

Working with text in R

Frank Edwards

What is a string?

A string is an ordered sequence of characters

- Strings are generally stored verbatim, and have no mathematical meaning (ie math operations will return errors)
- In R, these are character objects
- Generally wrapped in “ ”
- In R, can use `as.character` to convert any value to character

Let's try something

What's the difference between these commands

```
a <- 1
```

```
b <- "1"
```

- What does each command do?
- Try `str()` on each

Let's keep trying to break R

On your console, try these:

- `a<-c(1,2,3)`
- `b<-c("1", "2", "3")`
- `c<-c(1, "2", 3)`
- `d<-c(1, 2, "c")`

Let's keep trying to break R

On your console, try these:

- `a<-c(1,2,3)`
- `b<-c("1", "2", "3")`
- `c<-c(1, "2", 3)`
- `d<-c(1, 2, "c")`

What happened? Why?

Let's keep trying to break R

- `a + a`
- `a + "a"`
- `a + b`
- `"a" == "A"`
- `a == "a"`

Summary of strings in R

- R will coerce vectors to string when strings are included
- Strings are the most memory-intensive variable type (in order: logical, numeric, factor, character)
- Strings can only be compared to strings
- You should generally treat all categorical variables as strings in R (unless order matters! then use `factor()`)

Working with strings in R

The stringr package loads with tidyverse

```
library(tidyverse)
```

It has more powerful versions of base functions like:

- substr()
- grep()
- paste()
- strsplit()

Getting started

```
word <- "banana"  
str_length(word)
```

```
## [1] 6
```

```
word %>%  
  str_length()
```

```
## [1] 6
```

Pulling single characters from a string

```
word <- "banana"  
word_length <- str_length(word)  
word %>%  
  str_sub(1, 1)
```

```
## [1] "b"
```

Pulling single characters from a string

```
word <- "banana"  
word_length <- str_length(word)  
word %>%  
  str_sub(1, 1)
```

```
## [1] "b"
```

```
for (i in 1:word_length) {  
  print(str_sub(word, i, i))  
}
```

```
## [1] "b"
```

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

```
## [1] "n"
```

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

```
## [1] "n"
```

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

Pulling multiple characters

```
word <- "banana"  
word %>%  
  str_sub(1, 3)
```

```
## [1] "ban"
```

Pulling multiple characters

```
word <- "banana"  
word %>%  
  str_sub(1, 3)
```

```
## [1] "ban"
```

```
for (i in 1:word_length) {  
  print(str_sub(word, 1, i))  
}
```

```
## [1] "b"
```

```
## [1] "ba"
```

```
## [1] "ban"
```

```
## [1] "bana"
```

```
## [1] "banan"
```

```
## [1] "banana"
```

```
str_sub(word, 1, 2) <- "surprise"  
word
```

```
## [1] "surprisenana"
```

Indexing on strings, negative values

```
word
```

```
## [1] "surprisenana"
```

```
str_sub(word, -2, -1)
```

```
## [1] "na"
```

What happened here?

Some convenient functions

```
phrase <- "bananas are the tastiest"  
toupper(phrase)
```

```
## [1] "BANANAS ARE THE TASTIEST"
```

Some convenient functions

```
phrase <- "bananas are the tastiest"  
toupper(phrase)
```

```
## [1] "BANANAS ARE THE TASTIEST"
```

```
tolower(toupper(phrase))
```

```
## [1] "bananas are the tastiest"
```

Some convenient functions

```
phrase <- "bananas are the tastiest"  
toupper(phrase)
```

```
## [1] "BANANAS ARE THE TASTIEST"
```

```
tolower(toupper(phrase))
```

```
## [1] "bananas are the tastiest"
```

```
library(tools)  
toTitleCase(phrase)
```

```
## [1] "Bananas are the Tastiest"
```

Some convenient functions

```
phrase <- "bananas are the tastiest"  
toupper(phrase)
```

```
## [1] "BANANAS ARE THE TASTIEST"
```

```
tolower(toupper(phrase))
```

```
## [1] "bananas are the tastiest"
```

```
library(tools)  
toTitleCase(phrase)
```

```
## [1] "Bananas are the Tastiest"
```

```
odd <- "  bananas are the tastiest  "  
trimws(odd)
```

```
## [1] "bananas are the tastiest"
```

Splitting a string

```
str_split(phrase, pattern = " ")
```

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

```
## [1] "bananas" "are" "the" "tastiest"
```

Splitting a string

```
str_split(phrase, pattern = " ")
```

```
## [[1]]  
## [1] "bananas" "are" "the" "tastiest"
```

```
str_split(phrase, pattern = "a")
```

```
## [[1]]  
## [1] "b" "n" "n" "s" "re the t" "stiest"
```

Splitting a string to a fixed matrix

```
str_split_fixed(phrase, pattern = " ", n = 2)
```

```
##      [,1]      [,2]  
## [1,] "bananas" "are the tastiest"
```

Splitting a string to a fixed matrix

```
str_split_fixed(phrase, pattern = " ", n = 2)
```

```
##      [,1]      [,2]  
## [1,] "bananas" "are the tastiest"
```

```
str_split_fixed(phrase, pattern = " ", n = 3)
```

```
##      [,1]      [,2] [,3]  
## [1,] "bananas" "are"  "the tastiest"
```


Splitting a string to a fixed matrix

```
str_split_fixed(phrase, pattern = " ", n = 2)
```

```
##      [,1]      [,2]  
## [1,] "bananas" "are the tastiest"
```

```
str_split_fixed(phrase, pattern = " ", n = 3)
```

```
##      [,1]      [,2] [,3]  
## [1,] "bananas" "are" "the tastiest"
```

```
str_split_fixed(phrase, pattern = " ", n = 4)
```

```
##      [,1]      [,2] [,3] [,4]  
## [1,] "bananas" "are" "the" "tastiest"
```

Finding strings in strings

```
phrase
```

```
## [1] "bananas are the tastiest"
```

```
str_detect(phrase, "are")
```

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

Finding strings in strings

```
phrase
```

```
## [1] "bananas are the tastiest"
```

```
str_detect(phrase, "are")
```

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

```
str_detect(phrase, "scrumptious")
```

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

```
str_detect(phrase, "nana")
```

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

Squishing strings together

```
str_c(phrase, "seriously")
```

```
## [1] "bananas are the tastiestseriously"
```

```
### oops
```

```
str_c(phrase, "seriously", sep = " ")
```

```
## [1] "bananas are the tastiest seriously"
```

```
### or
```

```
str_c(phrase, " seriously")
```

```
## [1] "bananas are the tastiest seriously"
```

```
## not this
```

```
str_c(phrase, "seriously", sep = "!!")
```

```
## [1] "bananas are the tastiest!!seriously"
```

But we usually work with vectors!

- This is true
- All of this works on vectors
- Like a vector of fruits!

But we usually work with vectors!

- This is true
- All of this works on vectors
- Like a vector of fruits!

fruit

```
## [1] "apple"           "apricot"          "avocado"
## [4] "banana"          "bell pepper"      "bilberry"
## [7] "blackberry"      "blackcurrant"     "blood orange"
## [10] "blueberry"       "boysenberry"      "breadfruit"
## [13] "canary melon"    "cantaloupe"       "cherimoya"
## [16] "cherry"          "chili pepper"     "clementine"
## [19] "cloudberry"      "coconut"          "cranberry"
## [22] "cucumber"        "currant"          "damson"
## [25] "date"            "dragonfruit"      "durian"
## [28] "eggplant"        "elderberry"       "feijoa"
## [31] "fig"             "goji berry"       "gooseberry"
## [34] "grape"           "grapefruit"       "guava"
## [37] "honeydew"        "huckleberry"     "jackfruit"
## [40] "jambul"          "jujube"           "kiwi fruit"
## [43] "kumquat"         "lemon"            "lime"
## [46] "loquat"          "lychee"           "mandarine"
## [49] "mango"           "mulberry"         "nectarine"
## [52] "nut"             "olive"            "orange"
## [55] "pamelo"          "papaya"           "passionfruit"
## [58] "peach"           "pear"             "persimmon"
## [61] "physalis"        "pineapple"        "plum"
```

See, it works on vectors!

```
str_sub(fruit, 1, 2)
```

```
## [1] "ap" "ap" "av" "ba" "be" "bi" "bl" "bl" "bl" "bl" "bo" "br" "ca" "ca" "ch"  
## [16] "ch" "ch" "cl" "cl" "co" "cr" "cu" "cu" "da" "da" "dr" "du" "eg" "el" "fe"  
## [31] "fi" "go" "go" "gr" "gr" "gu" "ho" "hu" "ja" "ja" "ju" "ki" "ku" "le" "li"  
## [46] "lo" "ly" "ma" "ma" "mu" "ne" "nu" "ol" "or" "pa" "pa" "pa" "pe" "pe" "pe"  
## [61] "ph" "pi" "pl" "po" "po" "pu" "qu" "ra" "ra" "ra" "re" "ro" "sa" "sa" "st"  
## [76] "st" "ta" "ta" "ug" "wa"
```

Let's see how many fruits use the word "fruit"

```
fruit %>%  
  str_detect("fruit")
```

```
## [1] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE TRUE  
## [13] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE  
## [25] FALSE TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE TRUE FALSE  
## [37] FALSE FALSE TRUE FALSE FALSE TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE  
## [49] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE TRUE FALSE FALSE FALSE  
## [61] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE  
## [73] FALSE FALSE TRUE FALSE FALSE FALSE TRUE FALSE
```


Let's see how many fruits use the word "fruit"

```
fruit %>%  
  str_detect("fruit")
```

```
## [1] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE TRUE  
## [13] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE  
## [25] FALSE TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE TRUE FALSE  
## [37] FALSE FALSE TRUE FALSE FALSE TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE  
## [49] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE TRUE FALSE FALSE FALSE  
## [61] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE  
## [73] FALSE FALSE TRUE FALSE FALSE FALSE TRUE FALSE
```

```
## How many?
```

```
fruit %>%  
  str_detect("fruit") %>%  
  sum()
```

```
## [1] 8
```

Let's get those fruits that are called fruits

```
fruitfruits <- str_subset(fruit, "fruit")
```

Let's make them all one word

```
fruitfruits<-str_replace(
```

```
  fruitfruits,  
  pattern = " ",  
  replacement= "")
```

```
fruitfruits
```

```
## [1] "breadfruit" "dragonfruit" "grapefruit" "jackfruit" "kiwifruit"  
## [6] "passionfruit" "starfruit" "uglifruit"
```

Let's make them all two words

```
fruitfruits<-str_replace(
```

```
  fruitfruits,  
  pattern = "fruit",  
  replacement = " fruit")
```

```
fruitfruits
```

```
## [1] "bread fruit"    "dragon fruit"   "grape fruit"    "jack fruit"  
## [5] "kiwi fruit"     "passion fruit"  "star fruit"     "ugli fruit"
```

Using str_replace to handle NA

```
melons <- str_subset(fruit, pattern = "melon")  
melons[2] <- NA  
melons
```

```
## [1] "canary melon" NA "watermelon"
```

```
# > [1] 'canary melon' NA 'watermelon'  
str_replace_na(melons, "UNKNOWN MELON")
```

```
## [1] "canary melon" "UNKNOWN MELON" "watermelon"
```

Moving on to some more practical
examples

```
titanic <- read_csv("./data/titanic.csv")
```

A handy trick

```
tolower(names(titanic))
```

```
## [1] "survived"          "pclass"  
## [3] "name"              "sex"  
## [5] "age"               "siblings/spouses aboard"  
## [7] "parents/children aboard" "fare"
```

```
names(titanic) <- tolower(names(titanic))
```


Let's see what titles people used

```
titanic_titles <- titanic %>%  
  separate(name, into = c("title", "name"), sep = "\\.")  
## the \\ is there because . has a special meaning in regex (we'll come back to  
## that)  
titanic
```

```
## # A tibble: 887 x 8  
##   survived pclass name                sex    age siblings/spouses abo~1  
##     <dbl> <dbl> <chr>                <chr> <dbl>          <dbl>  
## 1         0      3 Mr. Owen Harris Braund    male    22             1  
## 2         1      1 Mrs. John Bradley (Floren~ fema~    38             1  
## 3         1      3 Miss. Laina Heikkinen     fema~    26             0  
## 4         1      1 Mrs. Jacques Heath (Lily ~ fema~    35             1  
## 5         0      3 Mr. William Henry Allen   male    35             0  
## 6         0      3 Mr. James Moran           male    27             0  
## 7         0      1 Mr. Timothy J McCarthy    male    54             0  
## 8         0      3 Master. Gosta Leonard Pal~ male     2             3  
## 9         1      3 Mrs. Oscar W (Elisabeth V~ fema~    27             0  
## 10        1      2 Mrs. Nicholas (Adele Ache~ fema~    14             1  
## # i 877 more rows  
## # i abbreviated name: 1: 'siblings/spouses aboard'  
## # i 2 more variables: 'parents/children aboard' <dbl>, fare <dbl>
```

Titles on the Titanic

```
unique(titanic_titles$title)
```

```
## [1] "Mr"          "Mrs"         "Miss"        "Master"      "Don"
## [6] "Rev"         "Dr"          "Mme"         "Ms"          "Major"
## [11] "Lady"        "Sir"         "Mlle"        "Col"         "Capt"
## [16] "the Countess" "Jonkheer"
```

Who's Jonkheer? Who's the Countess?

```
grep("Jonkheer", titanic$name)
```

```
## [1] 819
```

```
grep("the Countess", titanic$name)
```

```
## [1] 756
```

Both use regular expressions to match patterns in strings.

- `grep()` returns the index of matches (ie row number)
- `grepl()` returns TRUE or FALSE for matches
- Regular expressions (or regex) are super powerful and super confusing.
- Here's a cheat sheet (<https://www.rstudio.com/wp-content/uploads/2016/09/RegExCheatsheet.pdf>)
- Most of the time, we don't need to worry about regex. But special characters can trip you up.

Special characters in regex

`\^$.|?*()[{}"`

How many countesses are there?

```
table(grepl("the Countess", titanic$name))
```

```
##
```

```
## FALSE TRUE
```

```
##    886    1
```

Who was she?!

```
titanic[grep("the Countess", titanic$name), ]
```

```
## # A tibble: 1 x 8
##   survived pclass name                sex    age siblings/spouses abo~1
##   <dbl>   <dbl> <chr>                <chr> <dbl>          <dbl>
## 1       1     1   the Countess. of (Lucy Noe~ fema~   33          0
## # i abbreviated name: 1: 'siblings/spouses aboard'
## # i 2 more variables: 'parents/children aboard' <dbl>, fare <dbl>
```

```
titanic %>%
  filter(grepl("the Countess", titanic$name))
```

```
## # A tibble: 1 x 8
##   survived pclass name                sex    age siblings/spouses abo~1
##   <dbl>   <dbl> <chr>                <chr> <dbl>          <dbl>
## 1       1     1   the Countess. of (Lucy Noe~ fema~   33          0
## # i abbreviated name: 1: 'siblings/spouses aboard'
## # i 2 more variables: 'parents/children aboard' <dbl>, fare <dbl>
```

Let's just get her title and name

```
titanic %>%  
  filter(grepl("the Countess", titanic$name)) %>%  
  select(name)  
  
## # A tibble: 1 x 1  
##   name  
##   <chr>  
## 1 the Countess. of (Lucy Noel Martha Dyer-Edwards) Rothes
```

Recoding - ifelse and case_when

The ifelse() function

ifelse() commands require the following:

1. test: a conditional statement that returns TRUE or FALSE
2. yes: a value assigned when test==TRUE
3. no: a value assigned when test==FALSE

if and else

```
a <- 1
b <- 1
if (a == b) {
  "equal!"
} else {
  "not equal!"
}
```

```
## [1] "equal!"
```

```
if (a != b) {
  "not equal!"
} else {
  "equal!"
}
```

```
## [1] "equal!"
```

What if we want a comparison of each element in the vector?

```
a <- c(1, 2)
b <- c(2, 3)
ifelse(a == b, "equal!", "not equal!")
```

```
## [1] "not equal!" "not equal!"
```

We can use this to do all kinds of neat things.

We're going to be cruel for a moment

Let's add "You died" to the front of any the name of any passenger who died

```
cruelty<-titanic%>%  
  mutate(  
  
    name =  
      ifelse(  
        survived == 0,  
        str_c("You died", name, sep = " "),  
        name)  
  )%>%  
  select(survived, name)
```

What did it do?

```
cruelty
```

```
## # A tibble: 887 x 2
##   survived name
##   <dbl> <chr>
## 1      0 You died Mr. Owen Harris Braund
## 2      1 Mrs. John Bradley (Florence Briggs Thayer) Cumings
## 3      1 Miss. Laina Heikkinen
## 4      1 Mrs. Jacques Heath (Lily May Peel) Futrelle
## 5      0 You died Mr. William Henry Allen
## 6      0 You died Mr. James Moran
## 7      0 You died Mr. Timothy J McCarthy
## 8      0 You died Master. Gosta Leonard Palsson
## 9      1 Mrs. Oscar W (Elisabeth Vilhelmina Berg) Johnson
## 10     1 Mrs. Nicholas (Adele Achem) Nasser
## # i 877 more rows
```

Let's add a new variable - child

```
kids<-titanic%>%  
  mutate(  
    child = ifelse(age<18,  
                   "Child",  
                   "Adult"))
```

```
table(kids$child)
```

```
##  
## Adult Child  
##    757    130
```

```
table(titanic$age<18)
```

```
##  
## FALSE  TRUE  
##    757    130
```

Let's recode the variable sex

```
recode<-titanic%>%  
  mutate(  
  
    sex = ifelse(sex == "male",  
                  "m",  
                  "f"))
```

But what if we have more than one condition to evaluate?

Let's make a three category age variable: child, adult, elder

We could nest ifelse() commands:

```
age_recode<-titanic%>%  
  mutate(age_cat =  
  
          ifelse(age<18, "child",  
                  ifelse(age>65,  
                          "elder",  
                          "adult"))))
```

```
table(age_recode$age_cat)
```

```
##
```

```
## adult child elder
```

```
##    747    130     10
```


But that's hard to read and can get cumbersome with many categories

`case_when()` is a flexible approach to link together many conditional statements

```
age_recode2<-titanic%>%  
  mutate(age_cat =  
  
    case_when(  
      age < 18 ~ "child",  
      age >= 18 & age <= 65 ~ "adult",  
      age >65 ~ "elder"  
  
    ))
```

```
table(age_recode2$age_cat)
```

```
##  
## adult child elder  
##    747    130     10
```

A real example: HW 6

```
fe <- read_csv("./data/fe_1_25_19.csv")  
unique(fe$`Subject's age`)
```

```
## [1] "24"      "53"      "55"      "25"      "23"      "45"  
## [7] "20"      "29"      "31"      "19"      "36"      "28"  
## [13] "35"      NA        "26"      "41"      "68"      "49"  
## [19] "17"      "27"      "44"      "50"      "43"      "38"  
## [25] "21"      "32"      "34"      "14"      "18"      "33"  
## [31] "15"      "22"      "1"       "57"      "88"      "40"  
## [37] "37"      "48"      "85"      "56"      "42"      "52"  
## [43] "46"      "63"      "16"      "30"      "74"      "60"  
## [49] "59"      "51"      "69"      "13"      "10"      "47"  
## [55] "66"      "39"      "79"      "54"      "65"      "75"  
## [61] "20s"     "7"       "6"       "3"       "5"       "11"  
## [67] "72"      "58"      "71"      "12"      "80"      "78"  
## [73] "61"      "73"      "67"      "70"      "77"      "76"  
## [79] "8"       "9"       "64"      "62"      "4"       "83"  
## [85] "2"       "89"      "60s"     "18-25"   "18 months" "46/53"  
## [91] "3 months" "40s"     "30s"     "84"      "90"      "50s"  
## [97] "81"      "87"      "6 months" "9 months" "10 months" "86"  
## [103] "92"      "2 months" "7 months" "82"      "8 months" "91"  
## [109] "3 days"   "55."     "20s-30s" "95"      "101"     "107"  
## [115] "40-50"    "25-30"   "97"      "24-25"   "93"      "45 or 49"  
## [121] "25'"      "4 months" "70s"     "11 mon"  "7 mon"
```

More messy data

```
unique(fe$`Subject's gender`)
```

```
## [1] "Female"      "Male"        NA            "Transgender" "Femalr"  
## [6] "Transexual"  "White"
```

- Data cleaning involves writing code to solve problems in the raw data.
- We write programs that search out and fix issues so that we can conduct needed analysis.
- NEVER modify the original data. Doing so is not reproducible or documented.