Intro to R

Data Cleaning

Recap on summarization

- `summary(x)`: quantile information
- `summarize`: creates a summary table of columns of interest
- `count(variable)`: how many of each unique value do you have
- `group_by()`: changes all subsequent functions
    - combine with `summarize()` to get statistics per group

Cheatsheet

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!

Dealing with Missing Data

Air quality data

The `airquality` dataset comes with R about air quality in New York in 1973.

```r
?airquality # use this to find out more about the data
```

We can use **count** to see missing values

The will be at the bottom typically

```
Ozone_values <- count(airquality, Ozone)
tail(Ozone_values)
```

```
   Ozone  n
63   115  1
64   118  1
65   122  1
66   135  1
67   168  1
68    NA 37
```

Missing Data Issues

Recall that mathematical operations with `NA` often result in `NA`s.

```
sum(c(1,2,3,NA))
```

```
[1] NA
```

```
mean(c(1,2,3,NA))
```

```
[1] NA
```

```
median(c(1,2,3,NA))
```

```
[1] NA
```

filter() and missing data

Be **careful** with missing data using subsetting!

`filter()` removes missing values by default. Because R can't tell for sure if an `NA` value meets the condition. To keep them need to add `is.na()` conditional.

Think about if this is OK or not - it depends on your data!

```
airquality %>% filter(Solar.R > 330 | is.na(Solar.R))
```

```
  Ozone Solar.R Wind Temp Month Day
1    NA      NA 14.3   56     5   5
2    28      NA 14.9   66     5   6
3     7      NA  6.9   74     5  11
4    14     334 11.5   64     5  16
5    NA      NA  8.0   57     5  27
6    NA     332 13.8   80     6  14
7    78      NA  6.9   86     8   4
8    35      NA  7.4   85     8   5
9    66      NA  4.6   87     8   6
```

To remove **rows** with **`NA`** values for a **variable** use **`drop_na()`**

A function from the `tidyr` package. (Need a data frame to start!)

Disclaimer: Don’t do this unless you have thought about if dropping `NA` values makes sense based on knowing what these values mean in your data.

```
airquality %>% drop_na(Ozone)
```

```
    Ozone Solar.R Wind Temp Month Day
1      41     190  7.4   67     5   1
2      36     118  8.0   72     5   2
3      12     149 12.6   74     5   3
4      18     313 11.5   62     5   4
5      28      NA 14.9   66     5   6
6      23     299  8.6   65     5   7
7      19      99 13.8   59     5   8
8       8      19 20.1   61     5   9
9       7      NA  6.9   74     5  11
10     16     256  9.7   69     5  12
11     11     290  9.2   66     5  13
12     14     274 10.9   68     5  14
13     18      65 13.2   58     5  15
14     14     334 11.5   64     5  16
15     34     307 12.0   66     5  17
16      6      78 18.4   57     5  18
17     30     322 11.5   68     5  19
18     11      44  9.7   62     5  20
19      1       8  9.7   59     5  21
20     11     320 16.6   73     5  22
21      4      25  9.7   61     5  23
```

To remove rows with **`NA`** values for a **data frame** use **`drop_na()`**

This function of the `tidyr` package drops rows with **`any`** missing data in **`any`** column when used on a df.

```
airquality %>% drop_na()
```

```
    Ozone Solar.R Wind Temp Month Day
1      41     190  7.4   67     5   1
2      36     118  8.0   72     5   2
3      12     149 12.6   74     5   3
4      18     313 11.5   62     5   4
5      23     299  8.6   65     5   7
6      19      99 13.8   59     5   8
7       8      19 20.1   61     5   9
8      16     256  9.7   69     5  12
9      11     290  9.2   66     5  13
10     14     274 10.9   68     5  14
11     18      65 13.2   58     5  15
12     14     334 11.5   64     5  16
13     34     307 12.0   66     5  17
14      6      78 18.4   57     5  18
15     30     322 11.5   68     5  19
16     11      44  9.7   62     5  20
17      1       8  9.7   59     5  21
18     11     320 16.6   73     5  22
19      4      25  9.7   61     5  23
20     32      92 12.0   61     5  24
21     23      13 12.0   67     5  28
22     45     252 14.9   81     5  29
23    115     223  5.7   79     5  30
```

Summary

- `count()` can help determine if we have `NA` values
- `filter()` automatically removes `NA` values - can't confirm or deny if condition is met (need `| is.na()` to keep them)
- `drop_na()` can help you remove `NA` values from a variable or an entire data frame
- `NA` values can change your calculation results
- think about what `NA` values represent - don't drop them if you shouldn't

Practice

Practice

Use `count()` and `tail()` to determine the number of missing values in the `airquality` data for the `Solar.R` variable.

```
airquality %>% count(_______) %>% _____
```

Practice

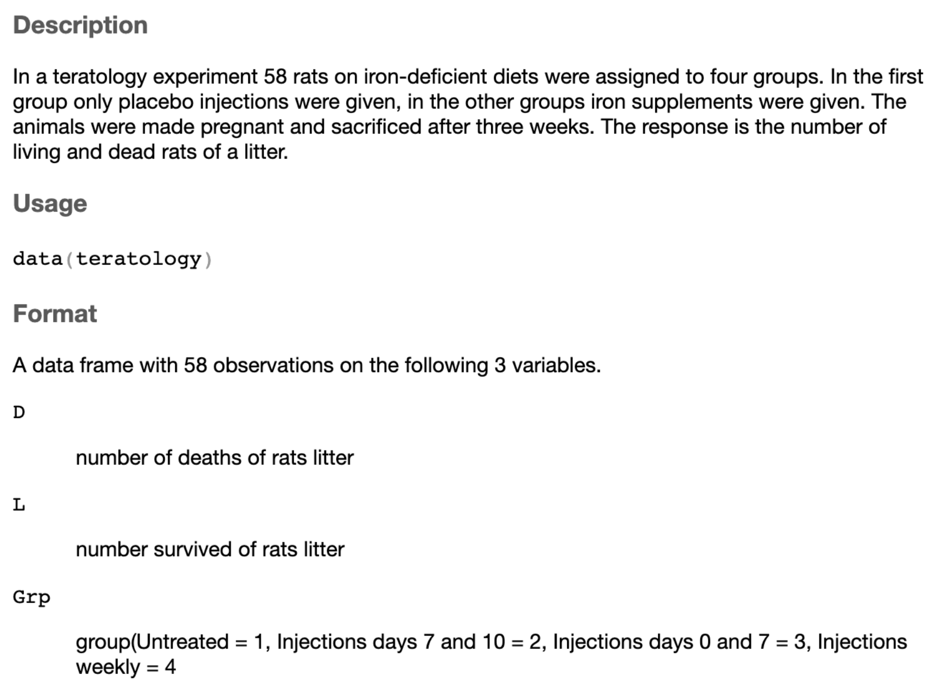
Filter the rows of `airquality` to remove rows with `NA` values for `Solar.R`.

```
filt_airqual <- _________ %>% ________(________)
```

Recoding Variables

Example of Recoding

```r
#install.packages("catdata")
library(catdata)
?catdata::teratology
data(teratology)
rat <- teratology
```



Description

In a teratology experiment 58 rats on iron-deficient diets were assigned to four groups. In the first group only placebo injections were given, in the other groups iron supplements were given. The animals were made pregnant and sacrificed after three weeks. The response is the number of living and dead rats of a litter.

Usage

```r
data(teratology)
```

Format

A data frame with 58 observations on the following 3 variables.

`D`

  number of deaths of rats litter

`L`

  number survived of rats litter

`Grp`

  group(Untreated = 1, Injections days 7 and 10 = 2, Injections days 0 and 7 = 3, Injections weekly = 4

Note about select

Once loading this `catdata` package, you need to specify that you want to use the `select` from `dplyr`. It just happens to have a function that is the same name and we want the dplyr version.

```
select <- dplyr::select
```

Oh dear…

It’s not very easy to tell what is what.

```
head(rat)
```

```
   D L Grp
1  1 9   1
2  4 7   1
3  9 3   1
4  4 0   1
5 10 0   1
6  9 2   1
```

Grp variable

```
rat %>%
  count(Grp)
```

Changing the class

We can use `as.character` or `as.numeric` to change a variable to each class respectively.

Let’s change the group to be character, since it doesn’t actually have numeric significance.

```
rat <- rat %>% mutate(Grp = as.character(Grp))
```

dplyr can help!

In `dplyr` you can use the `recode` function to change each Grp value to be something more useful!

(need `mutate` for data frames/tibbles!)

```
# General Format - this is not code!
{data_input} %>%
  mutate({new variable} = recode({Variable_fixing}, {old_value} = {new_value},
                                            {another_old_value} = {new_value}))
```

recode() function

Need quotes for values!

```
rat <-rat %>%
  mutate(Grp_recoded = recode(Grp, "1" = "Untreated",
                                   "2" = "Inj. Day 7 and 10",
                                   "3" = "Inj. Day 0 and 7",
                                   "4" = "Inj. Weekly"))
  rat %>% count(Grp_recoded)
```

```
        Grp_recoded  n
1  Inj. Day 0 and 7  5
2 Inj. Day 7 and 10 12
3       Inj. Weekly 10
4         Untreated 31
```

rename columns

Can use the `rename()` function.

```
# general format! not code!
{data you are creating or changing} <- {data you are using} %>%
                            rename({New Name} = {Old name})
```

```
head(rat, 2)
```

```
  D L Grp Grp_recoded
1 1 9   1   Untreated
2 4 7   1   Untreated
```

```
rat <- rat %>% rename("num_dead_litter" = "D",
                      "num_living_litter" = "L")
head(rat, 2)
```

```
  num_dead_litter num_living_litter Grp Grp_recoded
1               1                 9   1   Untreated
2               4                 7   1   Untreated
```

Practice

Practice

First load some data.

```
#install.packages("catdata")
library(catdata)
data(teratology2)
rat2 <-teratology2 # assign it to a new name
head(rat2)
```

```
  y Rat Grp
1 1   1  G1
2 0   1  G1
3 0   1  G1
4 0   1  G1
5 0   1  G1
6 0   1  G1
```

```
?teratology2 #find out more about the data
```

Practice

Recode the data to create a new variable from the `y` variable to be values of `dead` (instead of 1) and `living` (instead of 0). Call the variable status.

First change the y variable to be character.

```
rat2 <- rat2 %>% ______(y = __________(y))

rat2_recoded <- rat2 %>%
  _______(status = ______(_, ___ = ______,
                             ___ = ____))
```

Summary

- `recode()` requires `mutate()` when working with dataframes/tibbles
- `recode()` can help with simple recoding (an **exact** swap) for values
- `recode()` has the opposite order as `rename` - use “old value” = “new value”
- `rename()` helps us change column names - use new name = old name and it does not require `mutate()` [Workshop Website]

Extra slides if there is time

`case_when()` helps make sophisticated new variables

Note that automatically values not reassigned explicitly by `case_when()` will be `NA` unless otherwise specified.

```
# General Format - this is not code!
{data_input} %>%
  mutate({new_variable} = case_when({Variable}
              /some condition/ ~ {value_for_con},
                           TRUE ~ {value_for_not_meeting_condition})
```

{value_for_not_meeting_condition} could be something new or it can be the original values of a column

case_when()

`case_when` can do very sophisticated comparisons

```
rat <- rat %>%
  mutate("survival" =
              case_when(num_living_litter > num_dead_litter ~ "well",
                        num_living_litter == num_dead_litter ~ "even",
                        num_living_litter < num_dead_litter ~ "poor"))
```

Now it is easier to see what is happening

```
rat%>%
  count(Grp_recoded, survival)
```

```
        Grp_recoded survival  n
1  Inj. Day 0 and 7     well  5
2 Inj. Day 7 and 10     well 12
3       Inj. Weekly     well 10
4         Untreated     poor 26
5         Untreated     well  5
```

case_when will make NA values

If there is a condition not specified, NA values will be generated.

```
rat %>%
  mutate("survival" =
           case_when(num_living_litter > num_dead_litter ~ "well")) %>%
  count(Grp_recoded, survival)
```

```
        Grp_recoded survival  n
1  Inj. Day 0 and 7     well  5
2 Inj. Day 7 and 10     well 12
3       Inj. Weekly     well 10
4         Untreated     well  5
5         Untreated     <NA> 26
```

Summary

- `recode()` and `case_when()` require `mutate()` when working with dataframes/tibbles
- `recode()` can help with simple recoding (an **exact** swap)
- `case_when()` can recode based on **conditions** (need quotes for conditions and new values)
    - remember `case_when()` will generate `NA` values for anything not specified

Workshop Website