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/Module1/data/ufo/ufo_data_complete.csv.gz

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

```
Parsed with 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'
```

Data cleaning "before" R

You saw warning messages when reading in this dataset. Let's just drop those rows for now

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

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

```
FALSE  TRUE
81268  7408
```

Recoding Variables

Example of Recoding: base R

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.

In R, you can simply do something like:

```
data$gender[data$gender %in%
    c("Male", "M", "m")] <- "Male"
```

Example of Cleaning: more complicated

Sometimes though, it's not so simple. That's where functions that find patterns come in 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
```

String functions

Useful String Functions

Useful String functions

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

Pasting strings with `paste` and `paste0`

Paste can be very useful for joining vectors together:

```
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"
```

```
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!"
```

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

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

Paste Depicting How Collapse Works

```
paste(1:5)
```

```
[1] "1" "2" "3" "4" "5"
```

```
paste(1:5, collapse = " ")
```

```
[1] "1 2 3 4 5"
```

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
    - the first argument is a string like first argument is a `data.frame` in `dplyr`

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 pizza", 8,12)
```

```
[1] "pizza"
```

```
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:

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

```
[1]  1728 61579
```

'Find' functions: Finding Logicals

These are the indices where the pattern match occurs:

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

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

'Find' functions: finding values, `stringr` and `dplyr`

```
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."
```

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

```
# A tibble: 2 x 11
  datetime city  state country shape `duration (seco… `duration (hour… comments
  <chr>    <chr> <chr> <chr>   <chr>            <dbl> <chr>            <chr>
1 10/14/2… yuma  va    us      form…              300 5 minutes        ((HOAX?…
2 7/1/200… nort… ct    <NA>    unkn…               60 1 minute         Witness…
# … with 3 more variables: `date posted` <chr>, latitude <chr>, longitude <dbl>
```

Showing differnce 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(7)
```

```
[1] "I saw the cylinder shaped looked like a spaceship hovring above the east
[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.  Th
[5] "Me and my daughter seen the most beautiful shiney spaceship. Not a UFO it
[6] "I saw a Silver space ship rising into the early morning sky over Houston&
[7] "Saw a space ship hanging over the southern (Manzano) portion of the Sandi
```

Replace

Let's say we wanted to sort the data set by latitude and longitude:

```
class(ufo$latitude)
```

```
[1] "character"
```

```
sort(c("1", "2", "10")) #  not sort correctly (order simply ranks the data)
```

```
[1] "1"  "10" "2"
```

```
order(c("1", "2", "10"))
```

```
[1] 1 3 2
```

Replace

So we must change the coordinates into a numeric:

```
head(ufo$latitude, 4)
```

```
[1] "29.8830556" "29.38421"   "53.2"       "28.9783333"
```

```
head(as.numeric(ufo$latitude), 4)
```

```
Warning in head(as.numeric(ufo$latitude), 4): NAs introduced by coercion
```

```
[1] 29.88306 29.38421 53.20000 28.97833
```

Dropping bad observations

```
dropIndex = which(is.na(as.numeric(ufo$latitude)) |
                    is.na(as.numeric(ufo$longitude)))
```

```
Warning in which(is.na(as.numeric(ufo$latitude)) |
is.na(as.numeric(ufo$longitude))): NAs introduced by coercion
```

```
ufo_clean = ufo[-dropIndex,]
dim(ufo_clean)
```

```
[1] 88675    11
```

Replacing and subbing: `stringr`

We can do the same thing (with 2 piping operations!) in dplyr

```
ufo_dplyr = ufo_clean
ufo_dplyr = ufo_dplyr %>% mutate(
  latitude = latitude %>% as.numeric,
  longitude = longitude %>% as.numeric) %>%
    arrange(latitude,longitude)
ufo_dplyr[1:5, c("datetime", "latitude", "longitude")]
```

```
# A tibble: 5 x 3
  datetime         latitude longitude
  <chr>               <dbl>     <dbl>
1 5/15/1994 13:00     -82.9     -135
2 4/14/2002 22:22     -46.4      168.
3 7/3/2002 22:35      -46.4      168.
4 10/23/2008 04:45    -46.2      170.
5 6/1/1998 22:00      -45.7      171.
```

Special characters

```
money = tibble(group = letters[1:5],
  amount = c("$12.32", "$43.64", "$765.43", "$93.31", "$12.13"))
money %>% arrange(amount)
```

```
# A tibble: 5 x 2
  group amount
  <chr> <chr>
1 e     $12.13
2 a     $12.32
3 b     $43.64
4 c     $765.43
5 d     $93.31
```

```
as.numeric(money$amount)
```

```
Warning: NAs introduced by coercion
```

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

Special characters

In the past, we would recommend just replacing the `$` sign with an empty string and convert to numeric:

```
money$amountNum = as.numeric(str_replace(money$amount, fixed("$"), ""))
money %>% arrange(amountNum)
```

```
# A tibble: 5 x 3
  group amount  amountNum
  <chr> <chr>       <dbl>
1 e     $12.13       12.1
2 a     $12.32       12.3
3 b     $43.64       43.6
4 d     $93.31       93.3
5 c     $765.43     765.
```

Special characters

But now there are better helper functions for this:

```
money$amount = parse_number(money$amount)
money %>% arrange(amount)
```

```
# A tibble: 5 x 3
  group amount amountNum
  <chr>  <dbl>     <dbl>
1 e       12.1      12.1
2 a       12.3      12.3
3 b       43.6      43.6
4 d       93.3      93.3
5 c      765.      765.
```

Special characters

Also works for internal commas:

```
parse_number(c("12,123,123.00", "12,465.10"))
```

```
[1] 12123123.0    12465.1
```