Package 'unheadr'

July 22, 2025

Type Package

Title Handle Data with Messy Header Rows and Broken Values

Version 0.4.0

Depends R (>= 3.1.0)

Description Verb-like functions to work with messy data, often derived from spreadsheets or parsed PDF tables. Includes functions for unwrapping values broken up across rows, relocating embedded grouping values, and to annotate meaningful formatting in spreadsheet files.

License MIT + file LICENSE

Encoding UTF-8

LazyData true

Imports dplyr (>= 0.8.4), rlang (>= 0.2.1), forcats, stringr, tidyr, magrittr, tidyxl, readxl, tibble

RoxygenNote 7.3.1

Suggests knitr, rmarkdown, testthat (>= 2.1.0), covr

VignetteBuilder knitr

URL https://github.com/luisDVA/unheadr, https://unheadr.liomys.mx/

BugReports https://github.com/luisDVA/unheadr/issues

NeedsCompilation no

Author Luis D. Verde Arregoitia [aut, cre] (ORCID: <https://orcid.org/0000-0001-9520-6543>)

Maintainer Luis D. Verde Arregoitia <luis@liomys.mx>

Repository CRAN

Date/Publication 2024-05-13 19:10:02 UTC

Contents

`annotate_mf` *Annotate meaningful formatting*

Description

Turns cell formatting into annotations for values in the target variable.

Usage

```
annotate_mf(xlfilepath, orig, new)
```

Arguments

`xlfilepath`	Path to a single-sheet spreadsheet file (xls or xlsx).
`orig`	Target variable to annotate formatting in.
`new`	Name of new variable with cell formatting pasted as a string.

Details

Seven popular approaches for meaningful formatting (bold, colored text, italic, strikethrough, underline, double underline, and cell highlighting) are hardcoded in the function. `sheets`, `skip`, and `range` arguments for spreadsheet input are not supported. The hex8 code of the fill color used for text color and cell highlighting is also appended in the output. Ensure the data in the spreadsheet are rectangular before running; this includes blank but formatted cells beyond the data rectangle.

Value

A tibble with a new column in which the meaningful formatting is embedded as text.

Examples

```
example_spreadsheet <- system.file("extdata/dog_test.xlsx", package = "unheadr")
annotate_mf(example_spreadsheet, orig = Task, new = Task_annotated)
```

`annotate_mf_all` *Annotate meaningful formatting for all cells*

Description

Turns cell formatting into annotations for all values across all variables.

Usage

```
annotate_mf_all(xlfilepath)
```

Arguments

`xlfilepath`	Path to a single-sheet spreadsheet file (xls or xlsx).

Details

Seven popular approaches for meaningful formatting (bold, colored text, italic, strikethrough, underline, double underline, and cell highlighting) are supported in this function. `sheets`, `skip`, and `range` arguments for spreadsheet input are not supported. The hex8 code of the fill color used for text color and cell highlighting is also appended in the output. Ensure the data in the spreadsheet are rectangular before running; this includes blank but formatted cells beyond the data rectangle.

Value

A tibble with meaningful formatting embedded as text for all rows and columns.

Examples

```
example_spreadsheet <- system.file("extdata/boutiques.xlsx", package = "unheadr")
annotate_mf_all(example_spreadsheet)
```

`AOEunits` *Statistics for game units in Age of Empires II: Definitive Edition*

Description

A dataset with the numerical values that determine the behavior and performance of selected military units available in AoE2:DE (July 2020 Game Update).

Usage

```
AOEunits
```

Format

A data frame with 128 observations of 19 variables:

unit Unit name

building Building in which each unit is trained

type Unit class

age Age at which the unit becomes trainable

cost_wood Unit cost in Wood

cost_food Unit cost in Food

cost_gold Unit cost in Gold

build_time Training time in seconds

rate_of_fire Attack speed

attack_delay Retasking time

movement_speed Travel speed on land

line_of_sight Vision over the surrounding area

hit_points Unit health

min_range Minimum attacking range for ranged units

range Maximum attacking range for ranged units

damage Damage inflicted per attack

accuracy Chance that an attack will be on target

melee_armor Armor against melee attacks

pierce_armor Armor against projectiles

Source

Age of Empires II. Copyright Microsoft Corporation. This dataset was created under Microsoft's Usage rules for Digital Goods using assets from Age of Empires II, and it is not endorsed by or affiliated with Microsoft. All information shown is an interpretation of data collected in-game with no guarantee on the accuracy of any of the data presented.

AOEunits_raw — *Statistics for game units in Age of Empires II: Definitive Edition in a messy presentation*

Description

A messy version of the AOEunits dataset, meant for demonstrating data cleaning functions.

Usage

```
AOEunits_raw
```

Format

A data frame with 139 observations of 15 variables. See AOEunits for variable descriptions.

Source

Age of Empires II. Copyright Microsoft Corporation. This dataset was created under Microsoft's Usage rules for Digital Goods using assets from Age of Empires II, and it is not endorsed by or affiliated with Microsoft. All information shown is an interpretation of data collected in-game with no guarantee on the accuracy of any of the data presented.

boutiques.xlsx — *boutiques.xlsx spreadsheet*

Description

Open XML Format Spreadsheet with 1 sheet, 6 columns, and 8 rows. Toy dataset with Q1 profits for different store locations. Additional information is encoded as meaningful formatting. Bold indicates losses (negative values), colors indicate continent, and italic indicates a second location in the same city.

Details

This data is used in the example for annotate_mf_all().

dog_test.xlsx *dog_test.xlsx spreadsheet*

Description

Open XML Format Spreadsheet with 1 sheet, 2 columns, and 12 rows. Items describe various tasks or behaviors that dogs can be evaluated on, assigned into three categories which appear along with their average scores as embedded subheaders with meaningful formatting.

Details

This data is used in the example for `annotate_mf()`.

Source

Items are modified from the checklist written by Junior Watson.

References

http://www.dogtrainingbasics.com/checklist-well-behaved-dog/

mash_colnames *Make many header rows into column names*

Description

Make many header rows into column names

Usage

```
mash_colnames(
  df,
  n_name_rows,
  keep_names = TRUE,
  sliding_headers = FALSE,
  sep = "_"
)
```

Arguments

`df`	A `data.frame` or `tibble` object in which the names are broken up across the top *n* rows.
`n_name_rows`	Number of rows at the top of the data to be used to create the new variable (column) names. Must be >= 1.

keep_names — If TRUE, existing names will be included when building the new variable names. Defaults to TRUE.

sliding_headers — If TRUE, empty values in the first (topmost) header header row be filled column-wise. Defaults to FALSE. See details.

sep — Character string to separate the unified values (default is underscore).

Details

Tables are often shared with the column names broken up across the first few rows. This function takes the number of rows at the top of a table that hold the broken up names and whether or not to include the names, and mashes the values column-wise into a single string for each column. The `keep_names` argument can be helpful for tables we imported using a `skip` argument. If `keep_names` is set to `FALSE`, adjust the value of `n_name_rows` accordingly.

This function will throw a warning when possible `NA` values end up in the variable names. `sliding_headers` can be used for tables with ragged names in which not every column has a value in the very first row. In these cases attribution by adjacency is assumed, and when `sliding_headers` is set to `TRUE` the names in the topmost row are filled row-wise. This can be useful for tables reporting survey data or experimental designs in an untidy manner.

Value

The original data frame, but with new column names and without the top *n* rows that held the broken up names.

Author(s)

This function was originally contributed by Jarrett Byrnes through a GitHub issue.

Examples

```
babies <-
  data.frame(
    stringsAsFactors = FALSE,
    Baby = c(NA, NA, "Angie", "Yean", "Pierre"),
    Age = c("in", "months", "11", "9", "7"),
    Weight = c("kg", NA, "2", "3", "4"),
    Ward = c(NA, NA, "A", "B", "C")
  )
# Including the object names
mash_colnames(babies, n_name_rows = 2, keep_names = TRUE)

babies_skip <-
  data.frame(
    stringsAsFactors = FALSE,
    X1 = c("Baby", NA, NA, "Jennie", "Yean", "Pierre"),
    X2 = c("Age", "in", "months", "11", "9", "7"),
    X3 = c("Hospital", NA, NA, "A", "B", "A")
  )
#' # Discarding the automatically-generated names (X1, X2, etc...)
```

```
mash_colnames(babies_skip, n_name_rows = 3, keep_names = FALSE)

fish_experiment <-
  data.frame(
    stringsAsFactors = FALSE,
    X1 = c("Sample", NA, "Pacific", "Atlantic", "Freshwater"),
    X2 = c("Larvae", "Control", "12", "11", "10"),
    X3 = c(NA, "Low Dose", "11", "12", "8"),
    X4 = c(NA, "High Dose", "8", "7", "9"),
    X5 = c("Adult", "Control", "13", "13", "8"),
    X6 = c(NA, "Low Dose", "13", "12", "7"),
    X7 = c(NA, "High Dose", "10", "10", "9")
  )
# Ragged names
mash_colnames(fish_experiment,
  n_name_rows = 2,
  keep_names = FALSE, sliding_headers = TRUE
)
```

`primates2017` *Comparative data for 54 species of primates*

Description

A dataset with embedded subheaders.

Usage

```
primates2017
```

Format

A data frame with 69 rows and 4 variables:

scientific_name scientific names, with geographic region and taxonomic family embedded as subheaders.

common_name vernacular name

red_list_status IUCN Red List Status in January 2017

mass_kg mean body mass in kilograms

Source

Estrada, Alejandro, et al. "Impending extinction crisis of the world's primates: Why primates matter." Science Advances 3.1 (2017): e1600946. doi:10.1126/sciadv.1600946

primates2017_broken *Comparative data for 16 species of primates with some broken values*

Description

A dataset with embedded subheaders and some values (T. obscurus, T. leucocephalus and N. bengalensis) in the scientific_names variable broken up across two rows (typically done to fit the content in a table).

Usage

```
primates2017_broken
```

Format

A data frame with 19 rows and 4 variables:

scientific_name scientific names, with embedded subheaders for geographic region and taxonomic family and broken values

common_name vernacular name

red_list_status IUCN Red List Status in January 2017

mass_kg mean body mass in kilograms

Source

Estrada, Alejandro, et al. "Impending extinction crisis of the world's primates: Why primates matter." Science Advances 3.1 (2017): e1600946. doi:10.1126/sciadv.1600946

primates2017_wrapped *Comparative data for two species of primates*

Description

A dataset in which the elements for some of the values are in separate rows'

Usage

```
primates2017_wrapped
```

Format

A data frame with 9 rows and 6 variables:

scientific_name scientific names, see reference

common_name vernacular name

habitat habitat types listed in the IUCN Red List assessments

red_list_status IUCN Red List Status in January 2017

mass_kg mean body mass in kilograms

country Countries where the species is present, from IUCN Red List assessments

Source

Estrada, Alejandro, et al. "Impending extinction crisis of the world's primates: Why primates matter." Science Advances 3.1 (2017): e1600946. doi:10.1126/sciadv.1600946

---

regex_valign *Vertical character string alignment through regular expressions*

Description

Aligning strings with regex.

Usage

```
regex_valign(stringvec, regex_ai, sep_str = "")
```

Arguments

`stringvec`	A character vector with one element for each line.
`regex_ai`	A regular expression matching the position for alignment.
`sep_str`	Optional character vector that will be inserted at the positions matched by the regular expression.

Details

Written mainly for reading fixed width files, text, or tables parsed from PDFs.

Value

A character vector with one element for each line, with padding inserted at the matched positions so that elements are vertically aligned across lines.

See Also

This function is based loosely on `textutils::valign()`.

Examples

```
guests <-
  unlist(strsplit(c("6       COAHUILA        20/03/2020
7       COAHUILA             20/03/2020
18 BAJA CALIFORNIA     16/03/2020
109       CDMX      12/03/2020
1230   QUERETARO       21/03/2020"), "\n"))

# align at first uppercase word boundary , inserting a separator
regex_valign(guests, "\\b(?=[A-Z])", " - ")
# align dates at end of string
regex_valign(guests, "\\b(?=[0-9]{2}[\\/]{1}[0-9]{2}[\\/]{1}[0-9]{4}$)")
```

`squish_newlines` *Deduplicate and remove trailing line breaks*

Description

Deduplicate and remove trailing line breaks

Usage

```
squish_newlines(sepstring)
```

Arguments

`sepstring`	A character vector with new line control characters.

Details

Useful for tables with merged cells, often imported from Word or PDF files. Can be applied across multiple columns before separating into rows.

Value

A vector without trailing or multiple consecutive new line sequences.

Examples

```
vecWithNewlines <- c("dog\n\ncat\n\n\npig\n")
squish_newlines(vecWithNewlines)
```

unbreak_rows *Merge rows up*

Description

Merge rows up

Usage

```
unbreak_rows(df, regex, ogcol, sep = " ")
```

Arguments

df	A data frame with at least two contiguous rows to be merged.
regex	A regular expression to identify sets of rows to be merged, meant for the leading of the two contiguous rows.
ogcol	Variable with the text strings to match.
sep	Character string to separate the unified values (default is space).

Details

This function recodes empty strings ("") to NA for smoother pattern matching.

Value

A tibble or data frame with merged rows. Values of the lagging rows are pasted onto the values in the leading row, whitespace is squished, and the lagging row is dropped.

Examples

```
bball <-
  data.frame(
    stringsAsFactors = FALSE,
    v1 = c(
      "Player", NA, "Sleve McDichael", "Dean Wesrey",
      "Karl Dandleton"
    ),
    v2 = c("Most points", "in a game", "55", "43", "41"),
    v3 = c("Season", "(year ending)", "2001", "2000", "2010")
  )
unbreak_rows(bball, "Most", v2)
```

`unbreak_vals` *Unbreak values using regex to match the lagging half of the broken value*

Description

Unbreak values using regex to match the lagging half of the broken value

Usage

```
unbreak_vals(df, regex, ogcol, newcol, sep = " ", slice_groups)
```

Arguments

`df`	A data frame with one or more values within a variable broken up across two rows.
`regex`	Regular expression for matching the trailing (lagging) half of the broken values.
`ogcol`	Variable to unbreak.
`newcol`	Name of the new variable with the unified values.
`sep`	Character string to separate the unified values (default is space).
`slice_groups`	Deprecated. See details and Package News.

Details

This function is limited to quite specific cases, but useful when dealing with tables that contain, for example, scientific names broken across two rows. For unwrapping values, see `unwrap_cols`.

Value

A tibble with 'unbroken' values. The variable that originally contained the broken values gets dropped, and the new variable with the unified values is placed as the first column. The `slice_groups` argument is now deprecated; the extra rows and the variable with broken values will be dropped.

Examples

```
data(primates2017_broken)
# regex matches strings starting in lowercase (broken species epithets)
unbreak_vals(primates2017_broken, "^[a-z]", scientific_name, sciname_new)
```

`untangle2` *Rectangling embedded subheaders*

Description

Rectangling embedded subheaders

Usage

```
untangle2(df, regex, orig, new)
```

Arguments

`df`	A data frame with embedded subheaders.
`regex`	Regular expression to match the subheaders.
`orig`	Variable containing the extraneous subheaders.
`new`	Name of variable that will contain the group values.

Details

Special thanks to Jenny Bryan for fixing the initial tidyeval code and overall function structure.

Value

A tibble without the matched subheaders and a new variable containing the grouping data.

Examples

```
data(primates2017)
# put taxonomic family in its own variable (matches the suffix "DAE")
untangle2(primates2017, "DAE$", scientific_name, family)
# put geographic regions in their own variable (matching them all by name)
untangle2(
  primates2017, "Asia|Madagascar|Mainland Africa|Neotropics",
  scientific_name, family
)
# with magrittr pipes (re-exported in this package)
primates2017 %>%
  untangle2("DAE$", scientific_name, family) %>%
  untangle2(
    "Asia|Madagascar|Mainland Africa|Neotropics",
    scientific_name, region
  )
```

unwrap_cols *Unwrap values and clean up NAs used as padding*

Description

Unwrap values and clean up NAs used as padding

Usage

```
unwrap_cols(df, groupingVar, separator)
```

Arguments

df	A data frame with wrapped values and an inconsistent number of NA values used to as within-group padding.
groupingVar	Name of the variable describing the observational units.
separator	Character string defining the separator that will delimit the elements of the unwrapped value.

Details

This is roughly the opposite of `tidyr::separate_rows()`.

Value

A summarized tibble. Order is preserved in the grouping variable by making it a factor.

Examples

```
data(primates2017_wrapped)
# using commas to separate elements
unwrap_cols(primates2017_wrapped, scientific_name, ", ")

# separating with semicolons
df <- data.frame(
  ounits = c("A", NA, "B", "C", "D", NA),
  vals = c(1, 2, 2, 3, 1, 3)
)
unwrap_cols(df, ounits, ";")
```

Index