Package ‘bulkreadr’

November 16, 2023

Title The Ultimate Tool for Reading Data in Bulk

Version 1.1.0

Description Designed to simplify and streamline the process of reading and processing large volumes of data in R, this package offers a collection of functions tailored for bulk data operations. It enables users to efficiently read multiple sheets from Microsoft Excel and Google Sheets workbooks, as well as various CSV files from a directory. The data is returned as organized data frames, facilitating further analysis and manipulation. Ideal for handling extensive data sets or batch processing tasks, bulkreadr empowers users to manage data in bulk effortlessly, saving time and effort in data preparation workflows. Additionally, the package seamlessly works with labelled data from SPSS and Stata.

License MIT + file LICENSE

URL https://github.com/gbganalyst/bulkreadr

BugReports https://github.com/gbganalyst/bulkreadr/issues

Depends purrr

Imports curl, dplyr, fs, googlesheets4, haven, inspectdf, labelled, lubridate, magrittr, openxlsx, readr, readxl, sjlabelled, stats, stringr, tibble

Suggests knitr, rmarkdown, testthat (>= 3.0.0)

VignetteBuilder knitr

Config/testthat/edition 3

Encoding UTF-8

RoxygenNote 7.2.3

NeedsCompilation no

Author Ezekiel Ogundepo [aut, cre] (<https://orcid.org/0000-0003-3974-2733>),
Ernest Fokoue [ctb] (<https://orcid.org/0000-0002-0748-9166>),
Golibe Ezeechesi [ctb],
Fatimo Adebanjo [ctb],
Isaac Ajao [ctb]

Maintainer Ezekiel Ogundepo <gbganalyst@gmail.com>

Repository CRAN

Date/Publication 2023-11-16 22:13:55 UTC

R topics documented:

convert_to_date 2
fill_missing_values 3
generate_dictionary 4
inspect_na 5
look_for 6
pull_out 7
read_csv_files_from_dir 8
read_excel_files_from_dir 9
read_excel_workbook 10
read_gsheets 12
read_spss_data 13
read_stata_data 14

Index **16**

---

convert_to_date *User friendly date parsing function*

Description

`convert_to_date()` parses an input vector into POSIXct date object. It is also powerful to convert from excel date number like `42370` into date value like `2016-01-01`.

Usage

```
convert_to_date(date_num_char, tz = "UTC")
```

Arguments

`date_num_char`	A character or numeric vector of dates
`tz`	Time zone indicator. If `NULL` (default), a Date object is returned. Otherwise a POSIXct with time zone attribute set to `tz`.

Value

a vector of class Date

Examples

```
## ** heterogeneous dates **

dates <- c(
  44869, ”22.09.2022”, NA, ”02/27/92”, ”01-19-2022”,
  ”13-01-  2022”, ”2023”, ”2023-2”, 41750.2, 41751.99,
  ”11 07 2023”, ”2023-4”
)

convert_to_date(dates)
```

`fill_missing_values` *Fill missing values in a dataframe*

Description

`fill_missing_values()` is an efficient function that addresses missing values in a dataframe. It uses imputation by function, also known as column-based imputation, to fill numeric variables with the mean or median, and non-numeric variables with the mode. This approach ensures accurate and consistent replacements derived from individual columns, resulting in a complete and reliable dataset for improved analysis and decision-making.

Usage

```
fill_missing_values(df, use_mean = TRUE)
```

Arguments

`df`	The input dataframe to be processed.
`use_mean`	Logical. If `TRUE`, missing values in numeric columns will be replaced with the mean. If `FALSE`, missing values in numeric columns will be replaced with the median.

Value

A dataframe with missing values filled.

Examples

```
library(dplyr)

# Assuming 'df' is the dataframe you want to process

df <- tibble::tibble(
Sepal_Length = c(5.2, 5, 5.7, NA, 6.2, 6.7, 5.5),
Petal_Length = c(1.5, 1.4, 4.2, 1.4, NA, 5.8, 3.7),
Petal_Width = c(NA, 0.2, 1.2, 0.2, 1.3, 1.8, NA),
Species = c("setosa", NA, "versicolor", "setosa",
           NA, "virginica", "setosa")
)

# Using mean to fill missing values for numeric variables

result_df_mean <- fill_missing_values(df, use_mean = TRUE)

result_df_mean

# Using median to fill missing values for numeric variables

result_df_median <- fill_missing_values(df, use_mean = FALSE)

result_df_median

# Impute missing values (NAs) in a grouped data frame
# You can do that by using the following:

sample_iris <- tibble::tibble(
Sepal_Length = c(5.2, 5, 5.7, NA, 6.2, 6.7, 5.5),
Petal_Length = c(1.5, 1.4, 4.2, 1.4, NA, 5.8, 3.7),
Petal_Width = c(0.3, 0.2, 1.2, 0.2, 1.3, 1.8, NA),
Species = c("setosa", "setosa", "versicolor", "setosa",
           "virginica", "virginica", "setosa")
)

sample_iris %>%
group_by(Species) %>%
group_split() %>%
map_df(fill_missing_values)
```

generate_dictionary *Create a data dictionary from labelled data*

Description

generate_dictionary() creates a data dictionary from a specified data frame. This function is particularly useful for understanding and documenting the structure of your dataset, similar to data dictionaries in Stata or SPSS.

Usage

```
generate_dictionary(data)
```

Arguments

`data` a data frame or a survey object

Details

The function returns a tibble (a modern version of R's data frame) with the following columns:

- **position**: An integer vector indicating the column position in the data frame.
- **variable**: A character vector containing the names of the variables (columns).
- **description**: A character vector with a human-readable description of each variable.
- **column type**: A character vector specifying the data type (e.g., numeric, character) of each variable.
- **missing**: An integer vector indicating the count of missing values for each variable.
- **levels**: A list vector containing the levels for categorical variables, if applicable.

Value

A tibble representing the data dictionary. Each row corresponds to a variable in the original data frame, providing detailed information about the variable's characteristics.

Examples

```
# Creating a data dictionary from an SPSS file

file_path <- system.file("extdata", "Wages.sav", package = "bulkreadr")

wage_data <- read_spss_data(file = file_path)

generate_dictionary(wage_data)
```

`inspect_na` *Summarize missingness in data frame columns*

Description

`inspect_na()` summarizes the rate of missingness in each column of a data frame. For a grouped data frame, the rate of missingness is summarized separately for each group.

Usage

```
inspect_na(df)
```

Arguments

`df` A data frame

Details

The tibble returned contains the columns:

- **col_name**, a character vector containing column names of df1.
- **cnt**, an integer vector containing the number of missing values by column.
- **pcnt**, the percentage of records in each columns that is missing.

Value

A tibble summarizing the count and percentage of columnwise missingness for a data frame.

Examples

```
library(dplyr)

# dataframe summary

inspect_na(airquality)

# grouped dataframe summary

airquality %>%
group_by(Month) %>%
 inspect_na()
```

look_for *Look for keywords variable names and descriptions in labelled data*

Description

The `look_for()` function is designed to emulate the functionality of the Stata `lookfor` command in R. It provides a powerful tool for searching through large datasets, specifically targeting variable names, variable label descriptions, factor levels, and value labels. This function is handy for users working with extensive and complex datasets, enabling them to quickly and efficiently locate the variables of interest.

Usage

```
look_for(
  data,
  ...,
  labels = TRUE,
  values = TRUE,
  ignore.case = TRUE,
  details = c("basic", "none", "full")
)
```

Arguments

`data` a data frame or a survey object

`...` optional list of keywords, a character string (or several character strings), which can be formatted as a regular expression suitable for a `base::grep()` pattern, or a vector of keywords; displays all variables if not specified

`labels` whether or not to search variable labels (descriptions); `TRUE` by default

`values` whether or not to search within values (factor levels or value labels); `TRUE` by default

`ignore.case` whether or not to make the keywords case sensitive; `TRUE` by default (case is ignored during matching)

`details` add details about each variable (full details could be time consuming for big data frames, `FALSE` is equivalent to `"none"` and `TRUE` to `"full"`)

Value

A tibble data frame featuring the variable position, name and description (if it exists) in the original data frame.

Examples

```
look_for(iris)

# Look for a single keyword.

look_for(iris, "petal")

look_for(iris, "s")
```

`pull_out` *Extract or replace parts of an object*

Description

`pull_out()` is similar to `[`. It acts on vectors, matrices, arrays and lists to extract or replace parts. It is pleasant to use with the magrittr (`%>%`) and base (`|>`) operators.

Value

`pull_out()` will return an object of the same class as the input object.

Examples

```
good_choice <- letters %>%
  pull_out(c(5, 2, 1, 4))

good_choice

iris %>%
  pull_out(, 1:4) %>%
  head()
```

---

read_csv_files_from_dir

Reads all CSV files from a directory

Description

read_csv_files_from_dir reads all csv files from the "~/data" directory and returns an appended dataframe. The resulting dataframe will be in the same order as the CSV files in the directory.

Usage

```
read_csv_files_from_dir(dir_path = ".", col_types = NULL, .id = NULL)
```

Arguments

dir_path — Path to the directory containing the CSV files.

col_types — One of NULL, a cols() specification, or a string. See vignette("readr") for more details.

If NULL, all column types will be inferred from guess_max rows of the input, interspersed throughout the file. This is convenient (and fast), but not robust. If the guessed types are wrong, you'll need to increase guess_max or supply the correct types yourself.

Column specifications created by list() or cols() must contain one column specification for each column. If you only want to read a subset of the columns, use cols_only().

Alternatively, you can use a compact string representation where each character represents one column:

- c = character
- i = integer
- n = number
- d = double
- l = logical

- f = factor
- D = date
- T = date time
- t = time
- ? = guess
- _ or - = skip

By default, reading a file without a column specification will print a message showing what `readr` guessed they were. To remove this message, set `show_col_types = FALSE` or set ‘options(readr.show_col_types = FALSE).

`.id` The name of a column in which to store the file path. This is useful when reading multiple input files and there is data in the file paths, such as the data collection date. If `NULL` (the default) no extra column is created.

Value

A tibble. If there is any column type mismatch during data frames row binding, an error will occur. This is because R cannot combine columns of different types. For example, you cannot combine a column of integers with a column of characters.

See Also

`read_excel_files_from_dir()` which reads Excel workbooks data from a directory.

Examples

```
directory <- system.file("csvfolder",  package = "bulkreadr")

read_csv_files_from_dir(dir_path = directory, .id = "cut")

# Column types mismatch error ---------------------------------------
# If the `read_csv_files_from_dir()` function complains about a data type mismatch,
# then set the `col_types` argument to `"c"`.
# This will make all the column types in the resulting dataframe be characters.
```

read_excel_files_from_dir

Read Excel Workbooks data from a directory

Description

`read_excel_files_from_dir()` reads all Excel workbooks in the `"~/data"` directory and returns an appended dataframe.

Usage

```
read_excel_files_from_dir(dir_path, col_types = NULL, .id = NULL)
```

Arguments

dir_path	Path to the directory containing the xls/xlsx files.
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. The content of a cell in a skipped column is never read and that column will not appear in the data frame output. A list cell loads a column as a list of length 1 vectors, which are typed using the type guessing logic from col_types = NULL, but on a cell-by-cell basis.
.id	The name of an optional identifier column. Provide a string to create an output column that identifies each input. The column will use names if available, otherwise it will use positions.

Value

A tibble. If there is any column type mismatch during data frames row binding, an error will occur. This is because R cannot combine columns of different types. For example, you cannot combine a column of integers with a column of characters.

See Also

read_excel_workbook() which imports data from multiple sheets of an Excel workbook

Examples

```

directory <- system.file("xlsxfolder",  package = "bulkreadr")

read_excel_files_from_dir(dir_path = directory, .id = "cut")

# Column types mismatch error --------------------------------------
# If the `read_excel_files_from_dir()` function complains about a data type mismatch,
# then set the `col_types` argument to `"text"`.
# This will make all the column types in the resulting dataframe be characters.
```

read_excel_workbook *Import data from multiple sheets of an Excel workbook*

Description

read_excel_workbook() reads all the data from the sheets of an Excel workbook and return an appended dataframe.

Usage

```
read_excel_workbook(path, col_types = NULL, .id = NULL)
```

Arguments

`path` Path to the xls/xlsx file.

`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. The content of a cell in a skipped column is never read and that column will not appear in the data frame output. A list cell loads a column as a list of length 1 vectors, which are typed using the type guessing logic from `col_types = NULL`, but on a cell-by-cell basis.

`.id` The name of an optional identifier column. Provide a string to create an output column that identifies each input. The column will use names if available, otherwise it will use positions.

Value

A tibble. If there is any column type mismatch during data frames row binding, an error will occur. This is because R cannot combine columns of different types. For example, you cannot combine a column of integers with a column of characters.

See Also

`read_excel()`, which reads a Sheet of an Excel file into a data frame, and `read_gsheets()`, which imports data from multiple sheets in a Google Sheets.

Examples

```
path <- system.file("extdata", "Diamonds.xlsx", package = "bulkreadr", mustWork = TRUE)

read_excel_workbook(path = path, .id = "Year")


# Column types mismatch error --------------------------------------
# If the `read_excel_workbook()` function complains about a data type mismatch,
# then set the `col_types` argument to `"text"`.
# This will make all the column types in the resulting DataFrame be characters.
```

read_gsheets *Import data from multiple sheets in Google Sheets*

Description

The `read_gsheets()` function imports data from multiple sheets in a Google Sheets spreadsheet and appends the resulting dataframes from each sheet together to create a single dataframe. This function is a powerful tool for data analysis, as it allows you to easily combine data from multiple sheets into a single dataset.

Usage

```
read_gsheets(ss, col_types = NULL, .id = NULL)
```

Arguments

`ss` Something that identifies a Google Sheet:

- its file id as a string or `drive_id`
- a URL from which we can recover the id
- a one-row `dribble`, which is how googledrive represents Drive files
- an instance of `googlesheets4_spreadsheet`, which is what `gs4_get()` returns

Processed through `as_sheets_id()`.

`col_types` Column types. Either `NULL` to guess all from the spreadsheet or a string of readr-style shortcodes, with one character or code per column. If exactly one `col_type` is specified, it is recycled. See Column Specification for more.

`.id` The name of an optional identifier column. Provide a string to create an output column that identifies each input. The column will use names if available, otherwise it will use positions.

Value

A tibble. If there is any column type mismatch during data frames row binding, an error will occur. This is because R cannot combine columns of different types. For example, you cannot combine a column of integers with a column of characters.

See Also

`read_sheet()` which reads a Google (spread)Sheet into a data frame.

Examples

```
sheet_id <- ”1izO0mHu3L9AMySQUXGDn9GPs1n-VwGFSEoAKGhqVQh0”
```

```
read_gsheets(ss = sheet_id, .id = "sheet.name")

# Column types mismatch error ---------------------------------------
# If the `read_gsheets()` function complains about a data type mismatch,
# then set the `col_types` argument to `"c"`.
# This will make all the column types in the resulting dataframe be characters.

# For example,


sheet_id <- "1rrjKAV05POre9lDVtHtZePTa8VROf1onVO47cHnhrTU"

try(read_gsheets(ss = sheet_id)) # error, column types mismatch

read_gsheets(ss = sheet_id, col_types = "c")
```

`read_spss_data` *Read SPSS data file*

Description

`read_spss_data()` is designed to seamlessly import data from an SPSS data (`.sav` or `.zsav`) files. It converts labelled variables into factors, a crucial step that enhances the ease of data manipulation and analysis within the R programming environment.

Usage

```
read_spss_data(file, label = FALSE)
```

Arguments

`file`	The path to the SPSS data file.
`label`	Logical indicating whether to use variable labels as column names (default is FALSE).

Value

A tibble containing the data from the SPSS file.

See Also

`read_stata_data()` which reads Stata data file and converts labelled variables into factors.

Examples

```
# Read an SPSS data file without converting variable labels as column names

file_path <- system.file("extdata", "Wages.sav", package = "bulkreadr")

data <- read_spss_data(file = file_path)

data

# Read an SPSS data file and convert variable labels as column names

data <- read_spss_data(file = file_path, label = TRUE)

data
```

read_stata_data *Read Stata data file*

Description

Read Stata data file

Usage

```
read_stata_data(file, label = FALSE)
```

Arguments

`file`	The path to the Stata data file.
`label`	Logical indicating whether to use variable labels as column names (default is FALSE).

Value

A data frame containing the Stata data, with labeled variables converted to factors.

See Also

`read_spss_data()` which reads SPSS data file and converts labelled variables into factors.

Examples

```
# Read Stata data file without converting variable labels as column names

file_path <- system.file("extdata", "Wages.dta", package = "bulkreadr")

data <- read_stata_data(file = file_path)
```

```
data

# Read Stata data file and convert variable labels as column names

data <- read_stata_data(file = file_path, label = TRUE)

data
```

Index

as_sheets_id(), *12*

base::grep(), *7*

cols(), *8*
cols_only(), *8*
convert_to_date, 2

dribble, *12*
drive_id, *12*

fill_missing_values, 3

generate_dictionary, 4
gs4_get(), *12*

inspect_na, 5

list(), *8*
look_for, 6

pull_out, 7

read_csv_files_from_dir, 8
read_excel(), *11*
read_excel_files_from_dir, 9
read_excel_files_from_dir(), *9*
read_excel_workbook, 10
read_excel_workbook(), *10*
read_gsheets, 12
read_gsheets(), *11*
read_sheet(), *12*
read_spss_data, 13
read_spss_data(), *14*
read_stata_data, 14
read_stata_data(), *13*

tibble, *9–12*