

Regular expressions and reshaping using data tables and the nc package

by Toby Dylan Hocking

Abstract Regular expressions are powerful tools for extracting tables from non-tabular text data. Capturing regular expressions that describe information to extract from column names can be especially useful when reshaping a data table from wide (one row with many columns) to tall (one column with many rows). We present the R package `nc`, which provides functions for data reshaping, regular expressions, and a uniform interface to three C libraries (PCRE, RE2, ICU). We describe the main features of `nc`, then provide detailed comparisons with related R packages (`stats`, `utils`, `data.table`, `tidyr`, `reshape2`, `cdata`).

Introduction

Regular expressions are powerful tools for text processing that are available in many programming languages, including R. A regular expression *pattern* defines a set of *matches* in a *subject* string. For example, the pattern `.*[.]` matches zero or more non-newline characters, followed by a period, followed by zero or more non-newline characters. It would match the subjects `Sepal.Length` and `Petal.Width`, but it would not match in the subject `Species`.

The focus of this article is patterns with capture groups, which are typically defined using parentheses. For example, the pattern `(.*)[.](.*)` results in the same matches as the pattern in the previous paragraph, and it additionally allows the user to capture and extract the substrings by group index (e.g. group 1 matches `Sepal`, group 2 matches `Length`).

Named capture groups allow extracting the a substring by name rather than by index. Using names rather than indices is useful in order to create more readable regular expressions (names document the purpose of each sub-pattern), and to create more readable R code (it is easier to understand the intent of named references than numbered references). For example, the pattern `(?<part>.*)[.](?<dimension>.*)` documents that the flower part appears before the measurement dimension; the `part` group matches `Sepal` and the `dimension` group matches `Length`.

Recently, [Hocking \(2019a\)](#) proposed a new syntax for defining named capture groups in R code. Using this new syntax, named capture groups are specified using R arguments, which results in code that is easier to read and modify than capture groups defined in string literals. For example, the pattern in the previous paragraph can be written as `part=".*", "[.]", dimension=".*"`. Sub-patterns can be grouped for clarity and/or re-used using lists, and numeric data may be extracted by specifying group-specific type conversion functions.

A main thesis of this article is that regular expressions can greatly simplify the code required to specify wide-to-tall data reshaping operations. For one such operation the input is a “wide” table with many columns, and the desired output is a “tall” table with more rows, and some of the input columns converted into a smaller number of output columns (Figure 1). We refer to the columns involved in this conversion as “data input columns” and “data output columns”; the output may include additional columns with values captured from the input column names. For example the wide iris data (`W` in Figure 1) have four data input columns: `Sepal.Length`, `Sepal.Width`, `Petal.Length`, `Petal.Width`. For some purposes (e.g. displaying a histogram of each data input column) the desired reshaping operation results in a table with a single data output column (`S` in Figure 1) and two columns captured from input names. For other purposes (e.g. scatterplot to compare `Petal` and `Sepal` sizes) the desired reshaping operation results in a table with multiple data output columns (`M1` and `M2` in Figure 1). We propose to use the new regular expression syntax of [Hocking \(2019a\)](#), e.g. `part=".*", "[.]", dimension=".*"`, to define both types of wide-to-tall data reshaping operations. In particular, we propose using a single capturing regular expression for both (1) defining the subset of data input columns to convert, and (2) capturing additional output columns from the input column names. We will show that this results in a simple, powerful, non-repetitive syntax for wide-to-tall data reshaping.

In this article our original contribution is the R package `nc` which provides a new implementation of the previously proposed named capture regex syntax of [Hocking \(2019a\)](#), in addition to several new functions that perform wide-to-tall data reshaping using regular expressions. The main new ideas are (1) using un-named capture groups in the regex string literal to provide a uniform interface to three regex C libraries, (2) integration of capture groups and `data.table` functionality ([Dowle and Srinivasan, 2019](#)), and (3) specifying wide-to-tall reshape operations with a concise syntax which results in less repetitive user code than other packages. A secondary contribution of this article is a detailed comparison of current R functions for reshaping data with regular expressions.

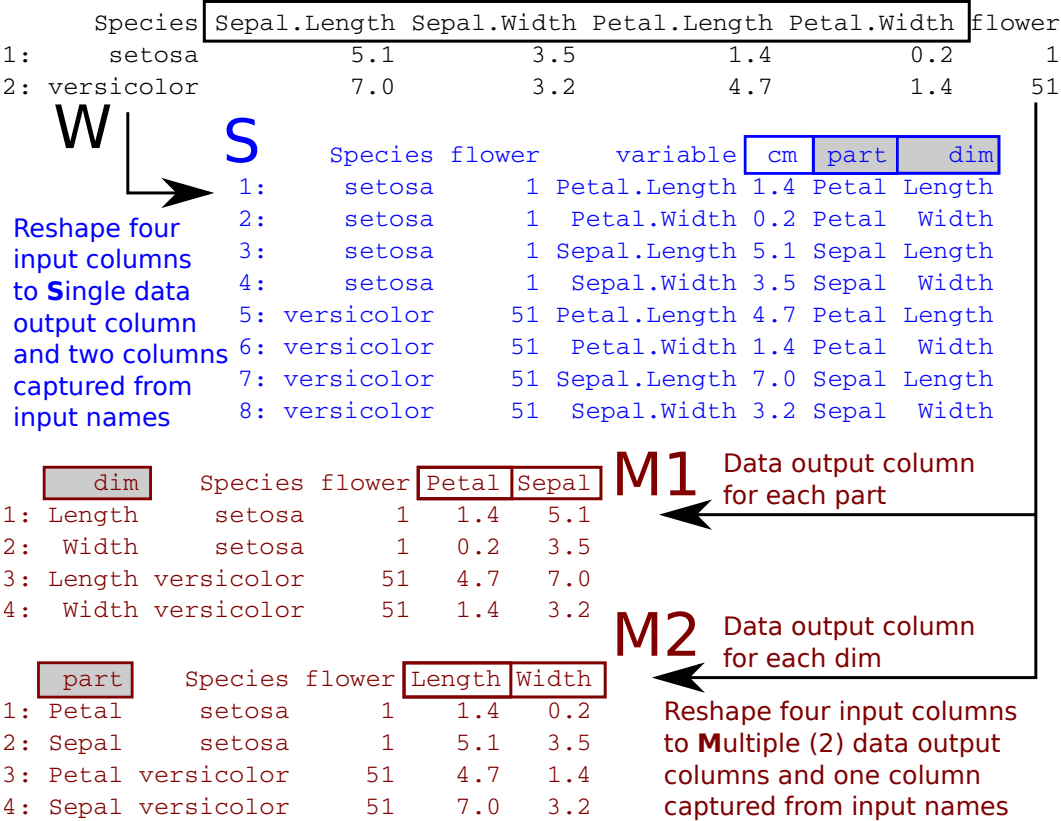


Figure 1: Wide data input columns (W, black) can be reshaped to a single data output column (S, blue) or multiple data output columns (M1, M2, red). Rectangle background indicates column type (white for data, grey for columns with values that come from text captured from input column names).

The organization of this article is as follows. The rest of this introduction provides an overview of current R packages for regular expressions and data reshaping. The second section describes the proposed functions of the `nc` package. The third section provides detailed comparisons with other R packages, in terms of syntax and computation times. The article concludes with a summary and discussion.

Related work

There are many R functions which can extract tables from non-tabular text using regular expressions. Recommended R package functions include `base::regexr` and `base::gregexpr` as well as `utils::strcapture`. CRAN packages include `namedCapture` (Hocking, 2019b), `rematch2` (Csárdi, 2017), `rex` (Ushey et al., 2017), `stringr` (Wickham, 2018), `stringi` (Gagolewski, 2018), `tidyr` (Wickham and Henry, 2018), and `re2r` (Wenfeng, 2017). Hocking (2019a) provides a detailed comparison of these packages in terms of features, syntax, and computation time.

For reshaping data from wide (one row with many columns) to tall (one column with many rows), there are several different R functions that provide similar functionality. Each function supports a different set of features (Table 1); each feature/column is explained in detail below:

- single** refers to support for converting input columns of the same type to a single output column.
- multiple** refers to support for converting input columns of possibly different types to multiple output columns; “sorted” means that conversion works correctly only if the columns are sorted in a regular order, e.g. `Sepal.Length`, `Sepal.Width`, `Petal.Length`, `Petal.Width`; “unsorted” means that conversion works correctly even if the columns are not sorted, e.g. `Sepal.Length`, `Sepal.Width`, `Petal.Width`, `Petal.Length`.
- regex** refers to support for regular expressions; “match” means a pattern is used to match the input column names; “capture” means that the capture groups in the specified pattern are used to create new output columns, which is especially useful when the names consist of several distinct variables, e.g. `Sepal.Length`; “no” means that regular expressions are not directly supported (although `base::grep` can always be used).

pkg::function	single	multiple	regex	na.rm	types	list
nc::capture_melt_multiple	no	unsorted	capture	yes	any	yes
nc::capture_melt_single	yes	no	capture	yes	any	yes
tidyr::pivot_longer	yes	unsorted	capture	yes	some	yes
stats::reshape	yes	sorted	capture	no	some	no
data.table::melt, patterns	yes	sorted	match	yes	no	yes
tidyr::gather	yes	no	no	yes	some	yes
reshape2::melt	yes	no	no	yes	no	no
cdata::rowrecs_to_blocks	yes	unsorted	no	no	no	yes
cdata::unpivot_to_blocks	yes	no	no	no	no	yes
utils::stack	yes	no	no	no	no	no

Table 1: Reshaping functions in R support various features: “single” for converting input columns into a single output column; “multiple” for converting input columns (either “sorted” in a regular order, or “unsorted” for any order) into multiple output columns of different types; “regex” for regular expressions to “match” input column names or to “capture” and create new output column names; “na.rm” for removal of missing values; “types” for converting input column names to non-character output columns; “list” for output of list columns.

na.rm refers to support for removing missing values.

types refers to support for converting captured text to numeric output columns.

list refers to support for output of list columns.

Recommended R package functions include `stats::reshape` and `utils::stack` for reshaping data from wide to tall. Of the features listed in Table 1, `utils::stack` only supports single column output, whereas `stats::reshape` supports the following features. For data with regular input column names (output column, separator, time value), regular expressions can be used to specify the separator (e.g. in `Sepal.Length`, `Sepal` is output column, `.` is separator, `Length` is time value). Multiple output columns are supported, but incorrect output may be computed if input columns are not sorted in a regular order. The time value is captured by default to an output column named `time`. Automatic type conversion is performed on time values when possible, but custom type conversion functions are not supported. There is neither support for missing value removal nor list column output.

The **tidyr** package provides two functions for reshaping data from wide to tall format: `gather` and `pivot_longer`. The older `gather` function only supports converting input columns to a single output column (not multiple). The input columns to convert may not be directly specified using regular expressions; instead R expressions such as `x:y` can be used to indicate all columns starting from `x` and ending with `y`. It does support limited type conversion; if the `convert=TRUE` argument is specified, the `utils::type.convert` function is used to convert the input column names to numeric, integer, or logical. In contrast the newer `pivot_longer` also supports multiple output columns (even if input columns are unsorted), and regular expressions for capturing new output columns from the input column names (but `grep` must be used to specify the input columns to convert). Limited type conversion is also supported in `pivot_longer`, via the `names_ptypes` argument, which should be a list with names corresponding to output columns and values corresponding to prototypes (zero-length atomic vectors, e.g. `numeric()`). Both functions support list columns and removing missing values, although different arguments are used (`na.rm` for `gather`, `values_drop_na` for `pivot_longer`).

The **reshape2** and **data.table** packages each provide a `melt` function for converting data from wide to tall (Wickham, 2007; Dowle and Srinivasan, 2019). The older **reshape2** version only supports converting input columns to a single output column, whereas the newer **data.table** version also supports multiple output columns. Regular expressions are not supported in **reshape2**, but can be used with `data.table::patterns` to match input column names to convert (although the output can be incorrect if columns are not sorted in a regular order). Neither function supports type conversion, and both functions support removing missing values from the output using the `na.rm` argument. List column output is supported in **data.table** but not **reshape2**.

The **cdata** package provides several functions for data reshaping, including `rowrecs_to_blocks` and `unpivot_to_blocks` which can convert data from wide to tall (Mount and Zumel, 2019). The simpler of the two functions is `unpivot_to_blocks`, which supports a single output column (interface similar to `reshape2::melt`/`tidyr::gather`). The user of `rowrecs_to_blocks` must provide a control table that describes how the input should be reshaped into the output. It therefore supports multiple output columns, for possibly unsorted input columns. Both functions support list column output, but other features from Table 1 are not supported (regular expressions, missing value removal, type conversion).

```

> nc::capture_melt_single

function (subject.df, ..., id.vars = NULL, variable.name = "variable",
  value.name = "value", na.rm = FALSE, verbose = getOption("datatable.verbose"))
{
  if (!is.data.frame(subject.df)) {
    stop("subject must be a data.frame")
  }
  variable <- names(subject.df)
  match.dt <- capture_first_vec(variable, ..., nomatch.error = FALSE)
  no.match <- apply(is.na(match.dt), 1, all)
  if (all(no.match)) {
    stop("no column names match regex below\n", var_args_list(...)$pattern)
  }
  names.dt <- data.table(variable, match.dt)[!no.match]
  if (is.null(id.vars)) {
    id.vars <- which(no.match)
  }
  tall.dt <- melt(data.table(subject.df), id.vars = id.vars,
    measure.vars = which(!no.match), variable.name = variable.name,
    value.name = value.name, na.rm = na.rm, variable.factor = FALSE,
    value.factor = FALSE, verbose = verbose)
  on.vec <- structure("variable", names = variable.name)
  tall.dt[names.dt, on = on.vec]
}
<bytecode: 0x2450398>
<environment: namespace:nc>
attr("ex")
function ()
{
  library(data.table)
  iris.dt <- data.table(observation = 1:nrow(iris), iris)
  (iris.tall <- nc::capture_melt_single(iris.dt, part = ".*",
    "[.]", dim = ".*"))
  (iris.part.cols <- dcast(iris.tall, observation + Species +
    dim ~ part))
  iris.part.cols[Sepal < Petal]
  (iris.dim.cols <- dcast(iris.tall, observation + Species +
    part ~ dim))
  iris.dim.cols[Length < Width]
}
<environment: namespace:nc>

```

Reproducible research statement. The source code for this article can be freely downloaded from <https://github.com/tdhock/nc-article>

Bibliography

- G. Csárdi. *rematch2: Tidy Output from Regular Expression Matching*, 2017. URL <https://CRAN.R-project.org/package=rematch2>. R package version 2.0.1. [p2]
- M. Dowle and A. Srinivasan. *data.table: Extension of 'data.frame'*, 2019. <http://r-datatable.com>. [p1, 3]
- M. Gagolewski. *R package stringi: Character string processing facilities*, 2018. URL <http://www.gagolewski.com/software/stringi/>. [p2]
- T. D. Hocking. Comparing namedcapture with other r packages for regular expressions. *R Journal*, 2019a. [p1, 2]
- T. D. Hocking. *namedCapture: Named Capture Regular Expressions*, 2019b. R package version 2019.01.14. [p2]
- J. Mount and N. Zumel. *cdata: Fluid Data Transformations*, 2019. URL <https://CRAN.R-project.org/package=cdata>. R package version 1.1.2. [p3]

- K. Ushey, J. Hester, and R. Krzyzanowski. *rex: Friendly Regular Expressions*, 2017. URL <https://CRAN.R-project.org/package=rex>. R package version 1.1.2. [p2]
- Q. Wenfeng. *re2r: RE2 Regular Expression*, 2017. URL <https://CRAN.R-project.org/package=re2r>. R package version 0.2.0. [p2]
- H. Wickham. Reshaping data with the reshape package. *Journal of Statistical Software*, 21(12):1–20, 2007. URL <http://www.jstatsoft.org/v21/i12/>. [p3]
- H. Wickham. *stringr: Simple, Consistent Wrappers for Common String Operations*, 2018. URL <https://CRAN.R-project.org/package=stringr>. R package version 1.3.1. [p2]
- H. Wickham and L. Henry. *tidyr: Easily Tidy Data with 'spread()' and 'gather()' Functions*, 2018. URL <https://CRAN.R-project.org/package=tidyr>. R package version 0.8.2. [p2]

Toby Dylan Hocking
School of Informatics, Computing, and Cyber Systems
Northern Arizona University
Flagstaff, Arizona
USA
toby.hocking@nau.edu