Data Cleaning Part 2

Data Wrangling in R

Data Cleaning Part 2

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.

```r
table(gender)
```

```
gender
     F FeMAle FEMALE     Fm      M     Ma   mAle   Male   M
    80     88     76     87     99     76     84     83
 Woman
    71
```

Example of Cleaning: more complicated

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

```r
#case_when way:
data_gen <-data_gen %>% mutate(gender =
                      case_when(gender %in% c("Male", "M",
                                          ~ "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

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",
                                         ~ "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
```

Reading in again

Now we have a chance to keep but clean these values!

```
ufo <-read_csv("https://sisbid.github.io/Data-Wrangling/dat
```

```
Warning: One or more parsing issues, call `problems()` on y
e.g.:
  dat <- vroom(...)
  problems(dat)
```

```
p <- problems(ufo)
ufo_clean <- ufo %>% slice((pull(p, row))*-1)
```

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

```r
colnames(ufo_clean)
```

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

```r
ufo_clean <- clean_names(ufo_clean)
colnames(ufo_clean)
```

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

str_detect and filter

Now let's fix our ufo data and remove those pesky backticks in the `duration_seconds` variable. First let's find them with `str_detect`.

```
ufo_clean %>% filter(str_detect( string = duration_seconds,
```

```
# A tibble: 3 x 11
  datetime      city  state country shape durat~1 durat~2 co
  <chr>         <chr> <chr> <chr>   <chr> <chr>   <chr>   <c
1 2/2/2000 19~ bouse az    us      <NA>  2`      each a~ Dr
2 4/10/2005 2~ sant~ ca    us      <NA>  8`      eight ~ 2 
3 7/21/2006 1~ ibag~ <NA>  <NA>    circ~ 0.5`    1/2 se~ Vi
# ... with 1 more variable: longitude <chr>, and abbreviate
#   1: duration_seconds, 2: duration_hours_min, 3: comments
#   5: latitude
```

str_remove

```
ufo_clean  <- ufo_clean %>% mutate(duration_seconds = str_r

ufo_clean  <- ufo_clean %>% mutate(duration_seconds = as.nu
ufo_clean %>% filter(str_detect( string = duration_seconds,
```

```
# A tibble: 0 x 11
# ... with 11 variables: datetime <chr>, city <chr>, state
#   country <chr>, shape <chr>, duration_seconds <dbl>,
#   duration_hours_min <chr>, comments <chr>, date_posted <
#   latitude <chr>, longitude <chr>
```

Paste can add things back to variables

```
ufo_clean %>% mutate(duration_seconds = paste(duration_seco
```

```
Rows: 88,679
Columns: 11
$ datetime          <chr> "10/10/1949 20:30", "10/10/1949
$ city              <chr> "san marcos", "lackland afb", "c
$ state             <chr> "tx", "tx", NA, "tx", "hi", "tn"
$ country           <chr> "us", NA, "gb", "us", "us", "us"
$ shape             <chr> "cylinder", "light", "circle", "
$ duration_seconds  <chr> "2700 sec", "7200 sec", "20 sec"
$ duration_hours_min <chr> "45 minutes", "1-2 hrs", "20 sec
$ comments          <chr> "This event took place in early
$ date_posted       <chr> "4/27/2004", "12/16/2005", "1/21
$ latitude          <chr> "29.8830556", "29.38421", "53.2"
$ longitude         <chr> "-97.9411111", "-98.581082", "-2
```

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:

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

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

```r
#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

‘Find’ functions: Finding Indices

These are the indices where the pattern match occurs:

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

```
# A tibble: 2 x 11
  datetime      city  state country shape durat~1 durat~2 co
  <chr>         <chr> <chr> <chr>   <chr>   <dbl> <chr>   <c
1 10/14/2006 ~ yuma  va    us      form~     300 5 minu~ ((
2 7/1/2007 23~ nort~ ct    <NA>    unkn~      60 1 minu~ Wi
# ... with 1 more variable: longitude <chr>, and abbreviate
#   1: duration_seconds, 2: duration_hours_min, 3: comments
#   5: latitude
```

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

```
# A tibble: 1 x 1
      n
  <int>
1     2
```

‘Find’ functions: Finding Logicals

filter() using str_detect() gives a tibble:

```r
filter(ufo_clean, str_detect(comments, "two aliens"))
```

```
# A tibble: 2 x 11
  datetime      city  state country shape durat~1 durat~2 co
  <chr>         <chr> <chr> <chr>   <chr>   <dbl> <chr>   <c
1 10/14/2006 ~ yuma  va    us      form~     300 5 minu~ ((
2 7/1/2007 23~ nort~ ct    <NA>    unkn~      60 1 minu~ Wi
# ... with 1 more variable: longitude <chr>, and abbreviate
#   1: duration_seconds, 2: duration_hours_min, 3: comments
#   5: latitude
```

```r
filter(ufo_clean, str_detect(comments, "two aliens")) %>% s
```

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

‘Find’ functions: `str_subset()` is easier

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

```
ufo_clean %>% pull(comments) %>% str_subset( "two aliens")
```

```
[1] "((HOAX??))  two aliens appeared from a bright light to
[2] "Witnessed two aliens walking along baseball field fenc
```

Showing difference in str_extract

str_extract extracts just the matched string

```
ufo_clean %>% mutate(two  = str_extract(comments, "two")) %
```

```
# A tibble: 2 x 2
  two       n
  <chr> <int>
1 two    2497
2 <NA>  86182
```

- Look for any comment that starts with “aliens”

```
ufo_clean %>% pull(comments) %>%str_subset( "^aliens.*")
```

```
[1] "aliens speak german???" "aliens exist"           "alie
```

Using Regular Expressions

That contains space then ship maybe with stuff in between

```
ufo_clean %>% pull(comments) %>%str_subset("space.?ship") %
```

```
[1] "I saw the cylinder shaped looked like a spaceship hovr
[2] "description of a spaceship spotted over Birmingham Ala
[3] "A space ship was descending to the ground"
[4] "On Monday october 3&#44 2005&#44 I spotted two spacesh
```

```
ufo_clean %>% pull(comments) %>% str_subset("space.ship") %
```

```
[1] "A space ship was descending to the ground"
[2] "I saw a Silver space ship rising into the early mornin
[3] "Saw a space ship hanging over the southern (Manzano) p
[4] "saw space ship for 5 min&#33 Got scared crapless&#33&#
```

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(pull(ufo_clean, duration_hours_min))
```

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

```
ufo_clean %>% 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'
[6] "5 mins"       "about 3 mins" "20 mins"
```

Separating columns

Better yet, you might notice that this data isn't tidy- there are more than two entries for each value - amount of time and unit. We could separate this using `separate()` from the `tidyr` package.

```
ufo_clean %>% separate(duration_hours_min,
                 into = c("duration_amount", "duration_unit
                 sep = " ") %>%
  select(duration_amount, duration_unit) %>% head()
```

```
# A tibble: 6 x 2
  duration_amount duration_unit
  <chr>           <chr>
1 45              minutes
2 1-2             hrs
3 20              seconds
4 1/2             hour
5 15              minutes
6 5               minutes
```

As you can see there is still plenty of cleaning to do!

Dates and times

The [lubridate](https://lubridate.tidyverse.org/) package is amazing for dates. Most important functions are those that look like ymd or mdy etc. They specify how a date should be interpreted.

```
library(lubridate)#need to load this one!

ufo_clean <- ufo_clean %>% mutate(date_posted = mdy(date_po
```

```
Warning: 193 failed to parse.
```

```
head(ufo_clean)
```

```
# A tibble: 6 x 11
  datetime  city  state country shape durat~1 durat~2 comme
  <chr>     <chr> <chr> <chr>   <chr>   <dbl> <chr>   <chr>
1 10/10/19~ san ~ tx    us      cyli~    2700 45 min~ This 
2 10/10/19~ lack~ tx    <NA>    light    7200 1-2 hrs 1949 
3 10/10/19~ ches~ <NA>  gb      circ~      20 20 sec~ Green
4 10/10/19~ edna  tx    us      circ~      20 1/2 ho~ My ol
5 10/10/19~ kane~ hi    us      light     900 15 min~ AS a
```

Summary

- `stringr` package has lots of helpful functions that work on vectors or variables in a data frame
- `str_detect` helps find patterns
- `str_detect` and `filter` can help you filter data based on patterns within value
- `str_extract` helps extract a pattern
- `str_sub` extracts pieces of strings based on the position of the the characters
- `str_subset` gives the values that match a pattern
- `separate` can separate columns into two
- `^` indicates the start of a string
- `$` indicates the end of a string
- the `lubridate` package is useful for dates and times

Lab

https://sisbid.github.io/Data-Wrangling/labs/data-cleaning-lab-part2.Rmd