  right_margin = 0,
  bottom_margin = 0,
  left_margin = 0
)
```

Arguments

`plot_data`	Data to plot
`aesthetics`	Mapping aesthetics using `ggplot2::aes()`; if you want a line chart with a single group, remember to include group = 1
`chart_type`	One of "stacked", "grouped", "line", "distribution" or "boxplot": for a bar chart with only one group, both "stacked" and "grouped" will work ("stacked" is the default); for boxplots, if you supply `ymin`, `lower`, `middle`, `upper` and `ymax` to the aesthetics, `ggplot2::geom_boxplot()` will use `stat = "identity"`, otherwise it will use `stat = "boxplot"`
`chart_direction`	One of "vertical" or "horizontal" ("horizontal" uses `ggplot2::coord_flip()`; keep in mind that this will, by design, reverse the order of items on the x-axis - this can be fixed using `rev()` in `aesthetics` or by reversing the factor levels in `plot_data`)
`labels`	If TRUE, adds `ggplot2::geom_text()` labels
`label_size`	Default = 8; Font size of labels
`label_colour`	Default = white; Colour of labels
`label_position`	Default = `position_stack(0.5)` for labelling stacked bar charts; Can take a ggplot `position_*` or a length 2 vector referring to horizontal and vertical justification, respectively, e.g. `c(0, 0.5)` or `c(NULL, 1)`
`error_bars`	If TRUE, adds confidence intervals; for bar charts uses `ggplot2::geom_errorbar()`, and for line charts uses `ggplot2::geom_ribbon()`
`series_breaks`	If supplied, will use the xintercept argument of `ggplot2::geom_vline()` to add breaks to the x-axis at the specified points; for line charts, remember to set the `group` argument in `ggplot2::aes()` to ensure the lines break properly (for a single group, use the column containing the series names, or for multiple groups, an `interaction()` between the grouping column and the series column)
`x_axis_title`	If supplied, adds a title to the x-axis
`y_axis_title`	If supplied, adds a title to the y-axis

`y_label_function`	Default = `scales::label_comma()`; a function which is applied to the y-axis labels within the `ggplot2::scale_y_continuous()` function (set to `NULL` to return the default labels)
`y_axis_breaks`	Default = 8; the number of breaks to show on the y-axis
`y_min`	Default = 0; The minimum value of the y-axis (must be either 0 or below 0)
`y_max`	Default = NA; The maximum value of the y-axis
`wrap_vars`	If supplied, will feed into the facets argument of `ggplot2::facet_wrap()`
`wrap_type`	One of "axis" (to return one plot with a wrapped axis) or "plot" (to return multiple plots)
`wrap_cols`	Feeds into the ncol argument of `ggplot2::facet_wrap()`; only used with `wrap_type = "plot"`
`wrap_rows`	Feeds into the nrow argument of `ggplot2::facet_wrap()`; only used with `wrap_type = "plot"`
`wrap_char_width`	Default = 25; Wraps facet labels to the specifies number of characters
`font_family`	Default = "GDS Transport Website"; This font must be loaded for it to be used, and should only be used when publishing on gov.uk
`font_size`	Default = 24; Font size for all chart text (except bar labels)
`af_palette`	Which one of the afcharts::af_colour_palettes to use (default = `"main"`); for boxplots, the dark colours do not have enough contrast with the boxplot lines so these are filtered out, giving a maximum of three colours available
`custom_palette`	A vector of colours; If supplied, will use this rather than any of the AF palettes (useful if you need to reorder an AF palette or need more than 6 colours)
`gridlines`	Fed into the grid argument of `afcharts::theme_af()`; If supplied, will override defaults (i.e. "y" for vertical bar charts, "x" for horizontal bar charts, and "xy" for line charts)
`zero_line`	Default = TRUE; If set to FALSE, will remove origin line (only do this if your y-axis does not include 0)
`legend_position`	If supplied, will override defaults (i.e. "right" for vertical charts and "top" for horizontal charts)
`legend_justification`	If supplied, will override defaults (i.e. "top" for vertical charts and "right" for horizontal charts)
`legend_cols`	Fed into the ncol argument of `ggplot2::guide_legend()`
`annotate_lines`	Uses `ggrepel::geom_text_repel()` to annotate lines in line charts, instead of using a legend; set to `FALSE` to use a legend instead, noting that it is more accessible if the resulting legend is in the order of the lines (see examples for how to do this)
`top_margin`	Default = 10; Top margin of plot, fed into the plot.margin argument of `ggplot2::theme()` using `ggplot2::margin()`
`right_margin`	Default = 0; Right margin of plot
`bottom_margin`	Default = 0; Bottom margin of plot
`left_margin`	Default = 0; Left margin of plot

Author(s)

Farm Business Survey team (fbs.queries@defra.gov.uk)

Examples

```
# Creating a line chart without annotating the lines
# but correcting the legend order

test_df <- dplyr::tibble(
  letters = rep(LETTERS[1:4], 3),
  grouping = c(rep("1", 4), rep("2", 4), rep("3", 4)),
  numbers = c(seq(1000, 4000, 1000),
              seq(1250, 4250, 1000),
              seq(1500, 4500, 1000))
)

var_order <- dplyr::filter(test_df, letters == max(letters)) |>
  dplyr::arrange(desc(numbers)) |>
  dplyr::pull(grouping)

plot_df <- test_df |>
  dplyr::mutate(grouping = factor(grouping, levels = var_order))

easy_plot(plot_df, aes(x = letters, y = numbers,
                       colour = grouping, group = grouping),
          chart_type = "line", annotate_lines = FALSE)
```

`estimate_band_means` *Apply Robust Integration Over Brackets*

Description

Combines Mean Constrained Integration Over Brackets and Robust Pareto Midpoint Estimator to estimate the mean value of each band, in banded data. Can be done by groups, e.g. year, or year and region, using the `grouping_cols` argument.

Usage

```
estimate_band_means(
  df,
  bands_df,
  band_col,
  grouping_cols = "year",
  weights = "weight",
  method = c("Geometric", "Arithmetic", "Harmonic", "Median"),
  min_alpha = 1.11
)
```

Arguments

`df`	A data frame containing the survey data
`bands_df`	A table of bands, corresponding with the values in `band_col`, created using `create_bands_df()`; fed into `compute_band_counts()`, and `calculate_mcib()` via `calculate_riob()`
`band_col`	String; The name of the column in `df` with bands in, fed into `compute_band_counts()`
`grouping_cols`	String (default = "year"); Columns to group the counts by, along with band, fed into `compute_band_counts()` (if no groups, set to `NULL`)
`weights`	String (default = "weight"); The name of the column containing the weights, fed into `compute_band_counts()`
`method`	String; one of "Arithmetic", "Harmonic", "Geometric" or "Median", fed into `calculate_rpme()` via `calculate_riob()`
`min_alpha`	Numeric; the minimum threshold for the alpha parameter of the Pareto distribution, fed into `calculate_rpme()` via `calculate_riob()`

Details

Mean Constrained Integration Over Brackets: `https://journals.sagepub.com/doi/10.1177/0081175018782579#sec-4`

We don't have a grand mean, so we only do the integration step.

Robust pareto midpoint estimator: `https://ideas.repec.org/c/boc/bocode/s457962.html` `https://arxiv.org/ftp/arxiv/papers/1402/1402.4061.pdf`

The default method is geometric and the default min_alpha value is 1.11, which are the ideal options for the FBS. These were chosen by calculating the Pareto Distribution using min_alpha values from 0-4 in 0.01 increments for each of the four mean methods. The method with the lowest errors was chosen as the default and of that method, the alpha value with an error value closest to zero was chosen as the default min_alpha value. This will need to be updated if you are not using FBS data, or the FBS sample size has significantly changed.

Author(s)

Farm Business Survey team (fbs.queries@defra.gov.uk)

See Also

Other functions for estimating means of banded data: `calculate_mcib()`, `calculate_riob()`, `calculate_rpme()`, `compute_band_counts()`, `create_bands_df()`

fix_suppression — *Fix suppression in tables where totals can be used to calculate suppressed values*

Description

This function finds which figures need to be secondarily suppressed and overwrites their sample size to one - any subsequent suppression code will then suppress these figures.

Usage

```
fix_suppression(
  df,
  groups,
  sample_size_col = "sample_size",
  export_path = "",
  file_name = "",
  pivot_col = NULL,
  save_excel_file = FALSE
)
```

Arguments

`df`	Data to fix
`groups`	A list of grouping dimensions, e.g. `list("group", c("band", "grouping_factor"))` would group by the `group` column in the first dimension and by the `band` and `grouping_factor` columns in the second dimension
`sample_size_col`	Default = `"sample_size"`; The name of the sample size column
`export_path`	File path; where the checking files should be saved
`file_name`	What should the checking file be called? Don't include file extension
`pivot_col`	String; If you want the manual checking table to be pivoted, which column should it be pivoted on?
`save_excel_file`	default is TRUE, set to FALSE when one does not wish to save an excel file

Details

This will loop though the group, or list of groupings, until there are no more figures which need to be suppressed.

Sometimes this function can oversuppress, therefore, it also outputs an excel file for manual checking. This file has the unsuppressed data in twice so that can manually suppress one while comparing it to the unsuppressed data. The file also has conditional formatting - values between 1 and 4 are highlighted in red and values between 5 and 14 are highlighted in yellow.

If doing suppression across multiple groups, it is sometimes useful to consider the order of the groups to minimize oversuppression.

Author(s)

Farm Business Survey team (fbs.queries@defra.gov.uk)

`get_address` *Get full URL for file attachments or links on gov.uk*

Description

Get full URL for file attachments or links on gov.uk

Usage

```
get_address(
  web_address,
  file_type = NULL,
  file_number = 1,
  search_term = NULL,
  find_csv_preview = F
)
```

Arguments

`web_address`	Webpage to look for the file on
`file_type`	File extension to search for (e.g. "xlsx", "ods"); default = `NULL`, leave as NULL if searching for a link rather than a file
`file_number`	If there are multiple files of the same type as `file_type` on the page, which one to return
`search_term`	Regex; Optional, used by `str_subset` to filter on link URLs
`find_csv_preview`	Logical, default = `FALSE`; when searching for a CSV on gov.uk, this will return the link to the CSV preview when set to TRUE, and will return the link to the actual CSV when set to FALSE

Author(s)

Agriculture in the UK team (AUK_stats_team@defra.gov.uk)

Farm Business Survey team (fbs.queries@defra.gov.uk)

See Also

Other commentary functions: `compare_two_years()`, `create_list()`, `get_diff_in_words()`, `get_pc_in_words()`

`get_cell_style` *Get cell styles for stats publications*

Description

Get cell styles for stats publications

Usage

```
get_cell_style(
  style1 = c("text", "number", "percent", "custom"),
  style2 = c("body", "heading", "highlight", "lastrow"),
  bold = FALSE,
  separator = FALSE,
  custom_format = NULL,
  indent = 0
)
```

Arguments

`style1`	Whether the data is text, numbers or percentages. Text cells get left-aligned, while number and percentage cells get right-aligned. By default, both percentages and numbers are rounded to a whole number; to round to a number of decimal places, define `style1` to `"custom"` and specify the `custom_format` argument.
`style2`	Where the data is in the table (heading row, body, average row or last row). Heading cells are bold, aligned vertically in the middle, and get upper and lower borders. All other cells are bottom-aligned. 'Highlight' cells get upper and lower borders, while 'last row' cells just get lower borders.
`bold`	logical, default = `FALSE`; should the text be bold?
`separator`	logical, default = `FALSE`; should there be a separating border to the right of the column?
`custom_format`	Used by numFmt argument of `openxlsx::createStyle()`; an Excel custom number format, e.g., "##0.0%" rounds a percentage to 1 decimal place.
`indent`	Numeric, default = 0; Horizontal indentation of cell contents.

Author(s)

Farm Business Survey team (fbs.queries@defra.gov.uk)

`get_diff_in_words` *Get differences in text for statistics commentary*

Description

There are a six output options to choose from:

1. Past tense, displays the percentage change as text as well as the `curr_rnd` value, e.g. `"increased by 10% to 100"`
2. Present tense, displays the percentage change as text as well as the `curr_rnd` value, e.g. `"increasing by 10% to 100"`
3. Reformats option 2 to show the `curr_rnd` value first, e.g. `"100, an increase of 10%"`
4. Uses 'higher'/'lower' to show only the percentage change, e.g. `"10% higher"`; no need to set the `curr_rnd` argument
5. Uses 'rise'/'fall' to show only the percentage change, e.g. `"a rise of 10%"`; no need to set the `curr_rnd` argument
6. Uses 'rose'/'fell' to show only the percentage change, e.g. `"rose by 10%"`; no need to set the `curr_rnd` argument

Usage

```
get_diff_in_words(option_num, pc, pc_rnd, curr_rnd = NULL, points = FALSE)
```

Arguments

`option_num`	Numeric; the text option to return
`pc`	Column containing the proportional difference between the two values you are comparing
`pc_rnd`	The rounded values from the column defined in the pc argument
`curr_rnd`	Column containing the rounded values for the most recent data
`points`	Logical, default = `FALSE`; is the difference in percentage points rather than %?

Author(s)

Farm Business Survey team (fbs.queries@defra.gov.uk)

See Also

Other commentary functions: `compare_two_years()`, `create_list()`, `get_address()`, `get_pc_in_words()`

Examples

```
get_diff_in_words(1, .1, "10%", curr_rnd = "100")
get_diff_in_words(2, .1, "10%", curr_rnd = "100")
get_diff_in_words(3, .1, "10%", curr_rnd = "100")
get_diff_in_words(4, .1, "10%")
get_diff_in_words(5, .1, "10%")
get_diff_in_words(6, .1, "10%")
```

get_ons_series *Read in quarterly ONS price indices and convert to annual series*

Description

Retrieves an ONS inflation / price indices series. The URLs for the GDP and CPIH series are included, or you can specify your own. Once the series has been retrieved, you can convert a current terms value into real terms using `value / index * 100`, where `index` is the price / deflator index for the year the value corresponds to (see examples).

Usage

```
get_ons_series(
  series_years,
  index = c("", "GDP", "CPIH"),
  ons_url = NULL,
  get_snapshot = F,
  snapshot_date = NULL,
  save_path = NULL,
  year_end_q = 4
)
```

Arguments

`series_years`	The years to get the ONS indices for (for financial years, just the first year, e.g. 2020 for 2020/21)
`index`	The index to use; either GDP or CPIH, or leave blank to download another series (must provide a URL)
`ons_url`	If provided, will download the CSV file at this URL to create the indices; can be the full link to the inflation / price indices series, just the URL extension, or the CSV generator (see details section)
`get_snapshot`	Logical (default = `FALSE`); if `TRUE`, allows you to choose a snapshot, rather than the live data
`snapshot_date`	Optional date of the desired snapshot, must be a string with format YYYY-MM-DD (i.e. coercible by base::as.Date); if not provided, you will be prompted to choose a snapshot from a list of the latest ten
`save_path`	If provided, will save the downloaded data to the specified directory

`year_end_q`	When converting the index from quarterly to annual, the year-end to use, i.e. the last quarter of the year; default is 4, which converts to calendar years (for financial years, set as 1)

Details

As an example, for the ’CPI ANNUAL RATE 00: ALL ITEMS 2015=100’ series, the following URL options will work:

- https://www.ons.gov.uk/economy/inflationandpriceindices/timeseries/d7g7/mm23
- economy/inflationandpriceindices/timeseries/d7g7/mm23
- https://www.ons.gov.uk/generator?format=csv&uri=economy/inflationandpriceindices/timeseries/d7g7/mm23

Value

A tibble containing the ONS inflation / price indices series for all years in `series_years` (for clarity, a column of type `yearqtr` showing the chosen year end is also included)

Author(s)

Farm Business Survey team (fbs.queries@defra.gov.uk)

Examples

```
library(dplyr)
set.seed(1)

current_terms <- tibble(
  year = 2015:2020,
  prices = "Current",
  value = runif(n = 6, min = 100, max = 1000)
)

gdp_series <- get_ons_series(2015:2020, index = "GDP")

real_terms <- left_join(current_terms, gdp_series,
                        by = "year") %>%
  mutate(prices = "Real",
         value = value / index * 100)

bind_rows(current_terms, real_terms)

(current_terms$value / real_terms$index) * 100
```

`get_pc_in_words` *Get percentage in text*

Description

Converts percentages to text, e.g. `0.25` will be converted to `"a quarter"`

Usage

```
get_pc_in_words(pc, pc_rnd)
```

Arguments

`pc`	Numeric; The percentage as a proportion, e.g. `0.1` for a value of 10%
`pc_rnd`	String; What to show if `pc` isn't converted to text, e.g. `"10%"`

Author(s)

Farm Business Survey team (fbs.queries@defra.gov.uk)

See Also

Other commentary functions: `compare_two_years()`, `create_list()`, `get_address()`, `get_diff_in_words()`

`get_user_input` *Wrap the readline function so it can be tested*

Description

Wrap the readline function so it can be tested

Usage

```
get_user_input(prompt = "")
```

`normalise_values` *Normalise values and check result*

Description

Uses `bestNormalize::bestNormalize()` to select the best normalising transformation for a vector of numeric values, then performs this transformation. Will also creates a plot with two charts showing the distributions of the untransformed and transformed variables, including the name of the transformation used and the skewness value of the result. If `save_path` is set, the plot will be saved as a PNG to the specified directory.

Usage

```
normalise_values(df, model_col, save_path = NULL, leave_one_out = TRUE, ...)
```

Arguments

`df`	A dataframe with a column you are running a model on
`model_col`	String; the name of the column you are running a model on
`save_path`	The directory where the plot should be saved to
`leave_one_out`	Passed to the `loo` argument of `bestNormalize::bestNormalize()`
`...`	Other arguments passed to `bestNormalize::bestNormalize()`

Author(s)

Farm Business Survey team (fbs.queries@defra.gov.uk)

See Also

Other survey modelling functions: `run_wald_test()`, `summarise_variables()`, `test_model_performance()`

`overwrite_num_cols` *Fixes column where numbers are stored as text*

Description

Overwrites selected columns and rows that contain both numeric and character elements. Once this is run, you should add a new style to the modified columns and rows so that they are displayed correctly in the output file. Use `openxlsx::addStyle()` and `get_cell_style()` to do this.

Usage

```
overwrite_num_cols(excel_wb, sheet, cols, rows, df)
```

Arguments

`excel_wb`	Openxlsx workbook name
`sheet`	Worksheet (either name as string, or location as numeric)
`cols`	Vector of column numbers to be overwritten
`rows`	Vector of row numbers to be overwritten
`df`	Data frame containing the data from the relevant worksheet

Details

This function is adapted from `rapid.spreadsheets::overwrite_df()`, to work with aftables::aftables; for the full rapid.spreadsheets code, see the GitHub page.

Value

Updated workbook with modified columns

Author(s)

Farm Business Survey team (fbs.queries@defra.gov.uk)

Examples

```
## Not run:
library(openxlsx)
library(aftables)

set.seed(1)

# Create an aftable
cover_df <- list("Section" = c("Title", "Content"))

contents_df <- data.frame("Sheet name" = "Table",
                          "Sheet title" = "Example",
                          check.names = FALSE)

table_df <- data.frame(
  Category = LETTERS[1:10],
  "Suppressed" = c(1:4, "[c]", 6:9, "[x]"),
  "Commas" = round_with_commas(rnorm(10) * 1e5, "optimise"),
  check.names = FALSE
)

aftable <- create_aftable(
  tab_titles = c("Cover", "Contents", contents_df$`Sheet name`),
  sheet_types = c("cover", "contents", "tables"),
  sheet_titles = c("Cover", "Contents", "Table"),
  sources = c(rep(NA_character_, 2), "Source"),
  tables = list(cover_df, contents_df, table_df))

excel_wb <- generate_workbook(aftable)
```

```
# Check the file
# note the format errors on the table sheet
openXL(excel_wb)

# Fix the errors
overwrite_num_cols(excel_wb, sheet = 3, cols = 2:3,
                   rows = 5:14, df = table_df)

# Check the file again
# the errors are gone, but the commas have disappeared
openXL(excel_wb)

# Add styling
addStyle(excel_wb, sheet = 3, cols = 2:3,
         rows = 5:14, gridExpand = TRUE,
         style = get_cell_style("number", "body"))

# Check the file a final time
# formatting is back and errors are still gone
openXL(excel_wb)

## End(Not run)
```

query_dash_files *Query files within a DASH directory*

Description

Query files in a DASH directory (volume or folder) using the `brickster` API. Designed to handle intermittent http2 errors by retrying failed attempts. Requires the Defra DASH platform and appropriate `brickster` environment variables.

Usage

```
list_dash_files(path, max_tries = 5, interval = 2)

exists_dash_dir(..., max_tries = 5, interval = 2)

exists_dash_file(..., max_tries = 5, interval = 2)
```

Arguments

`path`	A string containing the path to the volume or folder. Should be the full DASH string starting "/Volumes/..."
`max_tries`	Maximum number of tries to access directory metadata. Default is 5 attempts.
`interval`	Interval between tries to read in data. Default is 2 seconds.

Details

These functions wrap `brickster::db_volume_list()`, `brickster::db_volume_dir_exists()`, and `brickster::db_volume_file_exists()` with a retry loop to handle intermittent http2 errors, which are common but typically transient. These errors are not usually caused by incorrect code or file paths, and rerunning the same request often succeeds.

The function will retry the request up to `max_tries` times, waiting `interval` seconds between each attempt. If all attempts fail, an error is thrown.

You must be working on the Defra DASH platform and have set the required `brickster` environment variables, including a valid Databricks Personal Access Token (PAT).

Value

List of files in DASH directory returned, or boolean (for exists fns).

Author(s)

Josh Moatt (Joshua.Moatt@defra.gov.uk)

See Also

`brickster::db_volume_list()`, `brickster::db_volume_dir_exists()`, `brickster::db_volume_file_exists()`

Examples

```
## Not run:
# list files in volume/directory
list_dash_files(
  path = "/Volumes/prd_dash_lab/<volume-name>/<directory-name>"
)

exists_dash_dir(
  path = "/Volumes/prd_dash_lab/<volume-name>/<directory-name>"
)

exists_dash_file(
  path = "/Volumes/prd_dash_lab/<volume-name>/<directory-name>/<file-name>"
)

## End(Not run)
```

`read_file_from_volume` *Read data of various file types from the DASH data lake*

Description

A set of convenience functions to read data from common file formats from the DASH platform. Each function uses `brickster` and wraps the appropriate reader function for each file type. These functions will avoid the relatively frequent http2 errors that crop up with `brickster`. See details for more info on how they solve this problem. In order for these functions to work, you must be working on the Defra DASH platform and have set the required brickster environmental variables. There a specific functions for reading in .csv, .xlsx, .Rds, and .ods files as well as a generic function for reading in additional file types.

Usage

```
read_file_from_volume(path, ext, max_tries = 5, interval = 2)

read_csv_from_volume(path, ..., max_tries = 5, interval = 2)

read_rds_from_volume(path, ..., max_tries = 5, interval = 2)

read_xlsx_from_volume(path, ..., max_tries = 5, interval = 2)

read_ods_from_volume(path, ..., max_tries = 5, interval = 2)

read_rio_from_volume(path, ..., max_tries = 5, interval = 2)

read_parquet_from_volume(path, ..., max_tries = 5, interval = 2)
```

Arguments

`path`	A string containing the path to data to be read in. Should be the full DASH string starting "/Volumes/..."
`ext`	A string specifying the file type. Only used in the general function. for the specific functions, this will be set to the correct extension (e.g. .csv).
`max_tries`	Maximum number of tries to read in data. Default is 5 attempts.
`interval`	Interval between tries to read in data. Default is 2 seconds.
`...`	Additional arguments passed to the appropriate reader function.

Details

These functions are designed to handle the frequent http2 errors that occur with `brickster`. From testing, these errors are not code or file path errors, but are just minor bugs with the API. Often, if the same code is rerun, the data will be read in fine.

There is one generic function which can be used to create a temporary file for any file type (using the ext argument). There are also wrapper functions for specifically reading in .csv, .Rds, and .xlsx files.

All of the functions work in the same way. They use `brickster::db_volume_read()`, to attempt to read in the data and store it in an appropriate temporary file. They use a retry loop that catch errors with `tryCatch()`. If no error is thrown by `brickster`, the functions read in the temporary file into R using an appropriate reader function and returns the resulting data frame. If an error is thrown by `brickster`, the functions will wait a set number of seconds (controlled by the interval argument) before retrying. They will repeat the attempted data load until either the file is read successfully or the maximum number of attempts (set by max_tries) is reached - after which it throws an error.

In order for these functions to work, you must be working on the Defra DASH platform and have set the required `brickster` environmental variables. These variables include setting a databricks Personal Access Token (PAT). For more information on how to do this, see the specific project README.

Reader functions used are as follows:

- **.csv** - `readr::read_csv()`
- **.Rds** - `readr::read_rds()`
- **.ods** - `readODS::read_ods()`
- **.xlsx (single worksheet)** - `readxl::read_xlsx()`
- **.xlsx (whole workbook)** - `rio::import_list()`
- **.parquet** - arrow::read_parquet

Value

A dataframe.

Author(s)

Josh Moatt (Joshua.Moatt@defra.gov.uk)

See Also

`brickster::db_volume_read()`, `readr::read_csv()`, `readr::read_rds()`, `readxl::read_xlsx()`, `readODS::read_ods()`, `read_csv_from_volume()`, `read_xlsx_from_volume()`, `read_rds_from_volume()` `rio::import_list()`, `read_parquet_from_volume()`

Examples

```
## Not run:
# read Rds file
dat <- read_rds_from_volume(
  path = "/Volumes/prd_dash_lab/<path-to-file>/filename.Rds"
)

# read csv file
dat <- read_csv_from_volume(
  path = "/Volumes/prd_dash_lab/<path-to-file>/filename.csv",
  show_col_types = FALSE
)

# read ods file
dat <- read_ods_from_volume(
  path = "/Volumes/prd_dash_lab/<path-to-file>/filename.ods",
  sheet = "sheet-name"
)

# read xlsx single woksheet
dat <- read_xlsx_from_volume(
  path = "/Volumes/prd_dash_lab/<path-to-file>/filename.xlsx",
  sheet = "sheet-name"
)

dat <- read_rio_from_volume(
  path = "/Volumes/prd_dash_lab/<path-to-file>/filename.xlsx"
)

dat <- read_parquet_from_volume(
  path = "/Volumes/prd_dash_lab/<path-to-file>/filename.parquet"
)

## End(Not run)
```

round_with_commas *Round numbers for publishing/sharing*

Description

Round values with `janitor::round_half_up()` (as opposed to the R default 'round-to-even' method) and add comma separators.

Usage

```
round_with_commas(
  x,
  method = c("optimise", "round_to"),
  optimise_to = c("1", "100", "1k", "10k", "100k", "1m", "10m", "100m"),
  round_to = NULL,
  round_zeros = TRUE,
  ...
)
```

Arguments

x	The number to round

`method`	Either `"round_to"` for rounding all numbers to a power of 10 (must supply `round_to` argument), or `"optimise"` for rounding smaller numbers to more detail and larger numbers to less detail (see details)
`optimise_to`	The maximum value values are rounded to, as a string (1, or powers of 10 from 100 to 100 million, see details)
`round_to`	numeric; if method is set to `"round_to"`, provide any power of 10 (including minus powers, e.g. 0.1)
`round_zeros`	When `method = "optimise"`, should zeros be shown with no decimal places (`TRUE`), or to 1 decimal place (`FALSE`)?
`...`	Optional arguments to `scales::comma()` (other than `accuracy`, which this function takes care of)

Details

'Optimise' rounds smaller numbers to more detail and larger numbers to less detail.

When `optimise_to` is set to `"1"`:

- 0 is shown as 0
- Values > 0 and < 10 are rounded to 1 decimal place
- Values >= 10 are rounded to the nearest 1

When `optimise_to` is set to `"100"` the above plus:

- Values >= 10 and < 100 are rounded to the nearest 1
- Values >= 100 are rounded to the nearest 100

When `optimise_to` is set to `"1k"` (i.e. 1,000) the above plus:

- Values >= 100 and < 1,000 are rounded to the nearest 100
- Values >= 1,000 are rounded to the nearest 1,000

And so on, for `"10k"` (10,000), `"100k"` (100,000), `"1m"` (1,000,000), `"10m"` (10,000,000), and `"100m"` (100,000,000),

Value

A rounded value with comma separators, plus optional prefix and suffix

Author(s)

Farm Business Survey team (fbs.queries@defra.gov.uk)

`run_wald_test` *Wald test*

Description

Perform the Wald test on a survey model with `survey::regTermTest()`.

Usage

```
run_wald_test(survey_model)
```

Arguments

`survey_model`	A svyglm object

Author(s)

Farm Business Survey team (fbs.queries@defra.gov.uk)

See Also

Tested on models created with `survey::svyglm()`

Other survey modelling functions: `normalise_values()`, `summarise_variables()`, `test_model_performance()`

`set_console_prompt` *Set a custom R console prompt with Git branch*

Description

Sets a custom R console prompt that displays the active Git branch (e.g., `[@main]>`) when working inside a Git-enabled project. If no Git repository is detected, the prompt defaults to `">"`.

The prompt is added to your .Rprofile file and can be scoped either globally (`"user"`) or to the current project (`"project"`). You can manually edit the profile using `usethis::edit_r_profile()`.

Usage

```
set_console_prompt(scope)
```

Arguments

`scope`	Character. Either `"user"` to apply the prompt globally, or `"project"` to apply it only to the current project.

Details

This function modifies your R console prompt to reflect the current Git branch, helping you stay aware of your working context. It works by appending a custom `.First()` function to your .Rprofile, which sets the prompt and attaches a task callback to keep it updated.

If a custom prompt already exists in the profile, the function will abort to avoid overwriting it. You can manually edit the profile if needed.

Value

No return value. Side effect: modifies the `.Rprofile` file to set the prompt.

Author(s)

Josh Moatt (Joshua.Moatt@defra.gov.uk)

See Also

`usethis::edit_r_profile()`

Examples

```
## Not run:
# set user prompt
set_console_prompt("user")

# set project prompt
set_console_prompt("project")

## End(Not run)
```

`set_databricks_pat` *Prompt user to set Databricks Personal Access Token (PAT)*

Description

Prompts the user to enter their Databricks Personal Access Token (PAT) interactively. This avoids hardcoding sensitive credentials in scripts, reducing the risk of accidentally committing them to version control.

This function is useful when working on the Defra DASH platform with the `brickster` package. It is not required on Posit Workbench, which handles credentials internally, and does not work in Databricks notebooks.

Usage

```
set_databricks_pat()
```

Value

A character string containing the entered PAT. Typically assigned to an environment variable using `Sys.setenv(DATABRICKS_TOKEN = token)`.

Author(s)

Josh Moatt (Joshua.Moatt@defra.gov.uk)

Examples

```
## Not run:
# set PAT
token <- set_databricks_pat()

# set to environmental variables
Sys.setenv(DATABRICKS_TOKEN = token)

## End(Not run)
```

`set_rstudio_layout` *Customise RStudio IDE layout and preferences*

Description

Customises the RStudio IDE by updating user or project-specific preferences. Especially useful on the DASH RStudio server after a cluster restart, allowing quick restoration of preferred settings.

A full list of modifiable settings is available on the Posit website: Session User Settings.

Usage

```
set_rstudio_layout(...)
```

Arguments

`...`	Named arguments representing RStudio preferences to update. Each argument should correspond to a valid RStudio preference name and its desired value.

Value

No return value. Preferences are updated as a side effect.

Author(s)

Josh Moatt (Joshua.Moatt@defra.gov.uk)

See Also

`rstudioapi::readRStudioPreference()`, rstudioapi::writeRStudioPreference

Examples

```
## Not run:
# set pane layout
my_pane_layout <- list(
 quadrants = list(
   "Source",
   "TabSet1",
   "Console",
   "TabSet2"
 ),
 tabSet1 = list(
   "History",
   "Presentation"
 ),
 tabSet2 = list(
   "Environment",
   "Files",
   "Plots",
   "Connections",
   "Packages",
   "Help",
   "Build",
   "VCS",
   "Tutorial",
   "Viewer",
   "Presentations"
 ),
 hiddenTabSet = list(),
 console_left_on_top = FALSE,
 console_right_on_top = TRUE,
 additional_source_columns = 0
)

# apply preferences
set_rstudio_layout(
  always_save_history = FALSE, # don't auto save history
  save_workspace = "never", # don't save workspace
  load_workspace = FALSE, # don't load previous workspace
  restore_last_project = FALSE, # don't restore last opened project
  continue_comments_on_newline = TRUE, # continues comment on new line
  highlight_selected_line = FALSE, # highlight line cursor is on
  highlight_r_function_calls = TRUE, # highlight R function calls
  show_margin = FALSE, # don't show margin (default = 80 characters)
  rainbow_parentheses = TRUE, # colour match brackets
  color_preview = TRUE, # hexcode previews on
  panes = my_pane_layout # Pane layout as set above
)

## End(Not run)
```

`str_line_wrap` *Wrap strings to a set number of lines*

Description

Wrap strings to a set number of lines

Usage

```
str_line_wrap(string, lines)
```

Author(s)

Farm Business Survey team (fbs.queries@defra.gov.uk)

`summarise_variables` *Summarise variable to be modelled*

Description

Summarise variable to be modelled

Usage

```
summarise_variables(df, column)
```

Arguments

`df`	A data set
`column`	String; column to be summarised

Author(s)

Farm Business Survey team (fbs.queries@defra.gov.uk)

See Also

Other survey modelling functions: `normalise_values()`, `run_wald_test()`, `test_model_performance()`

test_model_performance *Check the sensitivity and specificity of a svyglm model*

Description

Currently only takes models with a continuous or binary response variable; could be adapted for models with categorical response variables. If `save_path` is set, the plot will be saved as a PNG to the specified directory.

Usage

```
test_model_performance(survey_model, save_path = NULL)
```

Arguments

`survey_model`	A svyglm object
`save_path`	The directory where the plot should be saved to

Author(s)

Farm Business Survey team (fbs.queries@defra.gov.uk)

See Also

Tested on models created with `survey::svyglm()`

Other survey modelling functions: `normalise_values()`, `run_wald_test()`, `summarise_variables()`

tidy_log *Tidy up a log created by futile.logger*

Description

The messages you have written using futile.logger::futile.logger should have the following format: "message: detail", e.g. `flog.info("Rows in data: %s", nrow(data))`. Or they can be a message without detail, e.g. `flog.warn("Data file not found")`.

Usage

```
tidy_log(log_path, export_path = log_path, export = TRUE, delim = "|")
```

Arguments

`log_path`	the file path where the futile logger log is saved
`export_path`	the path to write the tidy log to; the default is the current location of the log, i.e., will overwrite the current log
`export`	logical; if `TRUE` (default) will export the tidied log as a csv file, otherwise will return it to the console
`delim`	the delimiter used in the futile logger log; the default, `"|"`, should match any futile logger log, but the option has been given in case futile.logger changes

Details

Ensure that there is only one `": "` in the message, as this string is used to split the columns. This function will separate the log into four columns:

- `message_type`
- `timestamp`
- `message`
- `detail`

If more than one `": "` is used, everything after the first colon will end up in the detail column. If you have not included `": "` in your log message, the detail column will be populated with `NA`.

Author(s)

Farm Business Survey team (fbs.queries@defra.gov.uk)

`uc_volume_get` *Download a file from the DASH Unity Catalog*

Description

Downloads a file from the Databricks Unity Catalog (UC) to the RStudio server. This function wraps a call to `httr::GET()` and was adapted from the DASH playbook.

You must have a valid Databricks Personal Access Token (PAT) set up in your Databricks account.

For uploading files to UC, see `uc_volume_put()`.

Usage

```
uc_volume_get(workspace, volume, token, out_file)
```

Arguments

`workspace`	String. The full URL of the Databricks workspace.
`volume`	String. The full path to the file in the Unity Catalog (e.g., "/Volumes/...").
`token`	String. Your Databricks Personal Access Token (PAT).
`out_file`	String. The local file path on the RStudio server where the file should be saved

Value

No return value. The file is saved to the specified location on the RStudio server.

Author(s)

Josh Moatt (Joshua.Moatt@defra.gov.uk)

See Also

`httr::GET()`, `uc_volume_put()`

Examples

```
## Not run:
uc_volume_get(
  workspace = "https://adb-7422054397937474.14.azuredatabricks.net",
  volume = "/Volumes/prd_dash_lab/<path-to-file>/filename.csv",
  token = "dapid4b3d********************a687f9b",
  out_file = here::here("data", "filename.csv")
)

## End(Not run)
```

`uc_volume_put` *Write file to the DASH Unity Catalog*

Description

Uploads a local file to the DASH Unity Catalog (UC) using a PUT request via the `httr2` package. This function is useful for saving files from the RStudio server to Databricks volumes.

You must have a valid Databricks Personal Access Token (PAT) configured in your Databricks account.

For downloading files from UC, see `uc_volume_get()`.

Usage

```
uc_volume_put(workspace, volume, token, file, folder)
```

Arguments

`workspace`	String. The full URL of the Databricks workspace.
`volume`	String. The path to the Databricks volume or folder where the file should be saved.
`token`	String. Your Databricks Personal Access Token (PAT).
`file`	String. The name of the file to upload, including its extension.
`folder`	String. The local path to the folder containing the file to upload (excluding the file name).

Value

No return value. The file is uploaded to the specified location in the Unity Catalog.

Author(s)

Josh Moatt (Joshua.Moatt@defra.gov.uk)

See Also

`httr2::request()`, `uc_volume_get()`

Examples

```
## Not run:
uc_volume_put(
  workspace = "https://adb-7422054397937474.14.azuredatabricks.net",
  volume = "/Volumes/prd_dash_lab/<path-to-file>",
  token = "dapid4b3d********************a687f9b",
  file = "filename.csv",
  folder = here::here("data")
)

## End(Not run)
```

`view_available_icons` *View icons*

Description

Returns a graphic in the Plots pane of the available icons in `create_defra_stats_infographic()`. If text is overlapping in the Plots pane, click 'Zoom' and resize the window to suit.

Usage

```
view_available_icons()
```

Author(s)

Farm Business Survey team (fbs.queries@defra.gov.uk)

`write_files_to_volume` *Write files to the DASH Unity Catalog*

Description

Functions to streamline writing files to DASH volumes. These functions will help avoid the http2 error which seems to be a frequent problem with `brickster`. This suite of functions are built on `brickster::db_volume_write()`. There is a general function, `dash_volume_write()` which can be used to save previously created files to the DASH platform. However, there are also more specific functions, to save data frames a specific file types (.Rds, .csv, .xlsx workbooks and rich text files like .md). These functions use the general function alongside an appropriate write function, using a temporary file to avoid saving anything locally. In order for these functions to work, you must be working on the Defra DASH platform and have set the required `brickster` environmental variables.

Usage

```
dash_volume_write(..., max_tries = 5, interval = 2)

write_xlsx_to_volume(data, path, ...)

write_rds_to_volume(data, path, ...)

write_csv_to_volume(data, path, ...)

write_text_to_volume(data, path, ...)

write_parquet_to_volume(data, path, ...)
```

Arguments

`...`	Additional arguments passed to `brickster::db_volume_write()`.
`max_tries`	Maximum number of tries to read in data. Added to deal with persistent http errors. Default is 5.
`interval`	Interval between tries to read in data. Added to deal with persistent http errors. Default is 2
`data`	(All functions except `dash_volume_write()`) Object to be saved. Usually this will be a data frame or similar. For `write_xlsx_to_volume()`, this must be a workbook object created using the `openxlsx` package.
`path`	A string containing the path on DASH to save the file to. Should be the full DASH string starting "/Volumes/..." including file name and extension.
`file`	(`dash_volume_write()` only) A string containing the local file path to the file you want to save to DASH. Should be the full file path including file name and extension.

Details

These functions are designed to handle the frequent http2 errors that occur with `brickster`. From testing, these errors are not code or file path errors, but are just minor bugs with the API. Often, if the same code is rerun, the data will be saved without issue.

There is one generic function which can be used to write existing files to DASH, covering any file type. There are also wrapper functions for specifically writing .csv, .Rds, and .xlsx files as well as rich text files like .txt, .md, .svg and .html.

All of the functions use the general `dash_volume_write()` function. This uses `brickster::db_volume_write()`, to attempt to write the data to the specified DASH volume. It uses a retry loop that catches errors with `tryCatch()`. If no error is thrown by `brickster`, the function will end as the data has been written. If an error is thrown by `brickster`, the function will wait a set number of seconds (controlled by the interval argument) before retrying. It will repeat the attempt to save the data until either the file is written successfully or the maximum number of attempts (set by max_tries) is reached - after which it throws an error.

The more specific functions will save objects from your R environment (usually data frames) to DASH directly, without the need to save them locally first. This helps avoiding saving data to the project folder, thus reducing the risk of accidentally committing the data or output to GitHub. To do this, the functions use an appropriate write function (see below) to create a temporary file of the right format (e.g. .csv). This file is then passed to the general function `dash_volume_write()` and the file is written to DASH as described above.

Reader functions used are as follows:

- **.csv** - `readr::write_csv()`
- **.xlsx** - `openxlsx::saveWorkbook()`
- **.Rds** - `base::saveRDS()`
- **.md** - `base::cat()`
- **.parquet** - `arrow::write_parquet()`

In order for these functions to work, you must be working on the Defra DASH platform and have set the required brickster environmental variables. These variables include setting a databricks Personal Access Token (PAT). For more information on how to do this, see the specific project README.

Value

Data file saved to DASH volume.

Author(s)

Josh Moatt (Joshua.Moatt@defra.gov.uk)

Tom Pearson (Thomas.Pearson@defra.gov.uk)

See Also

`brickster::db_volume_write()`, `readr::write_csv()`, `openxlsx::saveWorkbook()`, `base::saveRDS()`, `write_xlsx_to_volume()`, `write_csv_to_volume()`, `write_rds_to_volume()`, `write_parquet_to_volume()`

Examples

```
## Not run:
# write existing file
dash_volume_write(
  path = "/Volumes/prd_dash_lab/<path-to-file>/filename.docx",
  file = here::here("outputs", "filename.docx")
)

# write csv file
write_csv_to_volume(
  path = "/Volumes/prd_dash_lab/<path-to-file>/filename.csv",
  data = my_data_frame
)

# write xlsx file
write_xlsx_to_volume(
  path = "/Volumes/prd_dash_lab/<path-to-file>/filename.xlsx",
  data = my_data_frame
)

# write Rds file
write_rds_to_volume(
  path = "/Volumes/prd_dash_lab/<path-to-file>/filename.Rds",
  data = my_data_frame
)

# write markdown file
write_text_to_volume(
  path = "/Volumes/prd_dash_lab/<path-to-file>/filename.md",
  data = my_data
)

# write markdown file
write_parquet_to_volume(
  path = "/Volumes/prd_dash_lab/<path-to-file>/filename.parquet",
  data = my_data
)

## End(Not run)
```

Index