Subsetting Data in R

Data Wrangling in R

Dealing with Missing Data

Missing data types

One of the most important aspects of data cleaning is missing values.

Types of "missing" data:

- `NA` - general missing data
- `NaN` - stands for "**N**ot **a** **N**umber", happens when you do 0/0.
- `Inf` and `-Inf` - Infinity, happens when you take a positive number (or negative number) by 0.

Finding Missing data

Each missing data type has a function that returns `TRUE` if the data is missing:

- `NA` - `is.na()`
- `NaN` - `is.nan()`
- `Inf` and `-Inf` - `is.infinite()`
- `is.finite()` returns `FALSE` for all missing data and `TRUE` for non-missing

Working with is.na()

```
t <- c(1,0,3,0)
x <-c(0,0,3,NA)
t / x
```

```
[1] Inf NaN   1  NA
```

```
is.na(t/x)
```

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

Missing Data with Logicals

One important aspect (esp with subsetting) is that logical operations return `NA` for `NA` values. The data could be `> 2` or not. We don't know, so `R` says there is no `TRUE` or `FALSE`, and instead that is missing:

```
x <- c(0, NA, 2, 3, 4)
x > 2
```

```
[1] FALSE    NA FALSE  TRUE  TRUE
```

Missing Data with Logicals

What to do? What if we want if `x > 2` and `x` isn't `NA`?

```
x
```

```
[1]  0 NA  2  3  4
```

```
!is.na(x)
```

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

```
x > 2 & !is.na(x)
```

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

Missing Data with Operations

Similarly with logicals, operations/arithmetic with `NA` will result in `NA`s:

```
x + 2
```

```
[1]  2 NA  4  5  6
```

```
x * 2
```

```
[1]  0 NA  4  6  8
```

UFO data again

```
ufo = read_csv("../data/ufo/ufo_data_complete.csv", col_typ
                  cols(
                    .default = col_character(),
                    `duration (seconds)` = col_double(),
                    longitude = col_double()
                  ))
head(ufo)
```

```
# A tibble: 6 x 11
  datetime  city  state country shape `duration (sec~` `dur
  <chr>     <chr> <chr> <chr>   <chr>              <dbl> <chr
1 10/10/19~ san ~ tx    us      cyli~               2700 45 m
2 10/10/19~ lack~ tx    <NA>    light               7200 1-2
3 10/10/19~ ches~ <NA>  gb      circ~                 20 20 s
4 10/10/19~ edna  tx    us      circ~                 20 1/2
5 10/10/19~ kane~ hi    us      light                900 15 m
6 10/10/19~ bris~ tn    us      sphe~                300 5 mi
# ... with 3 more variables: `date posted` <chr>, latitude
#   longitude <dbl>
```

Filtering and tibbles

The filter function automatically removes `NA` values.

```
count(ufo, country)
```

```
# A tibble: 6 x 2
  country     n
  <chr>   <int>
1 au        593
2 ca       3266
3 de        112
4 gb       2050
5 us      70293
6 <NA>    12561
```

```
ufo %>%filter(country == "de") %>% count(country)
```

```
# A tibble: 1 x 2
  country     n
  <chr>   <int>
1 de        112
```

Filtering for missing data

Missing value and filter can be powerful

```
ufo %>%
  filter(is.na(state) & is.na(country)) %>%
  head()
```

```
# A tibble: 6 x 11
  datetime  city  state country shape `duration (sec~` `dur
  <chr>     <chr> <chr> <chr>   <chr>              <dbl> <chr
1 10/10/19~ berm~ <NA>  <NA>    light                 20 20 s
2 10/10/19~ gisb~ <NA>  <NA>    disk                 120 2min
3 10/10/19~ zlat~ <NA>  <NA>    sphe~               1200 20 m
4 10/10/19~ lake~ <NA>  <NA>    light                300 5 mi
5 10/10/19~ turi~ <NA>  <NA>    disk                  15 15 s
6 10/10/19~ dors~ <NA>  <NA>    flash                  1 <1 s
# ... with 3 more variables: `date posted` <chr>, latitude
#   longitude <dbl>
```

Filtering and tibbles

Group logical statements with parentheses

```
ufo %>%
  filter(
    (!is.na(state) & is.na(country)) | city == "seattle") %
  head()
```

```
# A tibble: 6 x 11
  datetime  city  state country shape `duration (sec~` `dur
  <chr>     <chr> <chr> <chr>   <chr>             <dbl> <chr
1 10/10/19~ lack~ tx    <NA>    light              7200 1-2
2 10/10/19~ will~ az    <NA>    light               120 2 mi
3 10/10/19~ sadd~ ab    <NA>    tria~               270 4.5
4 10/10/19~ holm~ ny    <NA>    chev~               180 3 mi
5 10/10/19~ mani~ on    <NA>    disk                600 10/m
6 10/10/19~ kran~ ky    <NA>    tria~               180 3min
# ... with 3 more variables: `date posted` <chr>, latitude
#   longitude <dbl>
```

Renaming Columns

Renaming Columns of a data.frame

To rename columns in dplyr, you use the rename command (NEW=old)

```
ufo = ufo %>% rename(City = city, duration_s = `duration (s
glimpse(ufo)
```

```
Rows: 88,875
Columns: 11
$ datetime                  <chr> "10/10/1949 20:30", "10/10/1
$ City                      <chr> "san marcos", "lackland afb"
$ state                     <chr> "tx", "tx", NA, "tx", "hi",
$ country                   <chr> "us", NA, "gb", "us", "us",
$ shape                     <chr> "cylinder", "light", "circle
$ duration_s                <dbl> 2700, 7200, 20, 20, 900, 300
$ `duration (hours/min)`    <chr> "45 minutes", "1-2 hrs", "20
$ comments                  <chr> "This event took place in ea
$ `date posted`             <chr> "4/27/2004", "12/16/2005", "
$ latitude                  <chr> "29.8830556", "29.38421", "5
$ longitude                 <dbl> -97.941111, -98.581082, -2.9
```

Renaming All Columns of a `data.frame`: dplyr

To rename all columns you use the `rename_with` command (with a function)

```
ufo_upper = ufo %>% rename_with(toupper)
glimpse(ufo_upper)
```

```
Rows: 88,875
Columns: 11
$ DATETIME                  <chr> "10/10/1949 20:30", "10/10/1
$ CITY                      <chr> "san marcos", "lackland afb"
$ STATE                     <chr> "tx", "tx", NA, "tx", "hi",
$ COUNTRY                   <chr> "us", NA, "gb", "us", "us",
$ SHAPE                     <chr> "cylinder", "light", "circle
$ DURATION_S                <dbl> 2700, 7200, 20, 20, 900, 300
$ `DURATION (HOURS/MIN)`    <chr> "45 minutes", "1-2 hrs", "20
$ COMMENTS                  <chr> "This event took place in ea
$ `DATE POSTED`             <chr> "4/27/2004", "12/16/2005", "
$ LATITUDE                  <chr> "29.8830556", "29.38421", "5
$ LONGITUDE                 <dbl> -97.941111, -98.581082, -2.9
```

Adding columns to a `data.frame`

`mutate` - allows you to add or replace columns (IMPORTANT: need to reassign for it to stick!)

```
ufo2 <- ufo %>% mutate(State = toupper(state))
ufo2 %>% glimpse()
```

```
Rows: 88,875
Columns: 12
$ datetime                  <chr> "10/10/1949 20:30", "10/10/1
$ City                      <chr> "san marcos", "lackland afb"
$ state                     <chr> "tx", "tx", NA, "tx", "hi",
$ country                   <chr> "us", NA, "gb", "us", "us",
$ shape                     <chr> "cylinder", "light", "circle
$ duration_s                <dbl> 2700, 7200, 20, 20, 900, 300
$ `duration (hours/min)`    <chr> "45 minutes", "1-2 hrs", "20
$ comments                  <chr> "This event took place in ea
$ `date posted`             <chr> "4/27/2004", "12/16/2005", "
$ latitude                  <chr> "29.8830556", "29.38421", "5
$ longitude                 <dbl> -97.941111, -98.581082, -2.9
```

Creating conditional variables

One frequently-used tool is creating variables with conditions.

A general function for creating new variables based on existing variables is the `ifelse()` function, which "returns a value with the same shape as test which is filled with elements selected from either yes or no depending on whether the element of test is `TRUE` or `FALSE`."

```
ifelse(test, yes, no)

# test: an object which can be coerced
    to logical mode.
# yes: return values for true elements of test.
# no: return values for false elements of test.
```

Recoding to missing

Sometimes people code missing data in weird or inconsistent ways.

```
ages <-  data.frame(age = c(23,-999,21,44,32,57,65,54))
range(ages$age)
```

```
[1] -999   65
```

Recoding to create new column

Say we want to make a new column about if the age was over 30?

```
pull(ages, age)
```

```
[1]   23 -999   21   44   32   57   65   54
```

```
ages <- ages %>% mutate(over_20 = ifelse(age > 30, "Yes", "
ages
```

```
   age over_20
1   23      No
2 -999      No
3   21      No
4   44     Yes
5   32     Yes
6   57     Yes
7   65     Yes
8   54     Yes
```

Recoding value to missing

How do we change the -999 to be treated as missing for the age column?

```
pull(ages, age)
```

```
[1]   23 -999   21   44   32   57   65   54
```

```
ages <- ages %>% mutate(age = ifelse(age == -999, NA, age))
range(ages$age)
```

```
[1] NA NA
```

```
range(ages$age, na.rm=TRUE)
```

```
[1] 21 65
```

```
pull(ages, age)
```

```
[1] 23 NA 21 44 32 57 65 54
```

Adding columns to a data.frame: dplyr

```
ufo <- ufo %>% mutate(
                region = ifelse(
                country %in% c("us", "ca"),
                "North America",
                "Not North America")
              )
ufo %>% select(country, region) %>% head()
```

```
# A tibble: 6 x 2
  country region
  <chr>   <chr>
1 us      North America
2 <NA>    Not North America
3 gb      Not North America
4 us      North America
5 us      North America
6 us      North America
```

case_when provides a more general way

```
casewhen(test ~ value if test is true,
         test2 ~ vlue if test2 is true,
         TRUE ~ value if all above tests are not true) # de
```

```
ufo <- ufo %>% mutate(
            region = case_when(
              country %in% c("us", "ca") ~ "North America",
              country %in% c("de") ~ "Europe",
              country %in% "gb" ~ "Great Britain",
            ))

ufo %>% select(country, region) %>% head()
```

```
# A tibble: 6 x 2
  country region
  <chr>   <chr>
1 us      North America
2 <NA>    <NA>
3 gb      Great Britain
```

case_when defaults to NA when all tests fail

```
ufo <- ufo %>% mutate(
            region = case_when(
              country %in% c("us", "ca") ~ "North America",
              country %in% c("de") ~ "Europe"
              #country %in% "gb" ~ "Great Britain",
            ))

ufo %>% select(country, region) %>% head()
```

```
# A tibble: 6 x 2
  country region
  <chr>   <chr>
1 us      North America
2 <NA>    <NA>
3 gb      <NA>
4 us      North America
5 us      North America
6 us      North America
```

case_when() with value if all tests fail

```
ufo <- ufo %>% mutate(
            region = case_when(
              country %in% c("us", "ca") ~ "North America",
              country %in% c("de") ~ "Europe",
              country %in% "gb" ~ "Great Britain",
              TRUE ~ "Other"
            ))
ufo %>% select(country, region) %>% head()
```

```
# A tibble: 6 x 2
  country region
  <chr>   <chr>
1 us      North America
2 <NA>    Other
3 gb      Great Britain
4 us      North America
5 us      North America
6 us      North America
```

case_when() with value if all tests fail (use a variable!)

```
ufo <- ufo %>% mutate(
              region = case_when(
                country %in% "gb" ~ "Great Brit.",
                TRUE ~ region
              ))
ufo %>% select(country, region) %>% head()
```

```
# A tibble: 6 x 2
  country region
  <chr>   <chr>
1 us      North America
2 <NA>    Other
3 gb      Great Brit.
4 us      North America
5 us      North America
6 us      North America
```

Ordering the rows of a `data.frame`: dplyr

The `arrange` function can reorder rows By default, `arrange` orders in ascending order:

```
ufo %>% arrange(duration_s)
```

```
# A tibble: 88,875 x 12
   datetime         City  state country shape duration_s `du
   <chr>            <chr> <chr> <chr>   <chr>       <dbl> <ch
 1 10/10/1995 17~ puer~ pr    <NA>    <NA>            0 <NA>
 2 10/10/1999 21~ ashl~ mo    us      light           0 two
 3 10/10/2002 22~ baha~ <NA>  <NA>    egg             0 <NA>
 4 10/10/2002 22~ burn~ <NA>  au      cross           0 12
 5 10/10/2005 11~ edge~ fl    us      <NA>            0 300
 6 10/10/2005 24~ fran~ in    us      disk            0 ?
 7 10/10/2006 23~ knik  ak    us      tria~           0 5
 8 10/10/2007 05~ bake~ ca    us      circ~           0 had
 9 10/10/2008 09~ amar~ tx    us      flash           0 <NA>
10 10/10/2009 00~ gree~ <NA>  <NA>    rect~           0 <NA>
# ... with 88,865 more rows, and 4 more variables: `date po
```

Ordering the rows of a data.frame: dplyr

Use the desc to arrange the rows in descending order:

```
ufo %>% arrange(desc(duration_s))
```

```
# A tibble: 88,875 x 12
   datetime        City  state country shape duration_s `dur
   <chr>           <chr> <chr> <chr>   <chr>      <dbl> <chr
 1 10/1/1983 17:~ birm~ <NA>  gb      sphe~   97836000 31 y
 2 6/3/2010 23:30 otta~ on    ca      other   82800000 2300
 3 9/15/1991 18:~ gree~ ar    us      light   66276000 21 y
 4 4/2/1983 24:00 dont~ <NA>  <NA>    <NA>    52623200 2 mo
 5 8/10/2012 21:~ finl~ wa    us      light   52623200 2 mo
 6 8/24/2002 01:~ engl~ fl    us      light   52623200 2 mo
 7 6/30/1969 22:~ some~ <NA>  gb      cone    25248000 8 ye
 8 10/7/2013 20:~ okla~ ok    <NA>    circ~   10526400 4 mo
 9 3/1/1994 01:00 meni~ ca    us      unkn~   10526400 4 mo
10 8/3/2008 21:00 virg~ va    us      fire~   10526400 4 mo
# ... with 88,865 more rows, and 4 more variables: `date po
#   latitude <chr>, longitude <dbl>, region <chr>
```

Ordering the rows of a `data.frame`: dplyr

It is a bit more straightforward to mix increasing and decreasing orderings:

```
ufo %>% arrange(country, desc(duration_s))
```

```
# A tibble: 88,875 x 12
   datetime         City  state country shape duration_s `du
   <chr>            <chr> <chr> <chr>   <chr>      <dbl> <ch
 1 11/12/2013 21~ moun~ <NA>  au      sphe~    1209600 2 we
 2 5/12/2004 23:~ sydn~ <NA>  au      light     345600 4 da
 3 4/18/2004 12:~ sydn~ <NA>  au      light      86400 day
 4 4/15/1983 21:~ bris~ <NA>  au      chan~      37800 1 1/
 5 4/18/1996 17:~ bris~ <NA>  au      <NA>       18000 5  h
 6 6/9/2005 20:00 melb~ <NA>  au      circ~      18000 5 ho
 7 11/6/2009 22:~ pert~ <NA>  au      light      14400 4hrs
 8 3/15/2004 20:~ adel~ <NA>  au      form~      10800 1-3
 9 3/2/2014 24:00 pert~ <NA>  au      light      10800 2-3
10 6/20/2001 01:~ canb~ <NA>  au      tear~      10800 3 hr
# ... with 88,865 more rows, and 4 more variables: `date po
```

Lab

Link to Lab