

Package: misc (via r-universe)

September 24, 2026

Title Miscellaneous Functions for Data and Geospatial Work

Version 0.1.0

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, methods, sf (>= 1.0-17), rlang, jsonlite, writexl, zip

VignetteBuilder knitr

Suggests geobr, knitr, rmarkdown, clipr, conflicted, covr, htmlwidgets, httpgd, httpuv, janitor, leaflet, lexicon, magick, naniar, readxl, rnaturalearth, skimr, stringr, testthat, textclean, withr, tidyselect, lintr

License MIT + file LICENSE

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

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

Encoding UTF-8

Roxygen list(markdown = TRUE)

Config/roxygen2/version 8.0.0

Config/pak/sysreqs libabsl-dev cmake libfontconfig1-dev libfreetype6-dev libfribidi-dev libgdal-dev gdal-bin libgeos-dev git make libharfbuzz-dev libgit2-dev libicu-dev libpng-dev libuv1-dev libssl-dev libproj-dev libsqlite3-dev libudunits2-dev libx11-dev zlib1g-dev

Repository <https://kguidonimartins.r-universe.dev>

Date/Publication 2026-09-24 17:35:36 UTC

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

RemoteRef HEAD

RemoteSha bd587323fcf7a24f602b18c0e5aa0e93efcc1a25

Contents

add_gitignore	3
clean_geo	4
combine_words_ptbr	5
create_dirs	6
deduplicate_by	7
describe_data	8
filter_na	9
fix_invalid_geometries	9
intersect_filter_touch	10
intersect_mask_filter_area	12
ipak	14
na_count	15
na_viz	15
prefer	16
quick_map	17
read_all_sheets_then_save_csv	18
read_all_xlsx_then_save_csv	19
read_gdb	20
read_geo	21
read_kmz	22
read_sf_zip	23
read_sheet_then_save_csv	24
remove_columns_based_on_NA	26
remove_geom_holes	27
save_plot	28
save_temp_data	29
serve_html	30
serve_html_stop	32
serve_map	33
serve_plots	34
serve_plots_url	35
tad_view	36
trim_fig	37
view_excel	38
view_in	39
view_mapview_from_path	39
view_qgis	40
view_vd	41
view_vd_nonint	42

Index

44

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](#).

See Also

Other project-setup: [create_dirs\(\)](#)

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()  
}
```

clean_geo

*Clean a spatial file and write a normalized copy***Description**

Reads a spatial file (.zip containing a single shapefile, .shp, .gpkg, or .geojson), drops Z/M dimensions, replaces non-ASCII characters in every attribute column, reprojects the geometry to a target CRS and writes the result to a user-provided output path. The output format is determined by the extension of output and may differ from the input format (for example, a .shp can be cleaned and written as .gpkg).

Usage

```
clean_geo(path, output, crs = 4326, encoding = "ISO-8859-1", quiet = FALSE)
```

Arguments

path	Path to the input spatial file. Must be .zip (containing exactly one shapefile), .shp, .gpkg, or .geojson. Alternatively, an in-memory sf object (for example read and filtered with read_geo()) may be supplied directly; it is cleaned and written to output.
output	Path to the output file. Required. The extension determines the output format and must also be one of .zip, .shp, .gpkg, or .geojson. Existing files at output are overwritten.
crs	Target coordinate reference system passed to sf::st_transform() . Defaults to EPSG:4326 (WGS84).
encoding	Encoding string used when writing shapefile attribute tables (passed as layer_options = "ENCODING=<encoding>"). Applies only to .shp and .zip outputs. Defaults to "ISO-8859-1".
quiet	Logical. If TRUE, suppress progress messages.

Details

The input may also be an in-memory [sf](#) object, so a layer read and pre-filtered with [read_geo\(\)](#) can be written out directly without round-tripping through a file.

This function replaces a standalone batch script that cleaned shapefiles from a client geospatial portal. The non-ASCII replacement step relies on [textclean::replace_non_ascii\(\)](#); textclean lives in [Suggests](#):, so the function stops with an informative error if it is not installed.

Value

Invisibly returns the normalized output path (character).

See Also

Other geo-io: [read_gdb\(\)](#), [read_geo\(\)](#), [read_kmz\(\)](#), [read_sf_zip\(\)](#)

Examples

```
if (requireNamespace("textclean", quietly = TRUE)) {
  z <- system.file("extdata", "misc_example.zip", package = "misc")
  if (nzchar(z) && file.exists(z)) {
    out <- tempfile(fileext = ".zip")
    clean_geo(z, out)

    out_gpkg <- tempfile(fileext = ".gpkg")
    clean_geo(z, out_gpkg)
  }

  # An in-memory sf object can be passed directly as `path`.
  s <- sf::st_sf(
    id = 1L, name = "caf\u00e9",
    geometry = sf::st_sfc(sf::st_point(c(1, 2)), crs = 4326)
  )
  out_sf <- tempfile(fileext = ".gpkg")
  clean_geo(s, out_sf)
}
```

combine_words_ptbr	<i>Combine words using ptbr rules</i>
--------------------	---------------------------------------

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

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

See Also

Other project-setup: [add_gitignore\(\)](#)

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	<i>Remove duplicate rows based on specified grouping variables</i>
----------------	--

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

See Also

Other data-wrangling: [describe_data\(\)](#)

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

See Also

Other data-wrangling: [deduplicate_by\(\)](#)

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

See Also

Other missing-data: [na_count\(\)](#), [na_viz\(\)](#), [remove_columns_based_on_NA\(\)](#)

Examples

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

fix_invalid_geometries	<i>Fix invalid geometries in an sf object</i>
------------------------	---

Description

Detects invalid geometries with [sf::st_is_valid\(\)](#), reports them, and repairs the layer with [sf::st_make_valid\(\)](#).

Usage

```
fix_invalid_geometries(data)
```

Arguments

data An `sf::sf` object.

Details

Rows with invalid geometries are reported via `message()` before `sf::st_make_valid()` is applied to the whole layer. Warnings emitted by the repair itself are suppressed. Geometries that become empty after the repair are kept and flagged with a `warning()` naming the affected rows. When every geometry is already valid, data is returned unchanged and no repair is attempted.

Value

An `sf::sf` object with the same attributes and geometry column name as data, with invalid geometries repaired.

See Also

Other geo-tools: `intersect_filter_touch()`, `intersect_mask_filter_area()`, `quick_map()`, `remove_geom_holes()`, `view_mapview_from_path()`

Examples

```
crs_pl <- sf::st_crs(3857)
bowtie <- matrix(
  c(0, 0, 1e5, 0, 0, 1e5, 1e5, 1e5, 0, 0),
  ncol = 2L,
  byrow = TRUE
)
x <- sf::st_sf(
  id = "bowtie",
  geometry = sf::st_sfc(sf::st_polygon(list(bowtie)), crs = crs_pl)
)
fix_invalid_geometries(x)
```

intersect_filter_touch

Keep features that touch a mask layer

Description

Returns the rows of `x` whose geometries intersect (touch) any geometry in `y`, using `sf::st_intersects()`. No clipping is performed: geometries are returned unchanged. Useful for selecting features inside or crossing a boundary without computing actual geometric intersections.

Usage

```
intersect_filter_touch(x, y, repair = TRUE)
```

Arguments

<code>x</code>	An <code>sf::sf</code> object.
<code>y</code>	An <code>sf::sf</code> mask layer.
<code>repair</code>	If TRUE, apply <code>sf::st_make_valid()</code> to <code>x</code> and <code>y</code> first (warnings are suppressed per call). Default TRUE.

Details

A feature is kept when `lengths(st_intersects(x, y))` is greater than zero, i.e. when it shares at least one point with any feature in `y`. This is the lightest-touch selection strategy: it does not reproject, clip, or measure overlap area. Both layers must share the same CRS (or have CRS that `sf::st_intersects()` can reconcile); transform them beforehand if needed.

Value

An `sf::sf` subset of `x` containing only features that touch `y`, preserving the original geometry and attributes of `x`.

See Also

Other geo-tools: `fix_invalid_geometries()`, `intersect_mask_filter_area()`, `quick_map()`, `remove_geom_holes()`, `view_mapview_from_path()`

Examples

```
crs_pl <- sf::st_crs(3857)
ring <- matrix(
  c(0, 0, 1e6, 0, 1e6, 1e6, 0, 1e6, 0, 0),
  ncol = 2L,
  byrow = TRUE
)
y <- sf::st_sf(geometry = sf::st_sfc(sf::st_polygon(list(ring))), crs = crs_pl)
inner <- matrix(
  c(1e5, 1e5, 9e5, 1e5, 9e5, 9e5, 1e5, 9e5, 1e5, 1e5),
  ncol = 2L,
  byrow = TRUE
)
outside <- matrix(
  c(2e6, 2e6, 3e6, 2e6, 3e6, 3e6, 2e6, 3e6, 2e6, 2e6),
  ncol = 2L,
  byrow = TRUE
)
x <- sf::st_sf(
  id = c("in", "out"),
  geometry = sf::st_sfc(
    sf::st_polygon(list(inner)),
    sf::st_polygon(list(outside)),
    crs = crs_pl
  )
)
```

```
intersect_filter_touch(x, y)
```

```
intersect_mask_filter_area
```

Clip features to a mask and drop border slivers by area ratio

Description

Transforms both layers to a projected CRS, keeps features in `x` that touch the mask `y`, computes `sf::st_intersection()`, aggregates clipped area per identifier, and drops features whose clipped fraction of their original area is below `min_area_ratio`.

Usage

```
intersect_mask_filter_area(
  x,
  y,
  x_id = NULL,
  crs = NULL,
  min_area_ratio = 0.01,
  repair = TRUE,
  dissolve = TRUE
)
```

Arguments

<code>x</code>	An <code>sf::sf</code> object with POLYGON or MULTIPOLYGON geometries.
<code>y</code>	An <code>sf::sf</code> mask layer with polygon geometries.
<code>x_id</code>	Name of the column in <code>x</code> with unique identifiers. If NULL (default), a column <code>.row_id</code> is added (row order after reprojection).
<code>crs</code>	Target projected CRS for area and intersection, from <code>sf::st_crs()</code> . If NULL (default), a SIRGAS 2000 Albers (Brazil) definition in metre units is used. Pass another projected CRS with meaningful area units when working outside Brazil.
<code>min_area_ratio</code>	Numeric in <code>(0, 1]</code> : keep a feature when <code>area_clip / area_full</code> is greater than or equal to this value. Default <code>0.01</code> (about 1% of the feature area inside the mask).
<code>repair</code>	If TRUE, apply <code>sf::st_make_valid()</code> to <code>x</code> and <code>y</code> after transforming (warnings are suppressed per call).
<code>dissolve</code>	If TRUE (default), union clipped fragments so clipped has one row per kept <code>x_id</code> (see Details). FALSE returns one row per x-y fragment, as in versions prior to 0.1.0.

Details

For each feature in `x`, `area_full` is its area before clipping and `area_clip` is the sum of areas from intersecting `x` with `y`. The ratio `summary$area_ratio` is `area_clip / area_full`: the fraction of **each** `x` feature that falls inside `y` (not the fraction of `y` covered by `x`). Only **polygon** geometries are supported for `x` and `y`: points and lines are not meaningful for an area ratio. For example, `min_area_ratio = 0.5` retains a feature only when at least half of its area overlaps the mask; the default `0.01` drops only very small edge overlaps.

`y` is treated as a mask: only its geometry is used in `sf::st_intersection()`, so attributes of `y` never appear in clipped and never collide with attributes of `x`.

When `dissolve = TRUE` (the default), clipped has exactly one row per kept `x_id`: fragments produced by misaligned boundaries between `x` and `y` (for example two administrative meshes from different vintages) are unioned back into a single feature. Non-geometry attributes of `x` are assumed constant within an `x_id` and reduced with the first value per group; if an attribute actually varies within an `x_id`, a `warning()` names the offending column(s) instead of silently picking one value. `dissolve = FALSE` keeps the previous behaviour: one row per surviving `x-y` fragment.

Value

A list with clipped, an `sf::sf` object with intersection geometries that passed the threshold (one row per `x_id` when `dissolve = TRUE`, one row per fragment when `dissolve = FALSE`), and summary, a `dplyr::tibble()` with the ID column, `area_full`, `area_clip`, `area_ratio`, and logical `keep`.

See Also

Other geo-tools: `fix_invalid_geometries()`, `intersect_filter_touch()`, `quick_map()`, `remove_geom_holes()`, `view_mapview_from_path()`

Examples

```
ring <- matrix(
  c(0, 0, 1e6, 0, 1e6, 1e6, 0, 1e6, 0, 0),
  ncol = 2L,
  byrow = TRUE
)
crs_pl <- sf::st_crs(3857)
y <- sf::st_sf(geometry = sf::st_sfc(sf::st_polygon(list(ring))), crs = crs_pl)
inner <- matrix(
  c(1e5, 1e5, 9e5, 1e5, 9e5, 9e5, 1e5, 9e5, 1e5, 1e5),
  ncol = 2L,
  byrow = TRUE
)
x <- sf::st_sf(
  id = "feat_1",
  geometry = sf::st_sfc(sf::st_polygon(list(inner))), crs = crs_pl
)
out <- intersect_mask_filter_area(x, y, x_id = "id", crs = crs_pl, repair = FALSE)
nrow(out$summary)
```

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 basename()).
<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.

See Also

Other package-management: [prefer\(\)](#)

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.

See Also

Other missing-data: [filter_na\(\)](#), [na_viz\(\)](#), [remove_columns_based_on_NA\(\)](#)

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}`

See Also

Other missing-data: [filter_na\(\)](#), [na_count\(\)](#), [remove_columns_based_on_NA\(\)](#)

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

pkg_name a atomic vector with package names

quiet 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}`

See Also

Other package-management: [ipak\(\)](#)

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

region	character string or atomic vector containing countries names ou continents. Default is NULL.
type	character string informing map type. Can be "sf" or "ggplot". Default is NULL, which plots the whole world when region is also NULL and uses "sf" otherwise.

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.

See Also

Other geo-tools: [fix_invalid_geometries\(\)](#), [intersect_filter_touch\(\)](#), [intersect_mask_filter_area\(\)](#), [remove_geom_holes\(\)](#), [view_mapview_from_path\(\)](#)

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](#)

See Also

Other excel-import: [read_all_xlsx_then_save_csv\(\)](#), [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](#)

See Also

Other excel-import: [read_all_sheets_then_save_csv\(\)](#), [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, by_bbox = NULL, ...)
```

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> .
by_bbox	If NULL (default), every feature is read. Otherwise an sf, sfc, bbox or named numeric vector (xmin, ymin, xmax, ymax) whose bounding box limits the features read: only features that intersect it are returned, filtered by GDAL at read time through the <code>wkt_filter</code> argument of <code>sf::st_read()</code> . The bounding box is reprojected to each layer's CRS; if it has no CRS, its coordinates are assumed to be in the layer's CRS. If the layer has no CRS it can be reprojected to (no .prj, or an undefined LOCAL_CS), the coordinates are used as-is with a warning. Features with empty geometry are dropped. Layers without a geometry column (e.g. a shapefile made of a .dbf only, or an attribute table in a GeoPackage) cannot be filtered: they are read in full with a warning. When set, <code>nrows_aka_features</code> reports the number of features read, not the layer total. Cannot be combined with <code>wkt_filter</code> in
...	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.

See Also

Other geo-io: [clean_geo\(\)](#), [read_geo\(\)](#), [read_kmz\(\)](#), [read_sf_zip\(\)](#)

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, by_bbox = NULL, ...)
```

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 sf::read_sf() .
by_bbox	If NULL (default), every feature is read. Otherwise an sf, sfc, bbox or named numeric vector (xmin, ymin, xmax, ymax) whose bounding box limits the features read: only features that intersect it are returned, filtered by GDAL at read time through the wkt_filter argument of sf::st_read() . The bounding box is re-projected to each layer's CRS; if it has no CRS, its coordinates are assumed to be in the layer's CRS. If the layer has no CRS it can be reprojected to (no .prj, or an undefined LOCAL_CS), the coordinates are used as-is with a warning. Features with empty geometry are dropped. Layers without a geometry column (e.g. a shapefile made of a .dbf only, or an attribute table in a GeoPackage) cannot be filtered: they are read in full with a warning. When set, nrows_aka_features reports the number of features read, not the layer total. Cannot be combined with wkt_filter in
...	Additional arguments passed to sf::read_sf() .

Value

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

See Also

Other geo-io: [clean_geo\(\)](#), [read_gdb\(\)](#), [read_kmz\(\)](#), [read_sf_zip\(\)](#)

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 only the features that intersect a bounding box
nc <- system.file("shape/nc.shp", package = "sf")
bb <- sf::st_bbox(c(xmin = -80, ymin = 35, xmax = -79, ymax = 36), crs = 4326)
read_geo(nc, by_bbox = bb)
```

read_kmz

Read a KMZ file (KML in a ZIP)

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, by_bbox = NULL, ...)
```

Arguments

path	Path to a .kmz file.
quiet	Passed to sf::read_sf() .
by_bbox	If NULL (default), every feature is read. Otherwise an sf, sfc, bbox or named numeric vector (xmin, ymin, xmax, ymax) whose bounding box limits the features read: only features that intersect it are returned, filtered by GDAL at read time through the wkt_filter argument of sf::st_read() . The bounding box is reprojected to each layer's CRS; if it has no CRS, its coordinates are assumed to be in the layer's CRS. If the layer has no CRS it can be reprojected to (no .prj, or

an undefined LOCAL_CS), the coordinates are used as-is with a warning. Features with empty geometry are dropped. Layers without a geometry column (e.g. a shapefile made of a .dbf only, or an attribute table in a GeoPackage) cannot be filtered: they are read in full with a warning. When set, `nrows_aka_features` reports the number of features read, not the layer total. Cannot be combined with `wkt_filter` in

... 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.

See Also

Other geo-io: `clean_geo()`, `read_gdb()`, `read_geo()`, `read_sf_zip()`

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, by_bbox = NULL, ...)
```

Arguments

<code>path</code>	Path to a .zip file.
<code>quiet</code>	Passed to <code>sf::read_sf()</code> .
<code>by_bbox</code>	If NULL (default), every feature is read. Otherwise an sf, sfc, bbox or named numeric vector (xmin, ymin, xmax, ymax) whose bounding box limits the features read: only features that intersect it are returned, filtered by GDAL at read time through the <code>wkt_filter</code> argument of <code>sf::st_read()</code> . The bounding box is reprojected to each layer's CRS; if it has no CRS, its coordinates are assumed to be in the layer's CRS. If the layer has no CRS it can be reprojected to (no .prj, or an undefined LOCAL_CS), the coordinates are used as-is with a warning. Features

with empty geometry are dropped. Layers without a geometry column (e.g. a shapefile made of a .dbf only, or an attribute table in a GeoPackage) cannot be filtered: they are read in full with a warning. When set, `nrows_aka_features` reports the number of features read, not the layer total. Cannot be combined with `wkt_filter` in

... Additional arguments passed to `sf::read_sf()`.

Value

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

See Also

Other geo-io: `clean_geo()`, `read_gdb()`, `read_geo()`, `read_kmz()`

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

Read an excel sheet and save it to a CSV file

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.

See Also

Other excel-import: [read_all_sheets_then_save_csv\(\)](#), [read_all_xlsx_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_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

<code>data</code>	A data frame or tibble
<code>threshold</code>	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

See Also

Other missing-data: [filter_na\(\)](#), [na_count\(\)](#), [na_viz\(\)](#)

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)
```

remove_geom_holes	<i>Remove holes from polygon geometries</i>
-------------------	---

Description

Removes polygon holes from an sf, sfc, or sfg object. With a positive max_area, only holes smaller than the threshold are removed.

Usage

```
remove_geom_holes(x, max_area = 0)
```

Arguments

x	An sf, sfc, or sfg object of type POLYGON or MULTIPOLYGON.
max_area	Maximum area of holes to remove, in the units of x (or square metres for geographic coordinates). Default 0 removes all holes.

Value

An object of the same class as x, with selected holes removed.

Acknowledgment

This implementation is adapted from `ngeo::st_remove_holes()`, written by Michael Dorman and distributed under the MIT License.

Note

For a faster alternative when max_area is unnecessary, see `sfheaders::sf_remove_holes()`: <https://github.com/dcooley/sfheaders>.

References

Adapted from the StackOverflow answer by user lbusett: <https://stackoverflow.com/questions/52654701/removing-holes-from-polygons-in-r-sf>.

See Also

Other geo-tools: [fix_invalid_geometries\(\)](#), [intersect_filter_touch\(\)](#), [intersect_mask_filter_area\(\)](#), [quick_map\(\)](#), [view_mapview_from_path\(\)](#)

`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 <code>NULL</code> 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

See Also

Other save-output: `save_temp_data()`, `trim_fig()`

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()`).

See Also

Other save-output: `save_plot()`, `trim_fig()`

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)
  }
}
```

serve_html

Serve HTML widgets and reports to a remote browser

Description

Writes an HTML widget to disk and serves it over HTTP so it can be opened in a browser on another device, such as an iPad running a terminal over SSH or mosh. Handles mapview maps, any htmlwidget (leaflet, plotly, DT, gt), and HTML files that are already on disk, such as a rendered R Markdown or Quarto report.

Usage

```
serve_html(
  x,
```

```

    host = NULL,
    name = NULL,
    port = NULL,
    bind = "0.0.0.0",
    quiet = FALSE
  )

```

Arguments

x	A mapview object, an htmlwidget, an sf object (rendered with <code>mapview::mapview()</code>), or a path to an existing .html file.
host	Hostname to advertise in the URL. If NULL, resolved the same way as in <code>serve_plots()</code> .
name	Optional label used in the file name, to tell several served pages apart. Non-alphanumeric characters are replaced with <code>-</code> .
port	Port to bind. If NULL, a running session server is reused, otherwise a random free port is chosen.
bind	Address to bind the server to. Defaults to <code>"0.0.0.0"</code> so the server answers on every interface; use <code>"127.0.0.1"</code> to keep it local.
quiet	If TRUE, do not message the URL.

Details

One static server is started per R session and reused by later calls, so every URL handed out stays valid and several pages can be compared in different browser tabs. Files are served from a session directory below `tempdir()` and are removed when R exits. Call `serve_html_stop()` to shut the server down earlier.

Pages are served on a background thread, so they stay reachable while R is busy computing.

The URL is messaged on a line of its own and probed once before it is returned, for the same reasons given under Troubleshooting a 404 in `serve_plots()`.

Unlike `serve_plots()`, a static server has no access token. Page names carry a random component instead, and directory listing is disabled, so a URL cannot be guessed from the port alone. Binding to `0.0.0.0` still exposes the server to every machine that can reach this host, so prefer `bind = "127.0.0.1"` on untrusted networks.

Interactive maps load their base tiles from the internet, so the remote device needs network access beyond the tunnel to this machine.

Value

The page URL, invisibly, as a character scalar.

See Also

`serve_map()`, `serve_html_stop()`, `serve_plots()`

Other data-viewers: `serve_html_stop()`, `serve_map()`, `serve_plots()`, `serve_plots_url()`, `tad_view()`, `view_excel()`, `view_in()`, `view_qgis()`, `view_vd()`, `view_vd_nonint()`

Examples

```
if (interactive()) {  
  # a rendered report  
  serve_html("output/report.html")  
  
  # any htmlwidget (leaflet, plotly, DT, gt, ...)  
  serve_html(leaflet::leaflet())  
}
```

serve_html_stop	<i>Stop the session HTML server</i>
-----------------	-------------------------------------

Description

Shuts down the static server started by [serve_html\(\)](#). Served URLs stop working; the next [serve_html\(\)](#) call starts a fresh server.

Usage

```
serve_html_stop(quiet = FALSE)
```

Arguments

quiet	If TRUE, do not message the outcome.
-------	--------------------------------------

Value

TRUE if a server was stopped, FALSE otherwise, invisibly.

See Also

[serve_html\(\)](#)

Other data-viewers: [serve_html\(\)](#), [serve_map\(\)](#), [serve_plots\(\)](#), [serve_plots_url\(\)](#), [tad_view\(\)](#), [view_excel\(\)](#), [view_in\(\)](#), [view_qgis\(\)](#), [view_vd\(\)](#), [view_vd_nonint\(\)](#)

Examples

```
if (interactive()) {  
  serve_html_stop()  
}
```

serve_map	<i>Serve an interactive map to a remote browser</i>
-----------	---

Description

Thin wrapper around [serve_html\(\)](#) that renders `x` with `mapview::mapview()` first, so `mapview` arguments such as `zcol` or `map.types` can be passed straight through.

Usage

```
serve_map(x, ..., host = NULL, name = NULL, quiet = FALSE)
```

Arguments

<code>x</code>	An <code>sf</code> object, or anything else <code>mapview::mapview()</code> accepts. A <code>mapview</code> object is served as is.
<code>...</code>	Further arguments passed to <code>mapview::mapview()</code> , such as <code>zcol</code> or <code>map.types</code> .
<code>host</code>	Hostname to advertise in the URL. If <code>NULL</code> , resolved the same way as in serve_plots() .
<code>name</code>	Optional label used in the file name, to tell several served pages apart. Non-alphanumeric characters are replaced with <code>-</code> .
<code>quiet</code>	If <code>TRUE</code> , do not message the URL.

Value

The page URL, invisibly, as a character scalar.

See Also

[serve_html\(\)](#), [serve_html_stop\(\)](#)

Other data-viewers: [serve_html\(\)](#), [serve_html_stop\(\)](#), [serve_plots\(\)](#), [serve_plots_url\(\)](#), [tad_view\(\)](#), [view_excel\(\)](#), [view_in\(\)](#), [view_qgis\(\)](#), [view_vd\(\)](#), [view_vd_nonint\(\)](#)

Examples

```
if (interactive()) {
  pts <- sf::st_sf(
    id = 1:2,
    geometry = sf::st_sfc(
      sf::st_point(c(-49.25, -16.68)),
      sf::st_point(c(-49.30, -16.70)),
      crs = 4326
    )
  )
  serve_map(pts, zcol = "id")
}
```

serve_plots

Serve R plots over HTTP for a remote browser

Description

Starts an `httpgd::hgd()` graphics device bound to all interfaces and returns a URL that is reachable from another device, such as an iPad running a terminal over SSH or mosh.

Usage

```
serve_plots(
  host = NULL,
  port = 0,
  bind = "0.0.0.0",
  token = TRUE,
  quiet = FALSE,
  ...
)
```

Arguments

host	Hostname to advertise in the URL. If NULL, resolved from configuration and from the Tailscale node name (see Details).
port	Port to bind. The default 0 lets the operating system pick a free port, which avoids clashing with other local servers.
bind	Address to bind the server to. Defaults to "0.0.0.0" so the device answers on every interface; use "127.0.0.1" to keep it local.
token	Access token, passed to <code>httpgd::hgd()</code> . TRUE (default) generates one and embeds it in the returned URL.
quiet	If TRUE, do not message the URL.
...	Further arguments passed to <code>httpgd::hgd()</code> , such as width, height, zoom, or bg.

Details

httpgd advertises its server under the local mDNS name (for example machine.local). That name only resolves through multicast on the same physical network, so it is useless from a tablet connected over a VPN and is the usual reason a pasted URL loads nothing. `serve_plots()` replaces that hostname with one the remote device can actually resolve.

The advertised hostname is resolved in this order:

1. the host argument;
2. the MISC_PLOTS_HOST environment variable;
3. `options(misc.plots_host = "...")`;
4. the Tailscale MagicDNS name of this machine, read from `tailscale status --json`;

5. the httpgd default, with a warning that it may not resolve remotely.

The device lives as long as the R session. Call `grDevices::dev.off()` to stop it, or `serve_plots_url()` to recover the URL of a running device.

Binding to `0.0.0.0` exposes the device to every machine that can reach this host on that port. Keep `token = TRUE` unless the network is trusted.

Value

The plot URL, invisibly, as a character scalar.

Troubleshooting a 404

httpgd serves the plot viewer at `/live` and nothing else, so the URL has to be opened whole. A bare `host:port`, a trailing slash after `/live`, or a URL clipped by a terminal that wrapped it all answer 404 Not Found; a missing or stale token answers 401 Unauthorized instead. The URL is therefore messaged on a line of its own, so tapping it in a wrapped terminal does not hand the browser a fragment, and it is probed once before being returned: an HTTP error from this machine is reported as a warning rather than left for the browser to find.

Pinning port and token keeps the URL stable across sessions, which is worth doing when the remote browser bookmarks it: `serve_plots(port = 7070, token = "plots")`.

See Also

`serve_plots_url()`

Other data-viewers: `serve_html()`, `serve_html_stop()`, `serve_map()`, `serve_plots_url()`, `tad_view()`, `view_excel()`, `view_in()`, `view_qgis()`, `view_vd()`, `view_vd_nonint()`

Examples

```
if (interactive()) {
  serve_plots()
  plot(1:10)

  # pin the hostname when Tailscale is not in use
  serve_plots(host = "192.168.0.10")
}
```

`serve_plots_url`

Recover the URL of a running plot server

Description

Rebuilds the remote-reachable URL of the httpgd device started by `serve_plots()`, for when the URL scrolled out of the terminal.

Usage

```
serve_plots_url(host = NULL, quiet = FALSE)
```

Arguments

host	Hostname to advertise in the URL. If NULL, resolved from configuration and from the Tailscale node name (see Details).
quiet	If TRUE, do not message the URL.

Value

The plot URL, invisibly, as a character scalar.

See Also

[serve_plots\(\)](#)

Other data-viewers: [serve_html\(\)](#), [serve_html_stop\(\)](#), [serve_map\(\)](#), [serve_plots\(\)](#), [tad_view\(\)](#), [view_excel\(\)](#), [view_in\(\)](#), [view_qgis\(\)](#), [view_vd\(\)](#), [view_vd_nonint\(\)](#)

Examples

```
if (interactive()) {
  serve_plots()
  serve_plots_url()
}
```

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

See Also

Other data-viewers: [serve_html\(\)](#), [serve_html_stop\(\)](#), [serve_map\(\)](#), [serve_plots\(\)](#), [serve_plots_url\(\)](#), [view_excel\(\)](#), [view_in\(\)](#), [view_qgis\(\)](#), [view_vd\(\)](#), [view_vd_nonint\(\)](#)

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 untouchable)

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}](#)

See Also

Other save-output: [save_plot\(\)](#), [save_temp_data\(\)](#)

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("excel", "libreoffice", "gnumeric", "tad"))
```

Arguments

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

Value

Returns nothing

See Also

Other data-viewers: [serve_html\(\)](#), [serve_html_stop\(\)](#), [serve_map\(\)](#), [serve_plots\(\)](#), [serve_plots_url\(\)](#), [tad_view\(\)](#), [view_in\(\)](#), [view_qgis\(\)](#), [view_vd\(\)](#), [view_vd_nonint\(\)](#)

view_in	<i>Alternative data.frame viewer</i>
---------	--------------------------------------

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

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

Value

None

See Also

Other data-viewers: [serve_html\(\)](#), [serve_html_stop\(\)](#), [serve_map\(\)](#), [serve_plots\(\)](#), [serve_plots_url\(\)](#), [tad_view\(\)](#), [view_excel\(\)](#), [view_qgis\(\)](#), [view_vd\(\)](#), [view_vd_nonint\(\)](#)

Examples

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

view_mapview_from_path	<i>View spatial data from file path with optional map preview</i>
------------------------	---

Description

Reads a spatial data file (`.shp` or `.gpkg`) and optionally displays it in an interactive map preview. **macOS only:** tabular viewing uses [view_vd_nonint\(\)](#), which is not supported on Windows or Linux; on those systems the function stops with an error.

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

Requires **macOS** because the workflow always opens the attribute table with VisiData via [view_vd_nonint\(\)](#). 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

See Also

Other geo-tools: [fix_invalid_geometries\(\)](#), [intersect_filter_touch\(\)](#), [intersect_mask_filter_area\(\)](#), [quick_map\(\)](#), [remove_geom_holes\(\)](#)

view_qgis	<i>Open an sf object in QGIS</i>
-----------	----------------------------------

Description

Writes data to a temporary geospatial file and opens it in QGIS.

Usage

```
view_qgis(data, format = c("gpkg", "geojson"), layer = NULL, app = NULL)
```

Arguments

data	An object of class sf.
format	File format, either "gpkg" (default) or "geojson".
layer	Layer name for GeoPackage output. If NULL, inferred from the input expression or project name. Ignored for GeoJSON output.
app	QGIS application name or path. If NULL, resolved from configuration and standard macOS application directories.

Details

This viewer is supported only on macOS. In a non-interactive session the input is validated and returned invisibly without writing a file or opening QGIS. Interactive files are written below `~/misc` and are not removed automatically. QGIS can be configured with `MISC_QGIS_APP` or `options(misc.qgis_app = "QGIS-LTR")`.

Value

data, invisibly.

See Also

Other data-viewers: `serve_html()`, `serve_html_stop()`, `serve_map()`, `serve_plots()`, `serve_plots_url()`, `tad_view()`, `view_excel()`, `view_in()`, `view_vd()`, `view_vd_nonint()`

Examples

```
if (interactive()) {
  sf::st_sf(
    id = 1,
    geometry = sf::st_sfc(sf::st_point(c(0, 0)))
  ) %>% view_qgis()
}
```

view_vd

View data in VisiData

Description

macOS only. Does not work on Windows or Linux. Opens data in VisiData using the built-in Terminal.app by default (`terminal = "terminal"`); use `terminal = "auto"` with `MISC_VIEW_TERM` / `options(misc.view_term)` for a configurable choice, or `terminal = "iterm"` for iTerm2. If the input is an sf object, the geometry column will be dropped before viewing.

Usage

```
view_vd(data, type = "csv", terminal = c("terminal", "auto", "iterm"))
```

Arguments

data	A data.frame, tibble, or sf object to view
type	Either "csv" or "json" format for writing the temporary file. Use "json" for preserving list-columns.
terminal	Which macOS terminal to use: "terminal" (default) is the built-in Terminal.app; "iterm" forces iTerm2 (new tab if a window already exists, otherwise a new window); "auto" reads the choice from the environment variable <code>MISC_VIEW_TERM</code> and then from <code>options(misc.view_term)</code> — set either to "terminal" or "iterm". If auto finds nothing valid, Terminal.app is used.

Details

Platform: Supported only on **macOS** (Darwin). On Windows or Linux, the function stops with an error. It only performs the VisiData launch in **interactive** R sessions; in non-interactive sessions it does not open a terminal but still returns the data.

It creates a temporary file and opens it in VisiData in the selected terminal. For `terminal = "auto"`, set `MISC_VIEW_TERM` (e.g. in `~/.Renvirom`) and/or `options(misc.view_term = "iterm")` so VisiData opens in iTerm2 when you are inside tmux or other environments where a fixed default is needed. The temporary filename includes a timestamp for identification.

The VisiData CLI (`vd`) must be installed and on your PATH (VisiData is a Python package; see <https://www.visidata.org/>). If an executable named `vd` is also on your PATH, it is invoked instead as `vd <project_basename> <file>` (a local helper; not shipped with this package); `vd` is still required.

Value

Returns the input data invisibly

See Also

Other data-viewers: `serve_html()`, `serve_html_stop()`, `serve_map()`, `serve_plots()`, `serve_plots_url()`, `tad_view()`, `view_excel()`, `view_in()`, `view_qgis()`, `view_vd_nonint()`

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

macOS only. Does not work on Windows or Linux (see `view_vd()`). Opens a data frame in VisiData terminal viewer, saving to a fixed location in Downloads. Similar to `view_vd()` but without interactive mode check. Uses `vd` when on PATH, otherwise `vd` (see Details of `view_vd()`).

Usage

```
view_vd_nonint(data, title = NULL, terminal = c("terminal", "auto", "iterm"))
```

Arguments

data	A data frame or sf object to view
title	Optional title for the viewer window (default: "misc::view_vd")
terminal	Which macOS terminal to use; see view_vd() .

Details

Platform: Supported only on **macOS**. On Windows or Linux, stops with an error; see [view_vd\(\)](#) for VisiData and terminal requirements.

Value

Returns the input data frame unchanged

See Also

Other data-viewers: [serve_html\(\)](#), [serve_html_stop\(\)](#), [serve_map\(\)](#), [serve_plots\(\)](#), [serve_plots_url\(\)](#), [tad_view\(\)](#), [view_excel\(\)](#), [view_in\(\)](#), [view_qgis\(\)](#), [view_vd\(\)](#)

Index

- * **data-viewers**
 - serve_html, 30
 - serve_html_stop, 32
 - serve_map, 33
 - serve_plots, 34
 - serve_plots_url, 35
 - tad_view, 36
 - view_excel, 38
 - view_in, 39
 - view_qgis, 40
 - view_vd, 41
 - view_vd_nonint, 42
- * **data-wrangling**
 - deduplicate_by, 7
 - describe_data, 8
- * **excel-import**
 - read_all_sheets_then_save_csv, 18
 - read_all_xlsx_then_save_csv, 19
 - read_sheet_then_save_csv, 24
- * **geo-io**
 - clean_geo, 4
 - read_gdb, 20
 - read_geo, 21
 - read_kmz, 22
 - read_sf_zip, 23
- * **geo-tools**
 - fix_invalid_geometries, 9
 - intersect_filter_touch, 10
 - intersect_mask_filter_area, 12
 - quick_map, 17
 - remove_geom_holes, 27
 - view_mapview_from_path, 39
- * **missing-data**
 - filter_na, 9
 - na_count, 15
 - na_viz, 15
 - remove_columns_based_on_NA, 26
- * **package-management**
 - ipak, 14
 - prefer, 16
- * **project-setup**
 - add_gitignore, 3
 - create_dirs, 6
- * **save-output**
 - save_plot, 28
 - save_temp_data, 29
 - trim_fig, 37
- add_gitignore, 3
- add_gitignore(), 7
- basename(), 14
- clean_geo, 4
- clean_geo(), 21–24
- combine_words_ptbr, 5
- conflicted::conflict_prefer(), 16
- create_dirs, 6
- create_dirs(), 3
- deduplicate_by, 7
- deduplicate_by(), 8
- describe_data, 8
- describe_data(), 7
- dplyr::tibble(), 13
- filter_na, 9
- filter_na(), 15, 16, 26
- fix_invalid_geometries, 9
- fix_invalid_geometries(), 11, 13, 17, 27, 40
- ggplot2::ggsave(), 28
- grDevices::dev.off(), 35
- httpgd::hgd(), 34
- intersect_filter_touch, 10
- intersect_filter_touch(), 10, 13, 17, 27, 40

intersect_mask_filter_area, 12
 intersect_mask_filter_area(), 10, 11, 17, 27, 40
 ipak, 14
 ipak(), 17

 magick::image_write(), 37
 mapview::mapview(), 31, 33

 na_count, 15
 na_count(), 9, 16, 26
 na_viz, 15
 na_viz(), 9, 15, 26

 prefer, 16
 prefer(), 14
 print(), 14

 quick_map, 17
 quick_map(), 10, 11, 13, 27, 40

 read_all_sheets_then_save_csv, 18, 19
 read_all_sheets_then_save_csv(), 19, 25
 read_all_xlsx_then_save_csv, 19
 read_all_xlsx_then_save_csv(), 18, 25
 read_gdb, 20
 read_gdb(), 4, 21–24
 read_geo, 21
 read_geo(), 4, 21, 23, 24
 read_kmz, 22
 read_kmz(), 4, 21, 22, 24
 read_sf_zip, 23
 read_sf_zip(), 4, 21–23
 read_sheet_then_save_csv, 18, 19, 24
 read_sheet_then_save_csv(), 18, 19
 remove_columns_based_on_NA, 26
 remove_columns_based_on_NA(), 9, 15, 16
 remove_geom_holes, 27
 remove_geom_holes(), 10, 11, 13, 17, 40
 require(), 14

 save_plot, 6, 28
 save_plot(), 29, 37
 save_temp_data, 6, 29
 save_temp_data(), 29, 37
 saveRDS(), 29
 serve_html, 30
 serve_html(), 32, 33, 35–39, 41–43
 serve_html_stop, 32
 serve_html_stop(), 31, 33, 35–39, 41–43

 serve_map, 33
 serve_map(), 31, 32, 35–39, 41–43
 serve_plots, 34
 serve_plots(), 31–33, 35–39, 41–43
 serve_plots_url, 35
 serve_plots_url(), 31–33, 35, 37–39, 41–43

 sf, 4
 sf::read_sf(), 20–24
 sf::sf, 10–13, 20
 sf::st_crs(), 12
 sf::st_intersection(), 12, 13
 sf::st_intersects(), 10, 11
 sf::st_is_valid(), 9
 sf::st_layers(), 20–24
 sf::st_make_valid(), 9–12
 sf::st_read(), 20–23
 sf::st_transform(), 4

 tad_view, 36
 tad_view(), 31–33, 35, 36, 38, 39, 41–43
 tempdir(), 31
 textclean::replace_non_ascii(), 4
 tools::file_ext(), 20
 trim_fig, 28, 37
 trim_fig(), 28, 29

 view_excel, 38
 view_excel(), 31–33, 35–37, 39, 41–43
 view_in, 39
 view_in(), 31–33, 35–38, 41–43
 view_mapview_from_path, 39
 view_mapview_from_path(), 10, 11, 13, 17, 27
 view_qgis, 40
 view_qgis(), 31–33, 35–39, 42, 43
 view_vd, 41
 view_vd(), 31–33, 35–39, 41–43
 view_vd_nonint, 42
 view_vd_nonint(), 31–33, 35–42

 zip::zip_list(), 23