

Package ‘misc’

May 8, 2026

Title Miscellaneous Functions for Data and Geospatial Work

Version 0.0.5

Description Helpers for common data analysis tasks including missing-value summaries and filters, simple reporting and plotting utilities, 'Excel' import and export workflows, and reading geospatial formats (for example shapefiles in zip archives, file geodatabases, KMZ, and KML) via 'sf' and related packages. Also includes small project utilities such as creating directories, gitignore scaffolding, combined package loading, and optional 'lintr' setup.

Depends R (>= 4.1.0)

Imports data.table, dplyr, fs, ggplot2, glue, here, magrittr, purrr, tidyr, tools, usethis, readr, mapview, sf, rlang, jsonlite, writexl, zip

Suggests clipr, conflicted, covr, janitor, lexicon, magick, naniar, readxl, rnaturalearth, skimr, stringr, testthat, textclean, tidyselect, lintr

License MIT + file LICENSE

URL <https://github.com/kguidonimartins/misc>

BugReports <https://github.com/kguidonimartins/misc/issues>

Encoding UTF-8

RoxygenNote 7.3.2

NeedsCompilation no

Author Karlo Guidoni Martins [aut, cre] (ORCID:
<<https://orcid.org/0000-0002-8458-8467>>)

Maintainer Karlo Guidoni Martins <kguidonimartins@gmail.com>

Repository CRAN

Date/Publication 2026-04-04 09:10:07 UTC

Contents

add_gitignore	2
combine_words_ptbr	3
create_dirs	4
deduplicate_by	5
describe_data	6
filter_na	6
ipak	7
na_count	8
na_viz	8
prefer	9
quick_map	10
read_all_sheets_then_save_csv	11
read_all_xlsx_then_save_csv	12
read_gdb	12
read_geo	13
read_kmz	14
read_sf_zip	15
read_sheet_then_save_csv	15
remove_columns_based_on_NA	17
save_plot	18
save_temp_data	19
tad_view	20
trim_fig	21
view_excel	22
view_in	22
view_mapview_from_path	23
view_vd	24
view_vd_nonint	25
Index	26

add_gitignore	<i>Add a gitignore file to the project root</i>
---------------	---

Description

add_gitignore() fetch files using the API from gitignore.io. Also, add_gitignore() include tags (created by [ctags](#)) into the gitignore file.

Usage

```
add_gitignore(type = "r")
```

Arguments

type	a character vector with the language to be ignored
------	--

Value

No return value, called for side effects (creates .gitignore, or stops with an error if the file already exists).

Acknowledgment

add_gitignore() is inspired by `gitignore::gi_fetch_templates` and by some examples on the [gitignore.io wiki page](#).

Examples

```
if (interactive()) {  
  # Downloads from gitignore.io (requires network). Use combined `type` on  
  # first create, e.g. `add_gitignore(type = c("r", "python"))`.  
  add_gitignore()  
}
```

combine_words_ptbr *Combine words using ptbr rules*

Description

combine_words_ptbr() collapse words using ptbr rules. This function differ from `knitr::combine_words()` which uses oxford commas.

Usage

```
combine_words_ptbr(words, sep = NULL, last = NULL)
```

Arguments

words	a character vector with words to combine
sep	a character with the separator of the words. Default is NULL and insert ", "
last	a character vector with the last separator of the words. Default is NULL and insert " e "

Value

a character vector

Acknowledgment

combine_words_ptbr() uses `transformers` available in the excellent `{glue}` package

Examples

```
misc::ipak("glue")

# using in an ordinary text
feira <- c("banana", "maça", "pepino", "ovos")
glue("Por favor, compre: {combine_words_ptbr(feira)}")
```

create_dirs

Create directories

Description

The main purpose of `create_dirs()` is to create default directories used in data science projects. `create_dirs()` can also create custom directories.

Usage

```
create_dirs(dirs = NULL)
```

Arguments

`dirs` a character vector with the directory names. Default is `NULL` and create `data/{raw, clean, temp}`, `output/{figures, results, supp}`, and `R`

Value

No return value, called for side effects (creates directories and optional `.gitkeep` placeholder files).

Goal

There is a somewhat subjective discussion about the ideal directory structure for data science projects in general (see [here](#), [here](#), [here](#), and [here](#)). In my humble opinion, the decision should be made by the user/analyst/scientist/team. Here, I suggest a directory structure that has worked for me. In addition, the directory structure created fits perfectly with functions present in this package (for example [save_plot](#) and [save_temp_data](#)). Below is the *suggested* directory structure:

```
.
+- R          # local functions
+- data
| +- clean   # stores clean data
| +- raw     # stores raw data (read-only)
| +- temp    # stores temporary data
+- output
  +- figures # stores figures ready for publication/presentation
  +- results # stores text results and others
  +- supp    # stores supplementary material for publication/presentation
```

Acknowledgment

create_dirs() takes advantage of the functions available in the excellent `{fs}` package.

Examples

```
if (interactive()) {  
  # create a single directory  
  create_dirs("myfolder")  
  # create the default directories  
  create_dirs()  
  # see the resulting tree  
  fs::dir_tree()  
}
```

deduplicate_by

Remove duplicate rows based on specified grouping variables

Description

This function removes duplicate rows from a data frame while keeping the first occurrence of each unique combination of the specified grouping variables.

Usage

```
deduplicate_by(.data, ...)
```

Arguments

<code>.data</code>	A data frame or tibble
<code>...</code>	One or more unquoted variable names to group by

Value

A data frame with duplicate rows removed, keeping only the first occurrence for each unique combination of grouping variables

Examples

```
# Remove duplicates based on a single column  
mtcars %>% deduplicate_by(carb)  
  
# Remove duplicates based on multiple columns  
mtcars %>% deduplicate_by(carb, mpg)
```

describe_data	<i>Describe data</i>
---------------	----------------------

Description

Describe data

Usage

```
describe_data(data)
```

Arguments

data a data frame

Value

a skimr object

Examples

```
nice_data <- data.frame(c1 = c(1, NA), c2 = c(NA, NA))
nice_data %>%
  describe_data()
```

filter_na	<i>Easily filter NA values from data frames</i>
-----------	---

Description

filter_na() just wrap {dplyr} functions in a more convenient way, IMO.

Usage

```
filter_na(data, type = c("any", "all"))
```

Arguments

data a data frame or tibble

type a character vector indicating which type of NA-filtering must be done. If type = "any", filter_na() will filter any NA values present in the data frame. If type = "all", filter_na will filter only rows which all columns has NA values.

Value

a tibble object

Examples

```
nice_data <- data.frame(c1 = c(1, NA), c2 = c(NA, NA))
nice_data %>%
  filter_na("all")
nice_data %>%
  filter_na("any")
```

ipak

Load multiple CRAN and GitHub R packages

Description

Attaches packages that are already installed. Names that are not found on the library search path are reported with suggested `install.packages()` or `remotes::install_github()` calls to run yourself; this function does not install packages (CRAN policy).

Usage

```
ipak(pkg_list, force_cran = FALSE, force_github = FALSE)
```

Arguments

<code>pkg_list</code>	A character vector of package names. GitHub sources use "user/repo"; the installed package name is the repository name (see <code>basename()</code>).
<code>force_cran</code>	Logical. Ignored (retained for backwards compatibility; this function does not install or update packages).
<code>force_github</code>	Logical. Ignored (retained for backwards compatibility).

Value

A data frame with columns `pkg_name` (character), `success` (logical: whether `require()` attached the package), and `version` (character, NA when not loaded). Returned invisibly; summaries are printed via `print()` on subsets when rows exist.

Acknowledgment

`ipak()` was first developed by [Steven Worthington](#) and made publicly available [here](#). This version only loads packages and suggests install commands for missing ones.

Examples

```
pkg_list <- c("utils", "stats") # base packages – usually present
ipak(pkg_list)
```

na_count	<i>Count NA frequency in data</i>
----------	-----------------------------------

Description

na_count() is a way to display the count and frequency of NA in data. It can be slow over large datasets.

Usage

```
na_count(data, sort = TRUE)
```

Arguments

data	a data frame
sort	If TRUE, sort rows by descending na_percent. If FALSE, keep column order from the input.

Value

a long-format tibble

Acknowledgment

I learned this way of exploring data though the excellent webinar taught by Emily Robinson.

Examples

```
na_data <- data.frame(c1 = c(1, NA), c2 = c(NA, NA))
na_data %>% na_count()
```

na_viz	<i>Visualize NA frequency in data</i>
--------	---------------------------------------

Description

na_viz() create a ggplot plot showing the percentage of NA in each column

Usage

```
na_viz(data)
```

Arguments

data	a data frame
------	--------------

Value

a ggplot object

Acknowledgment

`na_viz()` is another name for the excellent `vis_miss()` of `{naniar}`

Examples

```
if (interactive()) {  
  na_data <- data.frame(c1 = c(1, NA), c2 = c(NA, NA))  
  na_data %>% na_viz()  
}
```

prefer

Defines preferred package::functions

Description

The most common conflict between `{tidyverse}` users is `dplyr::filter()` and `stats::filter()`; among `{raster}` users, the conflict is with `dplyr::select()`. `prefer()` eliminates conflicts between namespaces by forcing the use of all the functions of the chosen package, rather than looking for specific conflicts. Because of that and depending on the number of functions exported by a package, `prefer()` can be slow.

Usage

```
prefer(pkg_name, quiet = TRUE)
```

Arguments

<code>pkg_name</code>	a atomic vector with package names
<code>quiet</code>	If warnings should be displayed. Default is TRUE

Value

No return value, called for side effects (registers conflict preferences via `conflicted::conflict_prefer()`).

Acknowledgment

`prefer()` is shamelessly derived from a piece of code in [README.md](#) of the `{tidylog}`

Examples

```
# prefer `{dplyr}` functions over `{stats}`  
prefer("dplyr")
```

`quick_map`*Create maps quickly*

Description

`quick_map()` allows the creation of maps quickly using `{ggplot2}`. For this reason, the resulting map is fully editable through `{ggplot2}` layers.

Usage

```
quick_map(region = NULL, type = NULL)
```

Arguments

<code>region</code>	character string or atomic vector containing countries names ou continents. Default is <code>NULL</code> .
<code>type</code>	character string informing map type. Can be <code>"sf"</code> or <code>"ggplot"</code>

Value

a `ggplot` object

Acknowledgment

`quick_map()` depends heavily on the data available by the `{rnaturalearth}` package. In this sense, `quick_map()` uses a wide and dirty filtering of this data to create the map.

Examples

```
if (interactive()) {  
  # plot a world map  
  quick_map()  
  # plot a new world map  
  quick_map(region = "Americas", type = "sf")  
  # using ggplot  
  quick_map(region = "Americas", type = "ggplot")  
  # edit using ggplot2 layers  
  quick_map() +  
    ggplot2::theme_void() +  
    ggplot2::geom_sf(fill = "white")  
}
```

`read_all_sheets_then_save_csv`*Read and save all excel sheets and save them to a CSV file*

Description

`read_all_sheets_then_save_csv()` just loops `read_sheet_then_save_csv()` over the available excel sheets and save them in `data/temp/extracted_sheets`

Usage

```
read_all_sheets_then_save_csv(path_to_xlsx, dir_to_save = NULL)
```

Arguments

`path_to_xlsx` a character vector with path to the excel file

`dir_to_save` a character vector with the path to save the csv files. Default is NULL and save the csv files in the "data/temp/extracted_sheets" if it exists.

Value

A list (one element per sheet), each the return value of `read_sheet_then_save_csv()` for that sheet (invisibly NULL per call).

Acknowledgment

See: [read_sheet_then_save_csv](#)

Examples

```
if (interactive()) {  
  # read and into a csv  
  misc::create_dirs("ma-box")  
  xlsx_file <- system.file("xlsx-examples", "mtcars_workbook_001.xlsx", package = "misc")  
  read_all_sheets_then_save_csv(  
    path_to_xlsx = xlsx_file,  
    dir_to_save = "ma-box"  
  )  
}
```

```
read_all_xlsx_then_save_csv
```

Read all sheets from all excel files and save into CSV files

Description

Following the same principle of [read_all_sheets_then_save_csv](#) `read_all_xlsx_then_save_csv()` just loop `read_all_sheets_then_save_csv()` over all available xlsx files

Usage

```
read_all_xlsx_then_save_csv(path_to_xlsx)
```

Arguments

`path_to_xlsx` a character vector with the path to excel file

Value

A list (one element per .xlsx file found under `path_to_xlsx`), each the list returned by [read_all_sheets_then_save_csv\(\)](#) for that workbook.

Acknowledgment

See: [read_sheet_then_save_csv](#)

Examples

```
if (interactive()) {
  # read and into a csv
  xlsx_dir <- system.file("xlsx-examples", package = "misc")
  read_all_xlsx_then_save_csv(
    path_to_xlsx = xlsx_dir
  )
}
```

```
read_gdb
```

Read layers from a file geodatabase (.gdb)

Description

Read layers from a file geodatabase (.gdb)

Usage

```
read_gdb(path, layer = NULL, quiet = TRUE, ...)
```

Arguments

path	Path to a .gdb directory (the folder whose name ends in .gdb).
layer	If NULL (default), every layer reported by <code>sf::st_layers()</code> is read. If a character string, only that layer is read; it must exist in the geodatabase.
quiet	Passed to <code>sf::read_sf()</code> .
...	Additional arguments passed to <code>sf::read_sf()</code> .

Value

A tibble with columns `fpath` (path or GDAL dsn used for the layer), `file_type` (`tools::file_ext()`), `layer_name`, `geometry_type`, `nrows_aka_features`, `ncols_aka_fields`, `crs_name` (from `st_layers()`\$`crs` when available), and `data` (list-column of `sf::sf` objects). Layers are not row-bound; differing CRS are preserved per row.

Examples

```
gdb <- system.file("extdata", "misc_example.gdb", package = "misc")
if (nzchar(gdb) && dir.exists(gdb)) {
  read_gdb(gdb)
  read_gdb(gdb, layer = "OGRGeoJSON")
}
```

read_geo

Read a geospatial file or dataset (auto-detect by extension)

Description

Chooses the reader from `tools::file_ext(path)` (case-insensitive):

- .zip — `read_sf_zip()`
- .kmz — `read_kmz()`
- .kml — internal KML reader (same tibble layout; `fpath` is the .kml file)
- .gdb — `read_gdb()`
- anything else GDAL/sf can open on `path` — one row per layer from `sf::st_layers()` (e.g. .shp, .gpkg, .geojson)

Usage

```
read_geo(path, layer = NULL, quiet = TRUE, ...)
```

Arguments

path	Path to a spatial file or a .gdb directory.
layer	Passed to multi-layer GDAL readers. Ignored for .zip and .kmz.
quiet	Passed to <code>sf::read_sf()</code> .
...	Additional arguments passed to <code>sf::read_sf()</code> .

Value

A tibble as described in [read_gdb\(\)](#).

Examples

```
d <- system.file("extdata", package = "misc")
f <- function(...) file.path(d, ...)
if (file.exists(f("misc_example.zip"))) read_geo(f("misc_example.zip"))
if (file.exists(f("misc_example.kmz"))) read_geo(f("misc_example.kmz"))
if (file.exists(f("misc_example.kml"))) read_geo(f("misc_example.kml"))
if (file.exists(f("misc_example.gpkg"))) read_geo(f("misc_example.gpkg"))
if (file.exists(f("misc_example.geojson"))) read_geo(f("misc_example.geojson"))
if (file.exists(f("misc_example.shp"))) read_geo(f("misc_example.shp"))
if (dir.exists(f("misc_example.gdb"))) read_geo(f("misc_example.gdb"), layer = "OGRGeoJSON")
```

read_kmz	<i>Read a KMZ file (KML in a ZIP)</i>
----------	---------------------------------------

Description

Extracts the archive to a temporary directory and reads each KML layer with [sf::read_sf\(\)](#) after [sf::st_layers\(\)](#). Multiple KML files or multiple layers yield one row per layer; layer_name is simplified when there is only one layer in one file.

Usage

```
read_kmz(path, quiet = TRUE, ...)
```

Arguments

path	Path to a .kmz file.
quiet	Passed to sf::read_sf() .
...	Additional arguments passed to sf::read_sf() .

Value

A tibble with the same columns as [read_gdb\(\)](#). Here fpath is the path to the original .kmz (not the temporary .kml), and file_type is typically "kmz". Metadata columns still come from [sf::st_layers\(\)](#) on the extracted KML file used for reading.

Examples

```
kmz <- system.file("extdata", "misc_example.kmz", package = "misc")
if (nzchar(kmz) && file.exists(kmz)) read_kmz(kmz)
```

read_sf_zip	<i>Read shapefile(s) inside a ZIP archive via GDAL /vsizip/</i>
-------------	---

Description

Uses `zip::zip_list()` to find `.shp` members, then reads each with `sf::read_sf()` on a `/vsizip/...` path. Multiple shapefiles become one row each (list-column data), so differing CRS are not merged.

Usage

```
read_sf_zip(path, quiet = TRUE, ...)
```

Arguments

<code>path</code>	Path to a <code>.zip</code> file.
<code>quiet</code>	Passed to <code>sf::read_sf()</code> .
<code>...</code>	Additional arguments passed to <code>sf::read_sf()</code> .

Value

A tibble with `fpath` (the `/vsizip/...` dsn), `file_type`, metadata from `sf::st_layers()`, and `data` (list-column of `sf`). See `read_gdb()`.

Examples

```
z <- system.file("extdata", "misc_example.zip", package = "misc")
if (nzchar(z) && file.exists(z)) read_sf_zip(z)
```

read_sheet_then_save_csv	<i>Read an excel sheet and save it to a CSV file</i>
--------------------------	--

Description

`read_sheet_then_save_csv()` is heavily inspired in `readxl::read_excel()` (actually, this inherit almost all argument from it).

Usage

```
read_sheet_then_save_csv(
  excel_sheet,
  path_to_xlsx,
  dir_to_save = NULL,
  range = NULL,
  col_types = NULL,
  col_names = TRUE,
  na = "",
  trim_ws = TRUE,
  skip = 0,
  n_max = Inf,
  guess_max = min(1000, n_max),
  .name_repair = "unique"
)
```

Arguments

excel_sheet	a character vector with the name of the excel sheet
path_to_xlsx	a character vector with the path of the excel file
dir_to_save	a character vector with the path to save the csv file. Default is NULL and save the csv in the "data/temp" if it exists.
range	A cell range to read from. Includes typical Excel ranges like "B3:D87".
col_types	Either NULL to guess all from the spreadsheet or a character vector containing one entry per column from these options: "skip", "guess", "logical", "numeric", "date", "text" or "list". If exactly one col_type is specified, it will be recycled.
col_names	TRUE to use the first row as column names
na	Character vector of strings to interpret as missing values. By default, treats blank cells as missing data.
trim_ws	Should leading and trailing whitespace be trimmed?
skip	Minimum number of rows to skip before reading anything, be it column names or data.
n_max	Maximum number of data rows to read.
guess_max	Maximum number of data rows to use for guessing column types.
.name_repair	Handling of column names

Value

No return value, called for side effects (writes one CSV file for the requested sheet).

Acknowledgment

read_sheet_then_save_csv() is an adaptation of the awesome workflow described in an [article](#) from {readxl} package site.

Examples

```
if (interactive()) {  
  # read and into a csv  
  misc::create_dirs("ma-box")  
  xlsx_file <-  
    system.file("xlsx-examples", "mtcars_workbook_001.xlsx", package = "misc")  
  read_sheet_then_save_csv(  
    excel_sheet = "mtcars_sheet_001",  
    path_to_xlsx = xlsx_file,  
    dir_to_save = "ma-box"  
  )  
}
```

remove_columns_based_on_NA

Remove columns based on NA values

Description

Remove columns based on NA values

Usage

```
remove_columns_based_on_NA(data, threshold = 0.5)
```

Arguments

data	A data frame or tibble
threshold	The proportion of NA values allowed in a column (default: 0.5)

Value

A data frame with columns removed if they have more than the specified threshold of NA values

Examples

```
# Create sample data frame with NA values  
df <- data.frame(  
  a = c(1, 2, NA, 4, 5),  
  b = c(NA, NA, NA, 4, 5),  
  c = c(1, 2, 3, NA, 5)  
)  
  
# Remove columns with more than 50% NA values  
remove_columns_based_on_NA(df)  
  
# Use stricter threshold of 10% NA values  
remove_columns_based_on_NA(df, threshold = 0.1)
```

`save_plot`*Save a ggplot figure*

Description

`save_plot()` wraps `ggplot2::ggsave()` and offer option to remove white spaces around figures (creates a additional file in `output/figures/trim`; uses `trim_fig`)

Usage

```
save_plot(  
  object,  
  filename = NULL,  
  dir_to_save = NULL,  
  width = NA,  
  height = NA,  
  format = NULL,  
  units = NULL,  
  dpi = NULL,  
  overwrite = FALSE,  
  trim = FALSE  
)
```

Arguments

<code>object</code>	a ggplot object
<code>filename</code>	a character vector with the name of the file to save. Default is NULL and saves with the name of the object
<code>dir_to_save</code>	a character vector with the name of the directory to save
<code>width</code>	a numerical vector with the width of the figure
<code>height</code>	a numerical vector with the height of the figure
<code>format</code>	a character vector with format of the figure. Can "jpeg", "tiff", "png" (default), or "pdf"
<code>units</code>	a character vector with the units of the figure size. Can be "in", "cm" (default), or "mm"
<code>dpi</code>	a numerical vector with the resolution of the figure. Default is 300
<code>overwrite</code>	logical
<code>trim</code>	logical

Value

No return value, called for side effects (writes a graphics file via `ggplot2::ggsave()`, and optionally calls `trim_fig()`).

Acknowledgment

save_plot() is derived from `write_plot()`, available in the excellent `start` project template

Examples

```
if (interactive()) {
  library(misc)
  ipak(c("ggplot2", "dplyr"))
  create_dirs()
  p <- mtcars %>%
    ggplot() +
    aes(x = mpg, y = cyl) +
    geom_point()
  save_plot(p)
}
```

save_temp_data	<i>Save object as RDS file</i>
----------------	--------------------------------

Description

Save object as RDS file

Usage

```
save_temp_data(object, dir_to_save = NULL)
```

Arguments

object	R object
dir_to_save	a character vector with the directory name. Default is NULL and save object in the "data/temp" if it exists.

Value

No return value, called for side effects (writes an `.rds` file via `saveRDS()`).

Examples

```
if (interactive()) {
  # create and save a R object
  awesome <- "not too much!"
  misc::create_dirs("ma-box")
  save_temp_data(object = awesome, dir_to_save = "ma-box")
  # using default directories from `misc::create_dirs()`
  create_dirs()
  so_good <- "Yep!"
  save_temp_data(object = so_good)
```

```
# reading many temp data
ext <- "\\.[rRdDsS]$"
# list files
files <- list.files(
  path = "data/temp",
  pattern = ext,
  full.names = TRUE
)
# loop over files
for (i in files) {
  # read temporary file
  tmp <- readRDS(file = i)
  # remove extension from filename
  obj_name <- gsub(
    pattern = ext,
    replacement = "",
    x = basename(i)
  )
  # assign name
  assign(obj_name, tmp)
}
}
```

tad_view

Alternative data.frame viewer using tad

Description

tad_view() is an alternative to View() function when not using RStudio. Please, make sure you have **tad** installed in your system.

Usage

```
tad_view(data)
```

Arguments

data a data.frame/tibble data format.

Value

None

Examples

```
if (interactive()) {  
  library(misc)  
  mtcars %>%  
    tad_view()  
}
```

trim_fig

Remove white spaces around figures

Description

trim_fig() just remove white spaces around a figure and save it into the trim folder (maintain the original figure untouched)

Usage

```
trim_fig(figure_path, overwrite = FALSE)
```

Arguments

figure_path	a character vector with path of the figure
overwrite	logical

Value

No return value, called for side effects (writes a trimmed image file under a trim/ subdirectory via [magick::image_write\(\)](#)).

Acknowledgment

trim_fig() wraps the excellent image_trim() of [{magick}](#)

Examples

```
if (interactive()) {  
  library(misc)  
  ipak(c("ggplot2", "dplyr"))  
  create_dirs()  
  p <- mtcars %>%  
    ggplot() +  
    aes(x = mpg, y = cyl) +  
    geom_point()  
  save_plot(p)  
  trim_fig("output/figures/p.png")  
}
```

 view_excel

View data frame in Excel or other spreadsheet viewer

Description

Opens a data frame in Microsoft Excel or another spreadsheet viewer. Also copies the data to the system clipboard.

Usage

```
view_excel(data, viewer = c("libreoffice", "gnumeric", "tad", "excel"))
```

Arguments

data	A data frame to view
viewer	The spreadsheet viewer to use. One of "libreoffice", "gnumeric", "tad", or "excel" (default)

Value

Returns nothing

 view_in

Alternative data.frame viewer

Description

view_in() is an alternative to View() function when not using RStudio. To date, it works with gnumeric, libreoffice and tad.

Usage

```
view_in(data, viewer = c("libreoffice", "gnumeric", "tad"))
```

Arguments

data	a data.frame/tibble data format.
viewer	character app to open the csv file.

Value

None

Examples

```
if (interactive()) {  
  library(misc)  
  mtcars %>%  
    view_in()  
}
```

view_mapview_from_path

View spatial data from file path with optional map preview

Description

Reads a spatial data file (.shp or .gpkg) and optionally displays it in an interactive map preview. The data is also opened in VisiData for tabular viewing.

Usage

```
view_mapview_from_path(path, preview = FALSE)
```

Arguments

path	Path to the spatial data file (.shp or .gpkg)
preview	Logical. If TRUE, opens an interactive map preview in the browser. Default is FALSE.

Details

The function performs the following steps:

1. Validates that the input file exists and has the correct extension (.shp or .gpkg)
2. Creates a temporary HTML file for the map preview in ~/.local/share/mapview/
3. Reads the spatial data using sf::read_sf()
4. If preview=TRUE, creates an interactive map using mapview and opens it in the browser
5. Opens the attribute data in VisiData

Value

Returns nothing, called for side effects

`view_vd`*View data in VisiData*

Description

Opens data in VisiData through the Terminal application on macOS. If the input is an sf object, the geometry column will be dropped before viewing.

Usage

```
view_vd(data, type = "csv")
```

Arguments

<code>data</code>	A data.frame, tibble, or sf object to view
<code>type</code>	Either "csv" or "json" format for writing the temporary file. Use "json" for preserving list-columns.

Details

This function only works in interactive sessions on macOS. It creates a temporary file and opens it in VisiData through the Terminal application. The temporary filename includes a timestamp for identification.

Value

Returns the input data invisibly

Examples

```
if (interactive()) {  
  # View a data frame  
  mtcars %>% view_vd()  
  
  # View with custom title  
  mtcars %>% view_vd(title = "Car Data")  
  
  # View with list columns preserved  
  nested_df %>% view_vd(type = "json")  
}
```

view_vd_nonint	<i>View data frame in VisiData (non-interactive version)</i>
----------------	--

Description

Opens a data frame in VisiData terminal viewer, saving to a fixed location in Downloads. Similar to `view_vd()` but without interactive mode check.

Usage

```
view_vd_nonint(data, title = NULL)
```

Arguments

<code>data</code>	A data frame or sf object to view
<code>title</code>	Optional title for the viewer window (default: "misc::view_vd")

Value

Returns the input data frame unchanged

Index

add_gitignore, 2

basename(), 7

combine_words_ptbr, 3

conflicted::conflict_prefer(), 9

create_dirs, 4

deduplicate_by, 5

describe_data, 6

filter_na, 6

ggplot2::ggsave(), 18

ipak, 7

magick::image_write(), 21

na_count, 8

na_viz, 8

prefer, 9

print(), 7

quick_map, 10

read_all_sheets_then_save_csv, 11, 12

read_all_sheets_then_save_csv(), 12

read_all_xlsx_then_save_csv, 12

read_gdb, 12

read_gdb(), 13–15

read_geo, 13

read_kmz, 14

read_kmz(), 13

read_sf_zip, 15

read_sf_zip(), 13

read_sheet_then_save_csv, 11, 12, 15

read_sheet_then_save_csv(), 11

remove_columns_based_on_NA, 17

require(), 7

save_plot, 4, 18

save_temp_data, 4, 19

saveRDS(), 19

sf::read_sf(), 13–15

sf::sf, 13

sf::st_layers(), 13–15

tad_view, 20

tools::file_ext(), 13

trim_fig, 18, 21

trim_fig(), 18

view_excel, 22

view_in, 22

view_mapview_from_path, 23

view_vd, 24

view_vd_nonint, 25

zip::zip_list(), 15