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

Missing Data with Logicals

One important aspect (esp with subsetting) is that logical operations return `NA` for `NA` values. Think about it, the data could be `> 2` or not we don't know, so `R` says there is no `TRUE` or `FALSE`, so 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`?
Don't do `x != NA`, do `x > 2` and `x` is NOT `NA`:

```
x != NA
```

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

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

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

Missing Data with Logicals

What about seeing if a value is equal to multiple values? You can do `(x == 1 | x == 2) & !is.na(x)`, but that is not efficient.

```
(x == 0 | x == 2) # has NA
```

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

```
(x == 0 | x == 2) & !is.na(x) # No NA
```

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

what to do?

Missing Data with Logicals: `%in%`

Introduce the `%in%` operator:

```
x %in% c(0, 2) # NEVER has NA and returns logical
```

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

reads “return `TRUE` if `x` is in 0 or 2”. (Like `inlist` in Stata).

Missing Data with Logicals: `%in%`

NEVER has NA, even if you put it there (BUT DON'T DO THIS):

```
x %in% c(0, 2, NA) # NEVER has NA and returns logical
```

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

```
x %in% c(0, 2) | is.na(x)
```

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

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_types =
                 cols(
                   .default = col_character(),
                   `duration (seconds)` = col_double(),
                   longitude = col_double()
                 ))
head(ufo)
```

```
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'
.... ... .......... .......... ..................................
See problems(...) for more details.

# A tibble: 6 x 11
  datetime city  state country shape `duration (seco… `duration (hour… comment
  <chr>    <chr> <chr> <chr>   <chr>             <dbl> <chr>             <chr>
1 10/10/1… san … tx    us      cyli…              2700 45 minutes        This ev
2 10/10/1… lack… tx    <NA>    light              7200 1-2 hrs           1949 La
3 10/10/1… ches… <NA>  gb      circ…                20 20 seconds        Green/O
4 10/10/1… edna  tx    us      circ…                20 1/2 hour          My olde
5 10/10/1… kane… hi    us      light               900 15 minutes        AS a Ma
6 10/10/1… bris… tn    us      sphe…               300 5 minutes         My fath
# … with 3 more variables: `date posted` <chr>, latitude <chr>, longitude <dbl
```

Filtering and tibbles

Missing value and filter can be powerful (`head` - show first few rows)

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

```
# A tibble: 6 x 11
  datetime city  state country shape `duration (seco… `duration (hour… comment
  <chr>    <chr> <chr> <chr>   <chr>             <dbl> <chr>           <chr>
1 10/10/1… lack… tx    <NA>    light              7200 1-2 hrs         1949 La
2 10/10/1… ches… <NA>  gb      circ…                20 20 seconds      Green/O
3 10/10/1… pena… <NA>  gb      circ…               180 about 3 mins    penarth
4 10/10/1… berm… <NA>  <NA>    light                20 20 sec.         saw fas
5 10/10/1… will… az    <NA>    light               120 2 min           The obj
6 10/10/1… card… <NA>  gb      disk               1200 20 minutes      back ir
# … with 3 more variables: `date posted` <chr>, latitude <chr>, 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 (seco… `duration (hour… comment
  <chr>    <chr> <chr> <chr>   <chr>             <dbl> <chr>           <chr>
1 10/10/1… lack… tx    <NA>    light              7200 1-2 hrs         1949 La
2 10/10/1… will… az    <NA>    light               120 2 min           The obj
3 10/10/1… sadd… ab    <NA>    tria…               270 4.5 or more min. Lights
4 10/10/1… holm… ny    <NA>    chev…               180 3 minutes       Footbal
5 10/10/1… mani… on    <NA>    disk                600 10/mins         We coul
6 10/10/1… kran… ky    <NA>    tria…               180 3min            Triangl
# … with 3 more variables: `date posted` <chr>, latitude <chr>, 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 (seconds)`)
head(ufo)
```

```
# A tibble: 6 x 11
  datetime City  state country shape duration_s `duration (hour… comments
  <chr>    <chr> <chr> <chr>   <chr>      <dbl> <chr>            <chr>
1 10/10/1… san … tx    us      cyli…       2700 45 minutes       This ev…
2 10/10/1… lack… tx    <NA>    light       7200 1-2 hrs          1949 La…
3 10/10/1… ches… <NA>  gb      circ…         20 20 seconds       Green/O…
4 10/10/1… edna  tx    us      circ…         20 1/2 hour         My olde…
5 10/10/1… kane… hi    us      light        900 15 minutes       AS a Ma…
6 10/10/1… bris… tn    us      sphe…        300 5 minutes        My fath…
# … with 3 more variables: `date posted` <chr>, latitude <chr>, longitude <dbl
```

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

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

```
ufo_upper = ufo %>% rename_all(toupper)
head(ufo_upper)
```

```
# A tibble: 6 x 11
  DATETIME CITY  STATE COUNTRY SHAPE DURATION_S `DURATION (HOUR… COMMENTS
  <chr>    <chr> <chr> <chr>   <chr>      <dbl> <chr>            <chr>
1 10/10/1… san … tx    us      cyli…       2700 45 minutes       This ev…
2 10/10/1… lack… tx    <NA>    light       7200 1-2 hrs          1949 La…
3 10/10/1… ches… <NA>  gb      circ…         20 20 seconds       Green/O…
4 10/10/1… edna  tx    us      circ…         20 1/2 hour         My olde…
5 10/10/1… kane… hi    us      light        900 15 minutes       AS a Ma…
6 10/10/1… bris… tn    us      sphe…        300 5 minutes        My fath…
# … with 3 more variables: `DATE POSTED` <chr>, LATITUDE <chr>, LONGITUDE <dbl
```

Adding columns to a `data.frame`

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

```
ufo2 = ufo %>% mutate(State = toupper(state)) # we renamed city
ufo2 %>% select(State) %>% head
```

```
# A tibble: 6 x 1
  State
  <chr>
1 TX
2 TX
3 <NA>
4 TX
5 HI
6 TN
```

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
```

Adding new columns to a `data.frame`: base R

Can also use `$` to add columns, but only one column at a time

```
ufo2$State2 = tolower(ufo2$State)
ufo2 %>% select(state, State, State2) %>% head
```

```
# A tibble: 6 x 3
  state State State2
  <chr> <chr> <chr>
1 tx    TX    tx
2 tx    TX    tx
3 <NA>  <NA>  <NA>
4 tx    TX    tx
5 hi    HI    hi
6 tn    TN    tn
```

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

How do we change the `-999` to be treated as missing?

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

```
[1] NA NA
```

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

```
[1] 21 65
```

```
ages
```

```
  age
1  23
2  NA
3  21
4  44
5  32
6  57
7  65
8  54
```

Recoding from missing

What if you were the person that coded the `-999`

```
ages = ages %>% mutate(age = ifelse(is.na(age), -999, age))
ages
```

```
   age
1   23
2 -999
3   21
4   44
5   32
6   57
7   65
8   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
```

Adding columns to a `data.frame`: dplyr

Alternatively, `case_when` provides a more general way:

```
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
```

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 `duration (hour… comments
   <chr>    <chr> <chr> <chr>   <chr>      <dbl> <chr>            <chr>
 1 10/10/1… puer… pr    <NA>    <NA>           0 <NA>             Woman c…
 2 10/10/1… ashl… mo    us      light          0 two seperate ti… We saw …
 3 10/10/2… baha… <NA>  <NA>    egg            0 <NA>             we are …
 4 10/10/2… burn… <NA>  au      cross          0 12               the cra…
 5 10/10/2… edge… fl    us      <NA>           0 300              orange …
 6 10/10/2… fran… in    us      disk           0 ?                two yel…
 7 10/10/2… knik  ak    us      tria…          0 5                Slow mo…
 8 10/10/2… bake… ca    us      circ…          0 had a call of a… UFO sig…
 9 10/10/2… amar… tx    us      flash          0 <NA>             we saw …
10 10/10/2… gree… <NA>  <NA>    rect…          0 <NA>             Found t…
# … with 88,865 more rows, and 4 more variables: `date posted` <chr>,
#   latitude <chr>, longitude <dbl>, region <chr>
```

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 `duration (hour… comments
   <chr>    <chr> <chr> <chr>   <chr>      <dbl> <chr>            <chr>
 1 10/1/19… birm… <NA>  gb      sphe…   97836000 31 years         Firstly…
 2 6/3/201… otta… on    ca      other   82800000 23000hrs         ((HOAX?…
 3 9/15/19… gree… ar    us      light   66276000 21 years         Orange …
 4 4/2/198… dont… <NA>  <NA>    <NA>    52623200 2 months         Hi&#44 …
 5 8/10/20… finl… wa    us      light   52623200 2 months         There h…
 6 8/24/20… engl… fl    us      light   52623200 2 months         bright …
 7 6/30/19… some… <NA>  gb      cone    25248000 8 years          First t…
 8 10/7/20… okla… ok    <NA>    circ…   10526400 4 months         Bright …
 9 3/1/199… meni… ca    us      unkn…   10526400 4 months         Sun Cit…
10 8/3/200… virg… va    us      fire…   10526400 4 months         this ob…
# … with 88,865 more rows, and 4 more variables: `date posted` <chr>,
#   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 `duration (hour… comments
   <chr>    <chr> <chr> <chr>   <chr>      <dbl> <chr>            <chr>
 1 11/12/2… moun… <NA>  au      sphe…    1209600 2 weeks          Orange …
 2 5/12/20… sydn… <NA>  au      light     345600 4 days+          Infra r…
 3 4/18/20… sydn… <NA>  au      light      86400 day              It was …
 4 4/15/19… bris… <NA>  au      chan…      37800 1 1/2 hours      A brill…
 5 4/18/19… bris… <NA>  au      <NA>       18000 5  hours  plus   Five ho…
 6 6/9/200… melb… <NA>  au      circ…      18000 5 hours +        UFO sig…
 7 11/6/20… pert… <NA>  au      light      14400 4hrs             Unusual…
 8 3/15/20… adel… <NA>  au      form…      10800 1-3 hrs          ive got…
 9 3/2/201… pert… <NA>  au      light      10800 2-3 hours        Constan…
10 6/20/20… canb… <NA>  au      tear…      10800 3 hrs            8 tear …
# … with 88,865 more rows, and 4 more variables: `date posted` <chr>,
#   latitude <chr>, longitude <dbl>, region <chr>
```

Lab

Link to Lab