A new syntax for regular expressions and reshaping using data tables and the nc package

by Toby Dylan Hocking

Abstract Regular expressions are powerful tools for manipulating non-tabular text data. For many tasks (visualization, machine learning, etc), tables of numbers must be extracted from such data before processing by other R functions. We present the R package **namedCapture**, which facilitates such tasks by providing a new user-friendly syntax for defining regular expressions in R code. We begin by describing the history of regular expressions and their usage in R. We then describe the new features of the namedCapture package, and provide detailed comparisons with related R packages (**rex**, **stringr**, **stringi**, **tidyr**, **rematch2**, **re2r**).

Introduction

Today regular expression libraries are powerful and widespread tools for text processing. A regular expression *pattern* is typically a character string that defines a set of possible *matches* in some other *subject* strings. For example the pattern `o+` matches one or more lower-case o characters; it would match the last two characters in the subject `foo`, and it would not match in the subject `bar`.

The focus of this article is regular expressions with capture groups, which are used to extract subject substrings. Capture groups are typically defined using parentheses. For example, the pattern `[0-9]+` matches one or more digits (e.g. `123` but not `abc`), and the pattern `[0-9]+-[0-9]+` matches a range of integers (e.g. `9-5`). The pattern `([0-9]+)-([0-9]+)` will perform matching identically, but provides access by number/index to the strings matched by the capturing sub-patterns enclosed in parentheses (group 1 matches `9`, group 2 matches `5`). The pattern `(?P<start>[0-9]+)-(?P<end>[0-9]+)` further provides access by name to the captured sub-strings (`start` group matches `9`, `end` group matches `5`). In R named capture groups are 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).

In this article our original contribution is the R package **namedCapture** which provides several new features for named capture regular expressions. The main new ideas are (1) group-specific type conversion functions, (2) a user-friendly syntax for defining group names with R argument names, and (3) named output based on subject names and the `name` capture group.

The organization of this article is as follows. The rest of this introduction provides a brief history of regular expressions and their usage in R, then gives an overview of current R packages for regular expressions. The second section describes the proposed functions of the **namedCapture** 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.

`pkg::function`	single	multiple	regex	na.rm	types	list
`nc::capture_first_melt_multiple`	no	unsort	capture	yes	any	yes
`nc::capture_first_melt`	yes	no	capture	yes	any	yes
`tidyr::pivot_longer`	yes	unsort	capture	yes	some	yes
`stats::reshape`	yes	sort	match	no	no	no
`data.table::melt,patterns`	yes	sort	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	unsort	no	no	no	yes
`utils::stack`	yes	no	no	no	no	no

Table 1: Features of reshaping functions available in R.

Related work

```
> nc::capture_first_melt

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: 0x3bf29d8>
<environment: namespace:nc>
attr(,"ex")
function ()
{
    library(data.table)
    iris.dt <- data.table(observation = 1:nrow(iris), iris)
    (iris.tall <- nc::capture_first_melt(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

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