Package 'tidycells'

August 27, 2019

Type Package

Title Read Tabular Data from Diverse Sources and Easily Make Them Tidy

Version 0.2.1

Description Provides utilities to read, cells from complex tabular data and heuristic detection based 'structural assignment' of those cells to a columnar or tidy format. Read functionality has the ability to read structured, partially structured or unstructured tabular data from various types of documents. The 'structural assignment' functionality has both supervised and unsupervised way of assigning cells data to columnar/tidy format. Multiple disconnected blocks of tables in a single sheet are also handled appropriately. These tools are suitable for unattended conversation of messy tables into a consumable format(usable for further analysis and data wrangling).

License MIT + file LICENSE

URL https://r-rudra.github.io/tidycells/, https://github.com/r-rudra/tidycells

BugReports https://github.com/r-rudra/tidycells/issues

Depends R (>= 3.2.0)

Imports dplyr (>= 0.8.1), ggplot2, graphics, magrittr, methods, purrr (>= 0.3.2), rlang, stats, stringr (>= 1.4.0), tibble, tidyr, unpivotr (>= 0.5.1), utils

Suggests cli, covr, docxtractr, DT, knitr, miniUI, plotly, readr, readxl, rmarkdown, rstudioapi, shiny, shinytest, stringdist, tabulizer, testthat (>= 2.1.0), tidyxl, xlsx, XML

VignetteBuilder knitr

Encoding UTF-8

LazyData true

RoxygenNote 6.1.1

NeedsCompilation no

Author Indranil Gayen [aut, cre] (<https://orcid.org/0000-0003-0197-1944>)

Maintainer Indranil Gayen <nil.gayen@gmail.com>

Repository CRAN

Date/Publication 2019-08-27 06:40:03 UTC

R topics documented:

`analyze_cells` *Analyze Cells*

Description

After `Value Attribute Classification` done on a `cell_df` next task to do is analyze it's contents for data block detection, attribute orientation identification etc. The function `analyze_cells` (and also `analyse_cells`) does the same for you.

Note:

If you are not sure about what package functions actually do or how they work together, please start with `vignette("tidycells-intro")`.

Usage

```
analyze_cells(d, silent = TRUE)
```

Arguments

`d`	A `cell_df` after `Value Attribute Classification` done
`silent`	logical scalar indicating whether to raise a warning if heuristic detection fails. (Default TRUE).

Details

it returns detailed analysis of the data structure including data block detection, attribute orientation detection etc. The argument `silent` is set to `TRUE` by default, as the warning will be given whenever the `cell_analysis` is printed.

After this step one may like to do :

- `compose_cells`

If in an interactive session, following additional functions can be helpful for interactive visualizations:

- visual_data_block_inspection
- visual_orientation_modification
- visual_traceback

Value

Detailed analysis of the cell data structure. Which will be a cell_analysis class object.

See Also

compose_cells, collate_columns

Examples

```
d <- structure(c(
  "block 1", "", "C", "D", "", "block 2", "", "C",
  "D", "", "A", "1", "2", "", "", "A", "10", "20", "", "B", "3",
  "4", "", "", "B", "30", "40"
), .Dim = c(9L, 3L))
d <- as.data.frame(d)
cd <- as_cell_df(d) %>% numeric_values_classifier()

# see it
cd %>% plot(adaptive_txt_size = FALSE)
ca <- analyze_cells(cd)

# look at the plot for detected directions
plot(ca)
```

---

as_cell_df *Transform data into Cell-DF Structure*

Description

Transform an R object (mostly matrix or data.frame) into a cell_df for further processing in other tidycells functions.

Usage

```
as_cell_df(d, take_row_names = FALSE, take_col_names = FALSE)
```

Arguments

d	the data (either a matrix with column name or a data.frame)
take_row_names	consider row names as separate cells (applicable only for data with no (row, col) information). Default is FALSE.
take_col_names	consider column names as separate cells (applicable only for data with no (row, col) information). Default is FALSE.

Value

An object of class cell_df.

Note: After this, you may like to do Value Attribute Classification.

See Also

- validate_cells which is used to validate cell_df.
- as_cells from unpivotr package.

Examples

```
as_cell_df(iris)

# consider column name as cell
as_cell_df(iris, take_col_names = TRUE)

# if the data is already in a similar format it will not further transform
# which is not true for ---> unpivotr::as_cells
# check ---> unpivotr::as_cells(iris) %>% unpivotr::as_cells()
unpivotr::as_cells(iris) %>% as_cell_df()
```

---

collate_columns *Collate Columns Based on Content*

Description

After compose_cells, this function rearranges and rename attribute-columns in order to make columns properly aligned, based on the content of the columns.

Usage

```
collate_columns(composed_data, combine_threshold = 1, rest_cols = Inf,
  retain_other_cols = FALSE, retain_cell_address = FALSE)
```

Arguments

composed_data	output of compose_cells (preferably not processed)
combine_threshold	a numerical threshold (between 0-1) for content-based collation of columns. (Default 1)
rest_cols	number of rest columns (beyond combine_threshold joins these many numbers of columns to keep)
retain_other_cols	whether to keep other intermediate (and possibly not so important) columns. (Default FALSE)
retain_cell_address	whether to keep columns like (row, col, data_block). This may be required for traceback (Default FALSE)

Details

- **Dependency on *stringdist***: If you have stringdist installed, the approximate string matching will be enhanced. There may be variations in outcome if you have stringdist vs if you don't have it.
- **Possibility of randomness**: If the attribute column is containing many distinct values, then a column representative sample will be drawn. Hence it is always recommended to set.seed if reproducibility is a matter of concern.

Value

A column collated data.frame

Examples

```
d <- system.file("extdata", "marks_cells.rds", package = "tidycells", mustWork = TRUE) %>%
  readRDS()
d <- numeric_values_classifier(d)
da <- analyze_cells(d)

dc <- compose_cells(da, print_attribute_overview = TRUE)

collate_columns(dc)
```

---

compose_cells *Compose a Cell Analysis to a tidy form*

Description

After analyze_cells carried out, you may like to use this function in order to stitch the cells together as per the analyzed results, to form a meaningful structural representation (like tidy format).

Usage

```
compose_cells(ca, post_process = TRUE, attr_sep = " :: ",
  discard_raw_cols = FALSE, print_attribute_overview = FALSE,
  silent = FALSE)
```

Arguments

ca	a cell_analysis to process
post_process	logical scalar. If disabled a list will be returned without performing post-processing. (Default TRUE)
attr_sep	a character string to separate the attributes. (Default is <space>::<space>)
discard_raw_cols	logical scalar. If enabled only main processed columns will be returned. (Default FALSE)
print_attribute_overview	print the overview of the attributes (4 distinct values from each attribute of each block)
silent	whether to suppress warning message on compose failure (Default FALSE)

Value

a data.frame (as tibble) in tidy form.

Examples

```
cd <- 1:(9) %>%
  matrix(nrow = 3) %>%
  as_cell_df()
cd <- sample_based_classifier(cd, attribute_sample = "1")
cd <- cd %>% dplyr::filter(value != "1")
ca <- analyze_cells(cd)

compose_cells(ca)
```

---

read_cells *Read Cells from file*

Description

This function is designed to read cell level information (and the finally analyze, compose and collate_columns) from many file types like xls, pdf, doc etc. This is a wrapper function to functions from multiple packages. The support for a specific file is dependent on the installed packages. To see the list of supported files and potentially required packages (if any) just run `read_cells()` in the console. This function supports the file format based on content and not based on just the file extension. That means if a file is saved as pdf and then the extension is removed (or extension modified to say `.xlsx`) then also the `read_cells` will detect it as pdf and read its content.

Note :

- `read_cells` is supposed to work for any kind of data. However, if it fails in intermediate stage it will raise a warning and give results till successfully processed stage.
- The heuristic-algorithm are not well-optimized (yet) so may be slow on large files.
- If the target table has numerical values as data and text as their attribute (identifier of the data elements), straight forward method is sufficient in the majority of situations. Otherwise, you may need to utilize other functions.

A Word of Warning :

The functions used inside `read_cells` *are heuristic-algorithm based. Thus, outcomes may be unexpected. It is recommend to try* `read_cells` *on the target file. If the outcome is expected., it is fine. If not try again with* `read_cells(file_name, at_level = "compose")`. *If after that also the output is not as expected then other functions are required to be used. At that time start again with* `read_cells(file_name, at_level = "make_cells")` *and proceed to further functions.*

Usage

```
read_cells(x, at_level = c("collate", "detect_and_read", "make_cells",
  "va_classify", "analyze", "compose"), omit = NULL, simplify = TRUE,
  compose_main_cols_only = TRUE, from_level, silent = TRUE, ...)
```

Arguments

`x`	either a valid file path or a `read_cell_part`
`at_level`	till which level to process. Should be one of `detect_and_read`, `make_cells`, `va_classify`, `analyze`, `compose`, `collate`. Or simply a number (like 1 means `detect_and_read`, 5 means `compose`).
`omit`	(Optional) the file-types to omit. A character vector.
`simplify`	whether to simplify the output. (Default `TRUE`). If set to `FALSE` a `read_cell_part` will be returned.
`compose_main_cols_only`	whether to compose main columns only. (Default `TRUE`).
`from_level`	(Optional) override start level. (`read_cells` will process after `from_level`)
`silent`	if `TRUE` no message will be displayed.(Default `TRUE`)
`...`	further arguments

Details

It performs following set of actions if called with default `at_level`.

- **detect_and_read**: Detect file type based on content and attempt to read the same in a format suitable to convert as `cell_df`.
- **make_cells**: Convert the file content to `cell_df` using `as_cell_df`.
- **va_classify**: Run Value Attribute Classification using `numeric_values_classifier`.
- **analyze**: Analyze the cells using `analyze_cells`.
- **compose**: Compose the cell-analysis to a tidy form using `compose_cells`.
- **collate**: Finally, collate columns based on content using `collate_columns`.

Value

If `simplify=TRUE` then different kind of object is returned in different levels (depends on `at_level`). If `at_level="compose"` then only final tibble is returned otherwise if the output is not `NULL` an attribute will be present named `"read_cells_stage"`.

If `simplify=FALSE` then it will return a `read_cell_part` which you can process manually and continue again with `read_cells` (perhaps then `from_level` may be useful).

Examples

```
# see supported files
read_cells()

fold <- system.file("extdata", "messy", package = "tidycells", mustWork = TRUE)
# File extension is intentionally given wrong
# while filename is the actual identifier of the file type
fcsv <- list.files(fold, pattern = "^csv.", full.names = TRUE)[1]
# read the data
read_cells(fcsv)
read_cells(fcsv, simplify = FALSE)
```

---

value_attribute_classify

Value/Attribute Classifier

Description

After `as_cell_df` (entry point to `tidycells`) you may need to use this function or individual *Value/Attribute Classifier*-functions as listed below in *"see also"* - section.

Here the idea is to classify all cells into either `value`, `attribute`, `empty` which will be used by `analyze_cells` for further processing.

Usage

```
value_attribute_classify(d, classifier = basic_classifier())
```

Arguments

`d`	a Cell DF
`classifier`	a classifier

Details

In order to understand the data orientation and detect data-blocks Cell DF requires additional column named `type`. This `type` column potentially contains either `value`, `attribute`, `empty`. The `value` are given corresponding to cells with observations in it. The tag, `attribute` is for the identifier of these cells. Lastly, `empty` cells are useless cells or cells with no meaningful information.

For `classifier` following options are present:

- `basic_classifier` : naive classifier which recode `data_type`.
- `sample_based_classifier` : sample-based classifier.
- `numeric_values_classifier` : considers number like cells as values.

Each of the above are available as individual functions. Those can also be directly applied on a `cell-df`. However, it is recommended to use `value_attribute_classify` as it tests for integrity after classification.

Value

a Cell DF with Value/Attribute Classification. The underlying tibble will contain an extra column named `type`.

See Also

Individual classifier functions:

- basic_classifier
- sample_based_classifier
- numeric_values_classifier,

For interactive Value/Attribute Classification check visual_va_classify

Examples

```
iris %>%
  as_cell_df() %>%
  sample_based_classifier(value_sample = "setosa") %>%
  plot()

iris %>%
  as_cell_df() %>%
  sample_based_classifier(value_sample = "setosa") %>%
  numeric_values_classifier() %>%
  plot()

if (rlang::is_installed("tidyxl")) {
  cdn <- system.file("extdata", "RBI_HBS_Table_No_166.xlsx", package = "tidycells") %>%
    tidyxl::xlsx_cells()
  cdn <- cdn %>%
    dplyr::filter(sheet == sheet[1]) %>%
    as_cell_df()

 # all of these are same except value_attribute_classify will perform validate_cells once again
  cd1 <- sample_based_classifier(cdn, value_sample = "APR")
  cd2 <- sample_based_classifier(value_sample = "APR")(cdn)
  cd3 <- value_attribute_classify(cdn,
    classifier = sample_based_classifier(value_sample = "APR")
  )
  # see it
```

```
  plot(cd3)
}
```

Index