Data Cleaning

Data Wrangling in R

Data Cleaning

In general, data cleaning is a process of investigating your data for inaccuracies, or recoding it in a way that makes it more manageable.

MOST IMPORTANT RULE - LOOK AT YOUR DATA!

Useful checking functions

- `is.na()` - is `TRUE` if the data is `FALSE` otherwise
- `!` - negation (NOT)
  - if `is.na(x)` is `TRUE`, then `!is.na(x)` is `FALSE`
- `all()` takes in a `logical` and will be `TRUE` if ALL are `TRUE`
  - `all(!is.na(x))` - are all values of `x` NOT NA
- `any()` will be `TRUE` if ANY are true
  - `any(is.na(x))` - do we have any `NA`'s in `x`?
- `complete.cases()` - returns `TRUE` if EVERY value of a row is NOT `NA`
  - very stringent condition
  - `FALSE` missing one value (even if not important)

Read in the UFO dataset

Read in data from RStudio Cloud or download from: http://sisbid.github.io/Data-Wrangling/data/ufo/ufo_data_complete.csv.gz

```
ufo = read_csv("../data/ufo/ufo_data_complete.csv")
```

```
── Column specification ────────────────────────────────────────────────────────
cols(
  datetime = col_character(),
  city = col_character(),
  state = col_character(),
  country = col_character(),
  shape = col_character(),
  `duration (seconds)` = col_double(),
  `duration (hours/min)` = col_character(),
  comments = col_character(),
  `date posted` = col_character(),
  latitude = col_character(),
  longitude = col_double()
)

Warning: 199 parsing failures.
 row col   expected     actual                                      file
 877  -- 11 columns 12 columns '../data/ufo/ufo_data_complete.csv'
1712  -- 11 columns 12 columns '../data/ufo/ufo_data_complete.csv'
1814  -- 11 columns 12 columns '../data/ufo/ufo_data_complete.csv'
2857  -- 11 columns 12 columns '../data/ufo/ufo_data_complete.csv'
3733  -- 11 columns 12 columns '../data/ufo/ufo_data_complete.csv'
```

Clean names with the `clean_names()` function from the `janitor` package

```
colnames(ufo)
```

```
 [1] "datetime"             "city"                 "state"
 [4] "country"              "shape"                "duration (seconds)"
 [7] "duration (hours/min)" "comments"             "date posted"
[10] "latitude"             "longitude"
```

```
ufo = clean_names(ufo)
colnames(ufo)
```

```
 [1] "datetime"           "city"               "state"
 [4] "country"            "shape"              "duration_seconds"
 [7] "duration_hours_min" "comments"           "date_posted"
[10] "latitude"           "longitude"
```

Data cleaning "before" R

You saw warning messages when reading in this dataset. We can see these with the `problems()` function from `readr`.

```
p = problems(ufo)
p
```

```
# A tibble: 199 x 5
     row col   expected   actual     file
   <int> <chr> <chr>      <chr>      <chr>
 1   877 <NA>  11 columns 12 columns '../data/ufo/ufo_data_complete.csv'
 2  1712 <NA>  11 columns 12 columns '../data/ufo/ufo_data_complete.csv'
 3  1814 <NA>  11 columns 12 columns '../data/ufo/ufo_data_complete.csv'
 4  2857 <NA>  11 columns 12 columns '../data/ufo/ufo_data_complete.csv'
 5  3733 <NA>  11 columns 12 columns '../data/ufo/ufo_data_complete.csv'
 6  4755 <NA>  11 columns 12 columns '../data/ufo/ufo_data_complete.csv'
 7  5388 <NA>  11 columns 12 columns '../data/ufo/ufo_data_complete.csv'
 8  5422 <NA>  11 columns 12 columns '../data/ufo/ufo_data_complete.csv'
 9  5613 <NA>  11 columns 12 columns '../data/ufo/ufo_data_complete.csv'
10  5848 <NA>  11 columns 12 columns '../data/ufo/ufo_data_complete.csv'
# … with 189 more rows
```

Let's just drop those problematic rows for now.

Though you would usually want to check them!

```
ufo = ufo[-p$row,] # brackets can also be used for subsetting
```

Checking for logical conditions

- `any()` - checks if there are any `TRUE`s
- `all()` - checks if ALL are true

```
any(is.na(ufo$state)) # are there any NAs?
```

```
[1] TRUE
```

```
all(is.na(ufo$state)) # are the values all NAs?
```

```
[1] FALSE
```

Recoding Variables

Example of Cleaning: more complicated

For example, let's say gender was coded as Male, M, m, Female, F, f. Using Excel to find all of these would be a matter of filtering and changing all by hand or using if statements.

Sometimes though, it's not so simple. That's where functions that find patterns come to be very useful.

```
table(gender)
```

```
gender
     F FeMAle FEMALE     Fm      M     Ma    mAle   Male   MaLe   MALE    Man
    80     88     76     87     99     76     84     83     79     93     84
 Woman
    71
```

Example of Cleaning: more complicated

In R, you could use `case_when()`:

```
#case_when way:
data_gen <-data_gen %>% mutate(gender =
                      case_when(gender %in% c("Male", "M", "m", "Man")
                                ~ "Male",
                           TRUE ~ gender))
head(data_gen)
```

```
# A tibble: 6 x 1
  gender
  <chr>
1 F
2 Fm
3 MaLe
4 MaLe
5 FeMAle
6 FEMALE
```

Oh dear! This only fixes some values! It is difficult to notice values like `"MaLe"`.

String functions

The `stringr` package

Like `dplyr`, the `stringr` package:

- Makes some things more intuitive
- Is different than base R
- Is used on forums for answers
- Has a standard format for most functions: `str_`
    - the first argument is a string like first argument is a `data.frame` in `dplyr`

Useful String Functions

Useful String functions from base R and `stringr`

- `toupper()`, `tolower()` - uppercase or lowercase your data
- `str_sentence()` - uppercase just the first character (in the `stringr` package)
- `paste()` - paste strings together with a space
- `paste0` - paste strings together with no space as default
- `str_trim()` (in the `stringr` package) or `trimws` in base
  - will trim whitespace
- `nchar` - get the number of characters in a string

recoding with `str_to_sentence()`

```
#case_when way:
data_gen <-data_gen %>%
                mutate(gender = str_to_sentence(gender)) %>%
                mutate(gender =
                       case_when(gender %in% c("Male", "M", "m", "Man")
                                 ~ "Male",
                                 TRUE ~ gender))
head(data_gen)
```

```
# A tibble: 6 x 1
  gender
  <chr>
1 F
2 Fm
3 Male
4 Male
5 Female
6 Female
```

OK, now we are getting somewhere!

Pasting strings with `paste` and `paste0`

```
paste("Visit", 1:5, sep = "_")
```

```
[1] "Visit_1" "Visit_2" "Visit_3" "Visit_4" "Visit_5"
```

```
paste("Visit", 1:5, sep = "_", collapse = " ")
```

```
[1] "Visit_1 Visit_2 Visit_3 Visit_4 Visit_5"
```

```
# and paste0 can be even simpler see ?paste0
paste0("Visit", 1:5)
```

```
[1] "Visit1" "Visit2" "Visit3" "Visit4" "Visit5"
```

Paste

```
paste("Visit", 1:5, sep = "_") %>% length()
```

```
[1] 5
```

```
paste("Visit", 1:5, sep = "_", collapse = " ") %>% length()
```

```
[1] 1
```

Fancier pastes

```
paste("To", "is going be the ", "we go to the store!", sep = "day ")
```

```
[1] "Today is going be the day we go to the store!"
```

Substringing

`stringr`

- `str_sub(x, start, end)` - substrings from position start to position end
- `str_split(string, pattern)` - splits strings up - returns list! [we'll revisit in "Functional Programming"]

Substringing

Examples:

```
str_sub("I like friesian horses", 8,12)
```

```
[1] "fries"
```

```
#123456789101112
#I like fries
str_sub(c("Site A", "Site B", "Site C"), 6,6)
```

```
[1] "A" "B" "C"
```

Splitting/Find/Replace and Regular Expressions

- R can do much more than find exact matches for a whole string
- Like Perl and other languages, it can use regular expressions.
- What are regular expressions?
  - Ways to search for specific strings
  - Can be very complicated or simple
  - Highly Useful - think “Find” on steroids

A bit on Regular Expressions

- http://www.regular-expressions.info/reference.html
- They can use to match a large number of strings in one statement
- `.` matches any single character
- `*` means repeat as many (even if 0) more times the last character
- `?` makes the last thing optional
- `^` matches start of vector `^a` - starts with “a”
- `$` matches end of vector `b$` - ends with “b”

'Find' functions: `stringr`

`str_detect`, `str_subset`, `str_replace`, and `str_replace_all` search for matches to argument pattern within each element of a character vector: they differ in the format of and amount of detail in the results.

- `str_detect` - returns `TRUE` if `pattern` is found
- `str_subset` - returns only the strings which pattern were detected
  - convenient wrapper around `x[str_detect(x, pattern)]`
- `str_extract` - returns only strings which pattern were detected, but ONLY the pattern
- `str_replace` - replaces `pattern` with `replacement` the first time
- `str_replace_all` - replaces `pattern` with `replacement` as many times matched

Let's look at modifier for `stringr`

```
?modifiers
```

- `fixed` - match everything exactly
- `regexp` - default - uses **reg**ular **exp**ressions
- `ignore_case` is an option to not have to use `tolower`

'Find' functions: Finding Indices

These are the indices where the pattern match occurs:

```
str_detect(ufo$comments, "two aliens") %>% head()
```

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

```
str_detect(ufo$comments, "two aliens") %>% table()
```

```
.
FALSE  TRUE
88639     2
```

```
which(str_detect(ufo$comments, "two aliens"))
```

```
[1]  1728 61579
```

'Find' functions: Finding Logicals

`filter()` using `str_detect()` gives a tibble:

```
filter(ufo, str_detect(comments, "two aliens"))
```

```
# A tibble: 2 x 11
  datetime  city  state country shape duration_seconds duration_hours_… comments
  <chr>     <chr> <chr> <chr>   <chr>            <dbl> <chr>            <chr>
1 10/14/20… yuma  va    us      form…              300 5 minutes        ((HOAX?…
2 7/1/2007… nort… ct    <NA>    unkn…               60 1 minute         Witness…
# … with 3 more variables: date_posted <chr>, latitude <chr>, longitude <dbl>
```

```
filter(ufo, str_detect(comments, "two aliens")) %>% select(comments)
```

```
# A tibble: 2 x 1
  comments
  <chr>
1 ((HOAX??))  two aliens appeared from a bright light to peacefully investigate…
2 Witnessed two aliens walking along baseball field fence.
```

'Find' functions: `str_subset()` is easier

`str_subset()` gives the values that match the pattern:

```
str_subset(ufo$comments, "two aliens")
```

```
[1] "((HOAX??))  two aliens appeared from a bright light to peacefully investigate the surroundings in the woods"
[2] "Witnessed two aliens walking along baseball field fence."
```

Showing difference in `str_extract`

`str_extract` extracts just the matched string

```
ss = str_extract(ufo$comments, "two aliens")
head(ss)
```

```
[1] NA NA NA NA NA NA
```

```
ss[ !is.na(ss)]
```

```
[1] "two aliens" "two aliens"
```

- Look for any comment that starts with “aliens”

```
str_subset(ufo$comments, "^aliens.*")
```

```
[1] "aliens speak german???" "aliens exist"           "aliens in srilanka"
```

Using Regular Expressions

That contains space then ship maybe with stuff in between

```
str_subset(ufo$comments, "space.?ship") %>% head(4) # gets "spaceship" or "space ship" or...
```

```
[1] "I saw the cylinder shaped looked like a spaceship hovring above the east side of the Air Force base. Saw it for
[2] "description of a spaceship spotted over Birmingham Alabama in 1967."
[3] "A space ship was descending to the ground"
[4] "On Monday october 3&#44 2005&#44 I spotted two spaceships in the sky.  The first spotted ship was what seemed t
```

```
str_subset(ufo$comments, "space.ship") %>% head(4) # no "spaceship" must have character in bw
```

```
[1] "A space ship was descending to the ground"
[2] "I saw a Silver space ship rising into the early morning sky over Houston&#44 Texas."
[3] "Saw a space ship hanging over the southern (Manzano) portion of the Sandia Mountains on evening. It was brightl
[4] "saw space ship for 5 min&#33 Got scared crapless&#33&#33&#33&#33&#33&#33&#33&#33&#33&#33&#33&#33&#33&#33&#3
```

str_replace()

Let's say we wanted to make the time information more consistent. Using `case_when()` would be very tedious and error-prone!

We can use `str_replace()` to do so.

```
head(ufo$duration_hours_min, 8)
```

```
[1] "45 minutes"   "1-2 hrs"      "20 seconds"   "1/2 hour"     "15 minutes"
[6] "5 minutes"    "about 3 mins" "20 minutes"
```

```
ufo %>% mutate(duration_hours_min =
                 str_replace(string = duration_hours_min,
                             pattern = "minutes",
                             replacement ="mins")) %>%
  pull(duration_hours_min) %>%
  head(8)
```

```
[1] "45 mins"      "1-2 hrs"      "20 seconds"   "1/2 hour"     "15 mins"
[6] "5 mins"       "about 3 mins" "20 mins"
```

Dates and times

The `[lubridate]`(https://lubridate.tidyverse.org/) package is amazing, there's no reason to use anything else.

```
library(lubridate)#need to load this one!
head(ufo$datetime)
```

```
[1] "10/10/1949 20:30" "10/10/1949 21:00" "10/10/1955 17:00" "10/10/1956 21:00"
[5] "10/10/1960 20:00" "10/10/1961 19:00"
```

```
ufo$date_posted = mdy(ufo$date_posted)
head(ufo$date_posted)
```

```
[1] "2004-04-27" "2005-12-16" "2008-01-21" "2004-01-17" "2004-01-22"
[6] "2007-04-27"
```