IE6600 Computation and Visualization for Analytics

(optional)Data Wrangling: Stringr, forcats

updated: 2021-10-27

Data Wrangling: Stringr and forcats

Strings with stringr - Prerequisites

stringr is not part of the core tidyverse.

```
library(tidyverse)
library(stringr)
```

String Basics

```r
string1 <- "This is a string"
string2 <- 'To put a "quote" inside a string, use single quotes'
string3 <- "Or put a 'quote' the other way around"
string4 <- "Here is a string\nwith a newline"
```

```r
string1 <- "This is a string"
string1
```

```
## [1] "This is a string"
```

```r
string2 <- 'To put a "quote" inside a string, use single quotes'
string2
```

```
## [1] "To put a \"quote\" inside a string, use single quotes"
```

```r
string3 <- "Or put a 'quote' the other way around"
string3
```

```
## [1] "Or put a 'quote' the other way around"
```

```r
string4 <- "Here is a string\nwith a newline"
string4
```

```
## [1] "Here is a string\nwith a newline"
```

Use cat() to putput the objects and concatenate the representations

```r
string1 <- "This is a string"
cat(string1)
```

```
## This is a string
```

```r
string2 <- 'To put a "quote" inside a string, use single quotes'
cat(string2)
```

```
## To put a "quote" inside a string, use single quotes
```

```r
string3 <- "Or put a 'quote' the other way around"
cat(string3)
```

```
## Or put a 'quote' the other way around
```

```r
string4 <- "Here is a string\nwith a newline"
cat(string4)
```

```
## Here is a string
## with a newline
```

Multiple strings

```r
c("Nancy", "John", "Tom")
```

```
## [1] "Nancy" "John"  "Tom"
```

String with stringr

Base R contains many functions to work with strings but we'll avoid them because they can be inconsistent, which makes them hard to remember. Instead we'll use functions from stringr. These have more intuitive names, and all start with str_.

```r
library(stringr)
```

String length

```r
string <- c("a", "R for data science", NA)
string
```

```
## [1] "a"                  "R for data science" NA
```

```r
#Length of the vector
length(string)
```

```
## [1] 3
```

```r
#Length of each string
str_length(string)
```

```
## [1]  1 18 NA
```

String length (cont'd)

```r
#Character length
nchar(string)
```

```
## [1]  1 18 NA
```

```r
#Byte length
nchar(string, type="bytes")
```

```
## [1]  1 18 NA
```

Combining Strings

```r
str_c("x", "y")
```

```
## [1] "xy"
```

```r
str_c("x", "y", "z")
```

```
## [1] "xyz"
```

Combining Strings with sep

```r
str_c("x", "y", sep=",")
```

```
## [1] "x,y"
```

```r
str_c(c("x", "y"), c("1", "2"), sep=" = ")
```

```
## [1] "x = 1" "y = 2"
```

Question

```r
length(str_c(c("x", "y"), c("1", "2"), sep=" = "))
```

```
## [1] 2
```

```r
# How about the following?
length(str_c(c("x", "y"), c("1", "2"), sep=" = ", collapse=", "))
```

Anwser

```r
length(str_c(c("x", "y"), c("1", "2"), sep=" = ", collapse=", "))
```

```
## [1] 1
```

Combining Strings with paste

```r
paste("x","y")
```

```
## [1] "x y"
```

```r
paste0("x","y")
```

```
## [1] "xy"
```

```r
paste("x","y", sep=",")
```

```
## [1] "x,y"
```

```r
paste(c("x", "y"), c("1", "2"), sep=" = ")
```

```
## [1] "x = 1" "y = 2"
```

Combining Strings with missing values

Like most other functions in R, missing values are contagious. If you want them to print as "NA", use str_replace_na():

```
x <- c("abc", NA)
x
```

```
## [1] "abc" NA
```

```
str_c("|-", x, "-|")
```

```
## [1] "|-abc-|" NA
```

```
str_c("|-", str_replace_na(x), "-|")
```

```
## [1] "|-abc-|" "|-NA-|"
```

Subsetting strings

You can extract parts of a string using str_sub(). As well as the string, str_sub() takes start and end arguments that give the (inclusive) position of the substring:

```
x <- c("Apple", "Banana", "Pear")
str_sub(x, 1, 3)
```

```
## [1] "App" "Ban" "Pea"
```

```
# negative numbers count backwards from end
str_sub(x, -3, -1)
```

```
## [1] "ple" "ana" "ear"
```

```
#> [1] "ple" "ana" "ear"
```

Subsetting strings (cont'd)

```
str_sub("a", 1, 5)
```

```
## [1] "a"
```

Basic Matches

```
x <- c("apple", "banana", "pear")
str_view(x, "an")
```

```
apple
banana
pear
```

Basic Matches (cont'd)

```r
str_view(x, ".a.")
```

```
apple
banana
pear
```

Anchors

By default, regular expressions will match any part of a string. It's often useful to anchor the regular expression so that it matches from the start or end of the string. You can use:

- ^ to match the start of the string.
- $ to match the end of the string.

Anchors with ^

```r
x <- c("apple", "banana", "pear")
str_view(x, "^a")
```

```
apple
banana
pear
```

Anchors with $

```r
x <- c("apple", "banana", "pear")
str_view(x, "a$")
```

```
apple
banana
pear
```

Locating positions

```r
string <- c("CAT_F", "CAT_F", "DOG_M")
str_locate(string, "_")
```

```
##      start end
## [1,]     4   4
## [2,]     4   4
## [3,]     4   4
```

Locating positions (cont'd)

```r
str_split(string, "_")
```

```
## [[1]]
## [1] "CAT" "F"
##
## [[2]]
## [1] "CAT" "F"
##
## [[3]]
## [1] "DOG" "M"
```

Detecting srings

```r
string <- c("hello", "hi", "hey", "well met")
str_detect(string, "h")
```

```
## [1]  TRUE  TRUE  TRUE FALSE
```

Splitting

Use str_split() to split a string up into pieces. For example, we could split sentences into words:

```r
sentences %>%
    head(5) %>%
    str_split(" ")
```

Splitting (cont'd)

```
sentences %>%
    head(5) %>%
    str_split(" ")
```

```
## [[1]]
## [1] "The"     "birch"   "canoe"   "slid"    "on"      "the"     "smooth"
## [8] "planks."
##
## [[2]]
## [1] "Glue"        "the"         "sheet"       "to"          "the"
## [6] "dark"        "blue"        "background."
##
## [[3]]
## [1] "It's"  "easy"  "to"    "tell"  "the"   "depth" "of"    "a"     "well."
##
## [[4]]
## [1] "These"   "days"    "a"       "chicken" "leg"     "is"      "a"
## [8] "rare"    "dish."
##
## [[5]]
## [1] "Rice"   "is"     "often"  "served" "in"     "round"  "bowls."
```

Splitting (cont'd)

If you're working with a length-1 vector, the easiest thing is to just extract the first element of the list:

```r
"a|b|c|d" %>%
  str_split("\\|") %>%
  .[[1]]
```

```
## [1] "a" "b" "c" "d"
```

Splitting (cont'd)

Otherwise, like the other stringr functions that return a list, you can use simplify = TRUE to return a matrix:

```r
sentences %>%
  head(5) %>%
  str_split(" ", simplify = TRUE)
```

Splitting (cont'd)

```
sentences %>%
head(5) %>%
str_split(" ", simplify = TRUE)
```

```
##      [,1]    [,2]    [,3]    [,4]      [,5]  [,6]    [,7]     [,8]
## [1,] "The"   "birch" "canoe" "slid"    "on"  "the"   "smooth" "planks."
## [2,] "Glue"  "the"   "sheet" "to"      "the" "dark"  "blue"   "background."
## [3,] "It's"  "easy"  "to"    "tell"    "the" "depth" "of"     "a"
## [4,] "These" "days"  "a"     "chicken" "leg" "is"    "a"      "rare"
## [5,] "Rice"  "is"    "often" "served"  "in"  "round" "bowls." ""
##      [,9]
## [1,] ""
## [2,] ""
## [3,] "well."
## [4,] "dish."
## [5,] ""
```

Splitting (cont'd)

You can also request a maximum number of pieces:

```
fields <- c("Name: Hadley", "Country: NZ", "Age: 35")
fields %>% str_split(": ", n = 2, simplify = TRUE)
```

```
##      [,1]      [,2]
## [1,] "Name"    "Hadley"
## [2,] "Country" "NZ"
## [3,] "Age"     "35"
```

Factors with forcats - Prerequisites

To work with factors, we'll use the forcats package, which provides tools for dealing with categorical variables (and it's an anagram of factors!). It provides a wide range of helpers for working with factors. forcats is not part of the core tidyverse, so we need to load it explicitly.

```
library(tidyverse)
library(forcats)
```

Creating Factors

```r
month_levels <- c(
  "Jan", "Feb", "Mar", "Apr", "May", "Jun",
  "Jul", "Aug", "Sep", "Oct", "Nov", "Dec")
factor(c("Sep", "Apr", "Jun", "Nov"), levels=month_levels)
```

Creating Factors (cont'd)

```r
factor(c("Sep", "Apr", "June", "Nov"), levels=month_levels)
ordered(c("Sep", "Apr", "Jun", "Nov"), levels=month_levels)
```

Creating Factors (cont'd)

```r
ltts <- c("D","C","A")
factor(ltts)
```

```
## [1] D C A
## Levels: A C D
```

```r
as.factor(ltts)
```

```
## [1] D C A
## Levels: A C D
```

```r
factor(ltts, levels=unique(ltts))
```

```
## [1] D C A
## Levels: D C A
```

Creating Factors (cont'd)

```
factor(ltts) %>% fct_inorder()
```

```
## [1] D C A
## Levels: D C A
```

Rearranging the levels

```r
ltts <- factor(ltts)
ltts
```

```
## [1] D C A
## Levels: A C D
```

```r
relevel(ltts,"D")
```

```
## [1] D C A
## Levels: D A C
```

Rearranging the levels (cont'd)

```r
fct_relevel(ltts,"D")
```

```
## [1] D C A
## Levels: D A C
```

```r
fct_relevel(ltts,c("D","C","A"))
```

```
## [1] D C A
## Levels: D C A
```

Recoding

```
ltts
```

```
## [1] D C A
## Levels: A C D
```

```
recode(ltts, A="X", C="Y", D="Z")
```

```
## [1] Z Y X
## Levels: X Y Z
```

Recoding (cont'd)

```r
fct_recode(ltts, X="A", Y="C", Z="D")
```

```
## [1] Z Y X
## Levels: X Y Z
```

```r
fct_recode(ltts, AA="A", AA="C")
```

```
## [1] D  AA AA
## Levels: AA D
```

```r
fct_collapse(ltts, AA=c("A", "C"))
```

```
## [1] D  AA AA
## Levels: AA D
```