# 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!

# Read in the UFO dataset

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

```
ufo <- read_delim("../data/ufo/ufo_data_complete.txt")
```

```
New names:
* `` -> `...12`

Warning: One or more parsing issues, see `problems()` for d

Rows: 88875 Columns: 12
-- Column specification --------------------------------
Delimiter: "\t"
chr (9): datetime, city, state, country, shape, duration (h
dbl (3): duration (seconds), longitude, ...12

i Use `spec()` to retrieve the full column specification fo
i Specify the column types or set `show_col_types = FALSE`
```

# The "problems"

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: 3 x 5
    row   col expected actual file
  <int> <int> <chr>    <chr>  <chr>
1 30894     6 a double 2`     /Users/carriewright/Documents
2 39616     6 a double 8`     /Users/carriewright/Documents
3 65125     6 a double 0.5`   /Users/carriewright/Documents
```

# The "problems"

These all became NA values.

```
ufo[(p$row-1),] %>% glimpse()
```

```
Rows: 3
Columns: 12
$ datetime                  <chr> "2/2/2000 19:33", "4/10/2005
$ city                      <chr> "bouse", "santa cruz", "ibag
$ state                     <chr> "az", "ca", NA
$ country                   <chr> "us", "us", NA
$ shape                     <chr> NA, NA, "circle"
$ `duration (seconds)`      <dbl> NA, NA, NA
$ `duration (hours/min)`    <chr> "each a few seconds", "eight
$ comments                  <chr> "Driving through Plomosa Pas
$ `date posted`             <chr> "2/16/2000", "4/16/2005", "1
$ latitude                  <chr> "33.9325", "36.9741667", "4.
$ longitude                 <dbl> -114.00500, -122.02972, -75.
$ ...12                     <dbl> NA, NA, NA
```

# The “problems”

```
4/10/2005 22:52 santa cruz      ca      us              8`      eight seconds   2 red lights moving together and apart with a hazy cloaked craft inbetween. Dark sky.   4/16/2005       36.9741667      -12
38453   el  zumbador (el cobre) (venezuela)                     fireball        1200    20 minutes      ORGANIZACI&Oacute;N RESCATE HUMBOLDT / SAR / VENEZUELA http://www.rescate.com e-mail EMERGENCIAS: n
```

# Reading in again

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

```
ufo <- read_delim("../data/ufo/ufo_data_complete.txt",
                  col_types = cols("duration (seconds)" = c
```

```
New names:
* `` -> `...12`
```

```
ufo[(p$row-1),] %>% glimpse()
```

```
Rows: 3
Columns: 12
$ datetime                <chr> "2/2/2000 19:33", "4/10/2005
$ city                    <chr> "bouse", "santa cruz", "ibag
$ state                   <chr> "az", "ca", NA
$ country                 <chr> "us", "us", NA
$ shape                   <chr> NA, NA, "circle"
$ `duration (seconds)`    <chr> "2`", "8`", "0.5`"
$ `duration (hours/min)`  <chr> "each a few seconds", "eight
$ comments                <chr> "Driving through Plomosa Pas
```

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

```
colnames(ufo)
```

```
 [1] "datetime"                "city"                    "state"
 [4] "country"                 "shape"                   "duratio
 [7] "duration (hours/min)" "comments"                   "date po
[10] "latitude"                "longitude"               "...12"
```

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

```
 [1] "datetime"              "city"                  "state"
 [4] "country"               "shape"                 "duration_se
 [7] "duration_hours_min" "comments"                 "date_posted
[10] "latitude"              "longitude"             "x12"
```

# 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    M
    80      88      76      87      99      76      84      83
 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",
                                   ~ "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
```

# str_remove

Now let's fix our ufo data and remove those pesky backticks in the duration_seconds variable.

```
ufo <- ufo %>% mutate(duration_seconds = str_remove(string
                                                    pattern = "`")
ufo <- ufo %>% mutate(duration_seconds = as.numeric(duratio
ufo[(p$row-1),]
```

```
# A tibble: 3 x 12
  datetime  city  state country shape duration_seconds dura
  <chr>     <chr> <chr> <chr>   <chr>             <dbl> <ch
1 2/2/2000~ bouse az    us      <NA>                2   each
2 4/10/200~ sant~ ca    us      <NA>                8   eigh
3 7/21/200~ ibag~ <NA>  <NA>    circ~               0.5 1/2
# ... with 4 more variables: date_posted <chr>, latitude <c
#   longitude <dbl>, x12 <dbl>
```

# Paste can add things back to variables

```
head(Orange)
```

```
  Tree  age circumference
1    1  118            30
2    1  484            58
3    1  664            87
4    1 1004           115
5    1 1231           120
6    1 1372           142
```

```
Orange %>% mutate(Tree = paste(Tree, "Tree", sep = "_"))
```

```
     Tree  age circumference
1  1_Tree  118            30
2  1_Tree  484            58
3  1_Tree  664            87
4  1_Tree 1004           115
5  1_Tree 1231           120
6  1_Tree 1372           142
```

# Paste0 doesn't need a separator

```
head(Orange)
```

```
  Tree  age circumference
1    1  118            30
2    1  484            58
3    1  664            87
4    1 1004           115
5    1 1231           120
6    1 1372           142
```

```
Orange %>% mutate(Tree = paste0(Tree, "Tree"))
```

```
    Tree  age circumference
1  1Tree  118            30
2  1Tree  484            58
3  1Tree  664            87
4  1Tree 1004           115
5  1Tree 1231           120
6  1Tree 1372           142
```

# 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

# '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
88747     2
```

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

```
[1]  1730 61724
```

# 'Find' functions: Finding Logicals

filter() using str_detect() gives a tibble:

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

```
# A tibble: 2 x 12
  datetime  city  state country shape duration_seconds dura
  <chr>     <chr> <chr> <chr>   <chr>             <dbl> <chr
1 10/14/20~ yuma  va    us      form~               300 5 mi
2 7/1/2007~ nort~ ct    <NA>    unkn~                60 1 mi
# ... with 4 more variables: date_posted <chr>, latitude <c
#   longitude <dbl>, x12 <dbl>
```

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

```
# 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:

```
str_subset(ufo$comments, "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

```
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"                "alie
```

# Using Regular Expressions

That contains space then ship maybe with stuff in between

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

```
[1] "I saw the cylinder shaped looked like a spaceship hovi
[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
```

```
str_subset(ufo$comments, "space.ship") %>% head(4) # no "sp
```

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

```
[1] "45 minutes"   "1-2 hrs"      "20 seconds"   "1/2 hour"
[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"
[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
[5] "10/10/1960 20:00" "10/10/1961 19:00"
```

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

```
Warning: 194 failed to parse.
```

```
head(ufo$date_posted)
```

```
[1] "2004-04-27" "2005-12-16" "2008-01-21" "2004-01-17" "20
[6] "2007-04-27"
```