Data Transformation and Exploration

Lecture 03.1: Data Transformation

Lauren Sullivan

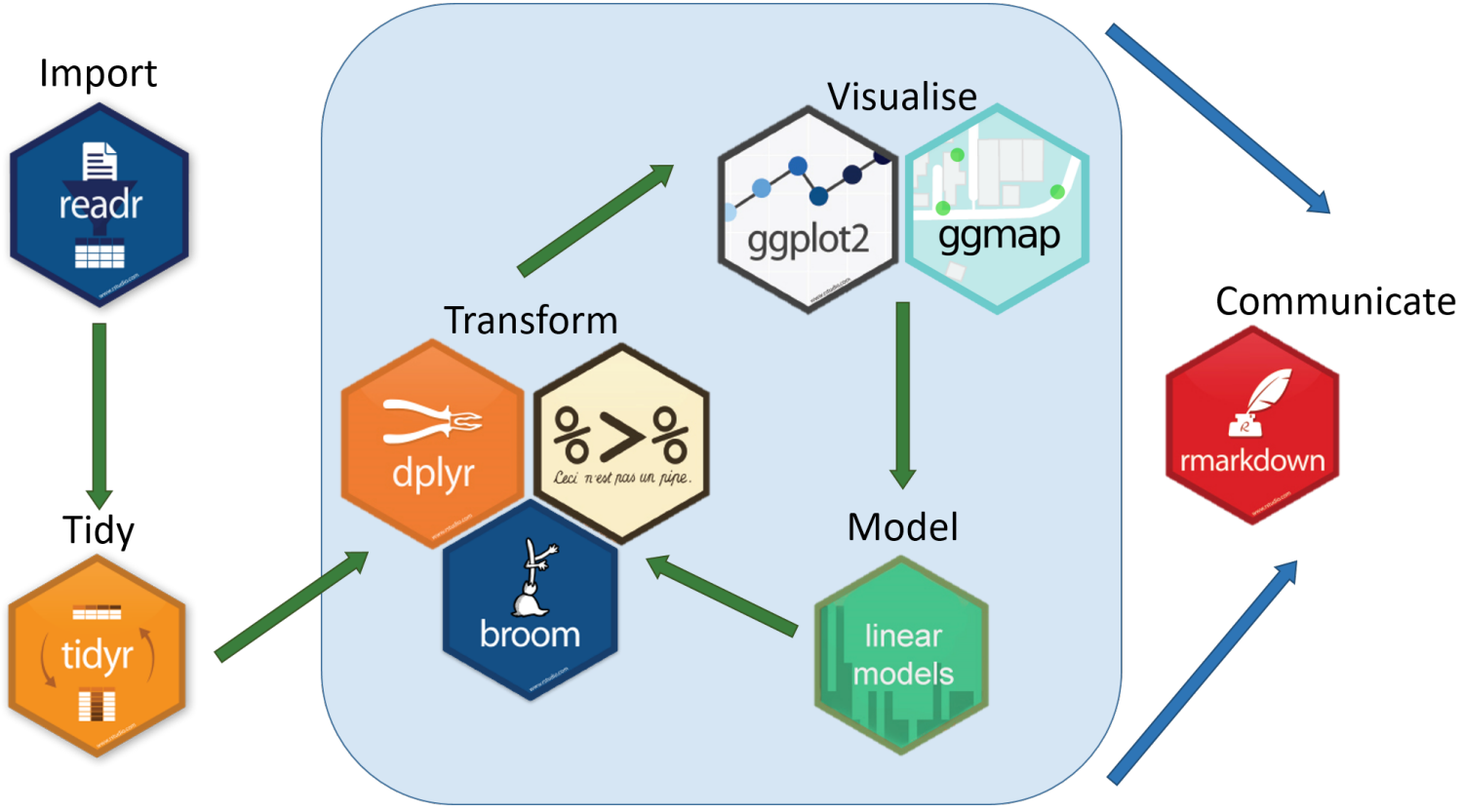
Module: Data Management, Visualization & Reproducibility

Data Manipulation Goals – Reminder

Once we have tidy data, we may need to update its format, create new variables, filter out other variables, etc.

- Perform all manipulation in R
  - Preserves data integrity
  - This will take a lot of time at first but is worth the effort
  - Remember Google is your friend!

Data Transformation with dplyr



Useful transformation functions

- `filter()`
- `arrange()`
- `select()`
- `mutate()` and `transmute()`
- `summarize()`

This lecture we will be using partial data from a study by Cravens and Boyle (2019) on the morphological data of bats.

Oecologia (2019) 189:69–77
https://doi.org/10.1007/s00442-018-4300-6

PHYSIOLOGICAL ECOLOGY – ORIGINAL RESEARCH

CrossMark

Illuminating the physiological implications of artificial light on an insectivorous bat community

Zachary M. Cravens[1] · Justin G. Boyles[1]

Received: 8 March 2018 / Accepted: 8 November 2018 / Published online: 16 November 2018

Eastern Red Bat morphology data

```
bats <- read_csv("../data/bats.csv")
bats[1:10,]
```

```
## # A tibble: 10 x 7
##    age   sex   condition   RFA  mass moonlight avg_temp
##    <chr> <chr> <chr>     <dbl> <dbl>     <dbl>    <dbl>
##  1 A     M     NR         34.5   5          99     23.4
##  2 A     M     NR         41.1  11.3        77     25.4
##  3 A     F     L          42.9  13          77     25.4
##  4 A     M     NR         44.5  12          77     25.4
##  5 A     M     NR         35.7   8.8        85     25.0
##  6 A     F     L          46.6  22          85     25.0
##  7 A     F     L          43.9  12          77     26.5
##  8 A     F     L          40.2  11.5        58     26.3
##  9 A     F     P          33.4  10          58     26.3
## 10 A     F     L          44.6  13.3        58     26.3
```

filter()

Subsets observations based on their values. The first argument is the dataframe to filter, and the following arguments are the expressions for how you want to filter. So say we only want to look at female bats.

```
filter(bats, sex == "F")
```

```
## # A tibble: 87 x 7
##    age   sex   condition   RFA  mass moonlight avg_temp
##    <chr> <chr> <chr>     <dbl> <dbl>     <dbl>    <dbl>
##  1 A     F     L          42.9  13          77     25.4
##  2 A     F     L          46.6  22          85     25.0
##  3 A     F     L          43.9  12          77     26.5
##  4 A     F     L          40.2  11.5        58     26.3
##  5 A     F     P          33.4  10          58     26.3
##  6 A     F     L          44.6  13.3        58     26.3
##  7 A     F     L          42.2  11.8        58     26.3
##  8 A     F     L          39.7  13.8         9     23.3
##  9 A     F     L          36.5  12.5         3     24.9
## 10 A     F     L          30.5  13          10     17.2
## # ... with 77 more rows
```

filter()

Or we only want to look at male bats with a mass above 15g.

```
filter(bats, sex == "M", mass > 15)
```

```
## # A tibble: 14 x 7
##    age   sex   condition   RFA  mass moonlight avg_temp
##    <chr> <chr> <chr>     <dbl> <dbl>     <dbl>    <dbl>
##  1 A     M     NR         46.9  17.5        10     17.2
##  2 A     M     NR         46.2  20          10     17.2
##  3 A     M     NR         46.4  20.5        81     27.3
##  4 J     M     NR         47.3  18.5        94     26.8
##  5 A     M     NR         44    17          50     25.4
##  6 J     M     NR         45.6  17.8        29     26.6
##  7 J     M     TD         46.1  16           4     26.7
##  8 J     M     TD         46.2  15.5         0     24.0
##  9 A     M     NR         45.2  15.5         0     24.0
## 10 A     M     TD         45.5  19           2     23.0
## 11 A     M     NR         45    19.5         2     23.0
## 12 A     M     NR         44.5  16.8         7     28.1
## 13 A     M     TD         46.5  17.8         7     28.1
## 14 A     M     TD         50    18.5        96     19.5
```

filter()

If we want to use the %>% we don't have to specify the dataframe.

```
bats %>% filter(sex == "M", mass > 15)
```

```
## # A tibble: 14 x 7
##    age   sex   condition   RFA  mass moonlight avg_temp
##    <chr> <chr> <chr>     <dbl> <dbl>     <dbl>    <dbl>
##  1 A     M     NR         46.9  17.5        10     17.2
##  2 A     M     NR         46.2  20          10     17.2
##  3 A     M     NR         46.4  20.5        81     27.3
##  4 J     M     NR         47.3  18.5        94     26.8
##  5 A     M     NR         44    17          50     25.4
##  6 J     M     NR         45.6  17.8        29     26.6
##  7 J     M     TD         46.1  16           4     26.7
##  8 J     M     TD         46.2  15.5         0     24.0
##  9 A     M     NR         45.2  15.5         0     24.0
## 10 A     M     TD         45.5  19           2     23.0
## 11 A     M     NR         45    19.5         2     23.0
## 12 A     M     NR         44.5  16.8         7     28.1
## 13 A     M     TD         46.5  17.8         7     28.1
## 14 A     M     TD         50    18.5        96     19.5
```

Logical Operators

You can use `filter()` combined with Boolean operators to select various sets.

- `&` is "and", `|` is "or", `!` is "not"
- Note: `filter()` excludes rows with `NA` for the variables specified.

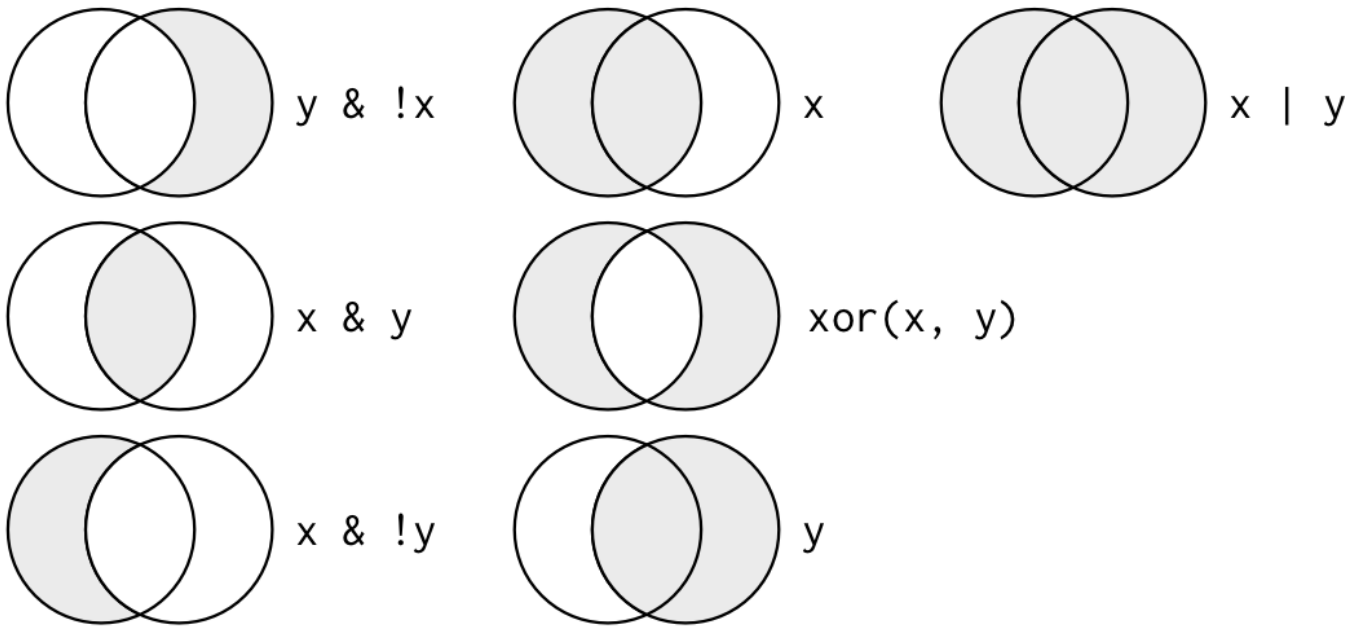


Figure 1: Boolean operators. The left circle is x, the right circle is y, and the shaded region indicates the given subset selected.

`filter()` with Boolean

So say you want the set of male bats with mass > 15g. You can also write this with boolean operators.

```
bats %>% filter(sex == "M" & mass > 15)
```

```
## # A tibble: 14 x 7
##    age   sex   condition   RFA  mass moonlight avg_temp
##    <chr> <chr> <chr>     <dbl> <dbl>     <dbl>    <dbl>
##  1 A     M     NR         46.9  17.5        10     17.2
##  2 A     M     NR         46.2  20          10     17.2
##  3 A     M     NR         46.4  20.5        81     27.3
##  4 J     M     NR         47.3  18.5        94     26.8
##  5 A     M     NR         44    17          50     25.4
##  6 J     M     NR         45.6  17.8        29     26.6
##  7 J     M     TD         46.1  16           4     26.7
##  8 J     M     TD         46.2  15.5         0     24.0
##  9 A     M     NR         45.2  15.5         0     24.0
## 10 A     M     TD         45.5  19           2     23.0
## 11 A     M     NR         45    19.5         2     23.0
## 12 A     M     NR         44.5  16.8         7     28.1
## 13 A     M     TD         46.5  17.8         7     28.1
## 14 A     M     TD         50    18.5        96     19.5
```

`filter()` with Boolean

Or say you want the set of bats that are adult females

```
bats %>% filter(sex == "F" & age == "A")
```

```
## # A tibble: 37 x 7
##    age   sex   condition   RFA  mass moonlight avg_temp
##    <chr> <chr> <chr>     <dbl> <dbl>     <dbl>    <dbl>
##  1 A     F     L          42.9  13          77     25.4
##  2 A     F     L          46.6  22          85     25.0
##  3 A     F     L          43.9  12          77     26.5
##  4 A     F     L          40.2  11.5        58     26.3
##  5 A     F     P          33.4  10          58     26.3
##  6 A     F     L          44.6  13.3        58     26.3
##  7 A     F     L          42.2  11.8        58     26.3
##  8 A     F     L          39.7  13.8         9     23.3
##  9 A     F     L          36.5  12.5         3     24.9
## 10 A     F     L          30.5  13          10     17.2
## # ... with 27 more rows
```

`filter()` with Boolean

Or you want the set of bats that either have condition "NR" or "L"

```
bats %>% filter(condition == "NR" | condition == "L")
```

```
## # A tibble: 129 x 7
##    age   sex   condition   RFA  mass moonlight avg_temp
##    <chr> <chr> <chr>     <dbl> <dbl>     <dbl>    <dbl>
##  1 A     M     NR         34.5   5          99     23.4
##  2 A     M     NR         41.1  11.3        77     25.4
##  3 A     F     L          42.9  13          77     25.4
##  4 A     M     NR         44.5  12          77     25.4
##  5 A     M     NR         35.7   8.8        85     25.0
##  6 A     F     L          46.6  22          85     25.0
##  7 A     F     L          43.9  12          77     26.5
##  8 A     F     L          40.2  11.5        58     26.3
##  9 A     F     L          44.6  13.3        58     26.3
## 10 A     M     NR         36.3   9.5        58     26.3
## # ... with 119 more rows
```

arrange()

Used to sort your tibbles into the orders you want.

- You can arrange by more than one column, and this will help break ties when they occur.

```
bats %>% arrange(avg_temp, condition, RFA)
```

```
## # A tibble: 168 x 7
##    age   sex   condition   RFA  mass moonlight avg_temp
##    <chr> <chr> <chr>     <dbl> <dbl>     <dbl>    <dbl>
##  1 A     F     L          30.5  13          10     17.2
##  2 A     M     NR         46.2  20          10     17.2
##  3 A     M     NR         46.9  17.5        10     17.2
##  4 J     M     TD         39.3  11.8        75     18.6
##  5 J     F     NR         38.9  11.3        50     19
##  6 J     F     NR         39     9.3        50     19
##  7 J     F     NR         40    10          50     19
##  8 J     F     NR         41.2  11          50     19
##  9 J     F     NR         41.3  10.5        50     19
## 10 J     F     NR         41.8  11.3        50     19
## # ... with 158 more rows
```

arrange()

As you may have noticed, `arrange()` sorts by both ascending numerical and alphabetical order, but you can flip this to descending order with `desc()`.

- `arrange()` always puts `NA`'s at the end, no matter if your sort in ascending or descending order.

```
bats %>% arrange(desc(avg_temp), desc(condition), desc(RFA))
```

```
## # A tibble: 168 x 7
##    age   sex   condition   RFA  mass moonlight avg_temp
##    <chr> <chr> <chr>     <dbl> <dbl>     <dbl>    <dbl>
##  1 A     M     TD         46.5  17.8         7     28.1
##  2 J     M     TD         38.5   9.5         7     28.1
##  3 J     M     TD         36.5   9.5         7     28.1
##  4 J     M     TD         35.5   9           7     28.1
##  5 A     M     NR         44.5  16.8         7     28.1
##  6 J     F     NR         42.9  10.3         7     28.1
##  7 J     F     NR         42.6  12           7     28.1
##  8 J     F     NR         42    11.8         7     28.1
##  9 J     F     NR         41.6  12           7     28.1
## 10 J     F     NR         40.4  12           7     28.1
## # ... with 158 more rows
```

select()

If you want to select certain columns, use `select()`. For example, select the columns for sex, moonlight and mass.

```
bats %>% select(sex, moonlight, mass)
```

```
## # A tibble: 168 x 3
##    sex   moonlight  mass
##    <chr>     <dbl> <dbl>
##  1 M            99   5
##  2 M            77  11.3
##  3 F            77  13
##  4 M            77  12
##  5 M            85   8.8
##  6 F            85  22
##  7 F            77  12
##  8 F            58  11.5
##  9 F            58  10
## 10 F            58  13.3
## # ... with 158 more rows
```

mutate() and transmute()

Mutate always adds a new column at the end of the data. So say you want to know the ratio of each bat's right forearm length (RFA) to its bodymass (mass).

```
bats %>% mutate(ratio = RFA / mass)
```

```
## # A tibble: 168 x 8
##    age   sex   condition   RFA  mass moonlight avg_temp ratio
##    <chr> <chr> <chr>     <dbl> <dbl>     <dbl>    <dbl> <dbl>
##  1 A     M     NR         34.5   5          99     23.4  6.9
##  2 A     M     NR         41.1  11.3        77     25.4  3.64
##  3 A     F     L          42.9  13          77     25.4  3.3
##  4 A     M     NR         44.5  12          77     25.4  3.71
##  5 A     M     NR         35.7   8.8        85     25.0  4.06
##  6 A     F     L          46.6  22          85     25.0  2.12
##  7 A     F     L          43.9  12          77     26.5  3.66
##  8 A     F     L          40.2  11.5        58     26.3  3.50
##  9 A     F     P          33.4  10          58     26.3  3.34
## 10 A     F     L          44.6  13.3        58     26.3  3.35
## # ... with 158 more rows
```

mutate() and transmute()

If all you care about is the new variable, use `transmute()`

```
bats %>% transmute(ratio = RFA / mass)
```

```
## # A tibble: 168 x 1
##    ratio
##    <dbl>
##  1  6.9
##  2  3.64
##  3  3.3
##  4  3.71
##  5  4.06
##  6  2.12
##  7  3.66
##  8  3.50
##  9  3.34
## 10  3.35
## # ... with 158 more rows
```

Useful Creation Functions

There are a lot of useful functions for creating new variables out there. I will list some here, but use Google if you are ever looking for something specific.

- Arithmetic operators: `+`, `-`, `*`, `/`, `^`
- Logs: `log()` - which is a natural log, `log2()`, `log10()`
- Offsets: `lead()`, `lag()`
- Cumulative: `cumsum()`, `cumprod()`, ..., etc
- Ranking: `min_rank()`, `row_number()`, `percent_rank()`, ..., etc

summarize()

This is a very useful function. And allows you to collapse parts of dataframes into a single row (so summarize your data based on specifications). So say we want to get an average mass of bats

```
bats %>% summarize(avg_mass = mean(mass))
```

```
## # A tibble: 1 x 1
##   avg_mass
##      <dbl>
## 1     11.7
```

summarize()

If you want to summarize by multiple groups, use the %>%

```
bats %>%
  group_by(sex) %>%
  summarize(avg_mass = mean(mass))
```

```
## # A tibble: 2 x 2
##   sex   avg_mass
##   <chr>    <dbl>
## 1 F         12.2
## 2 M         11.2
```

```
bats %>%
  group_by(sex, age) %>%
  summarize(avg_mass = mean(mass))
```

```
## # A tibble: 4 x 3
## # Groups:   sex [2]
##   sex   age   avg_mass
##   <chr> <chr>    <dbl>
## 1 F     A         14.7
## 2 F     J         10.3
## 3 M     A         13.5
## 4 M     J          9.98
```

summarize()

Here are some of the many functions that can be useful with `summarize()`

- Exploratory statistics: `mean(x)`, `median(x)`, `sd(x)`, `IQR(x)`
- Measures of rank or position: `min(x)`, `quantile(x, c(0.05, 0.95))`, `max(x)`, `first(x)`, `nth(x, 2)`, `last(x)`
- Counts: `n()` - size of current group, `sum(x)` (try `sum(~is.na(x)` to make sure you don't include `NA`'s), `n_distinct(x)` - the number of unique elements

Combine these functions to explore your data!

Summarize by group and arrange by mass

```
bats %>%
  group_by(sex, age, condition) %>%
  summarize(avg_mass = mean(mass)) %>%
  arrange(desc(avg_mass))
```

```
## # A tibble: 9 x 4
## # Groups:   sex, age [4]
##   sex   age   condition avg_mass
##   <chr> <chr> <chr>         <dbl>
## 1 F     A     PL             17.9
## 2 M     A     TD             16.2
## 3 F     A     L              14.2
## 4 M     A     NR             13.1
## 5 F     A     NR             13
## 6 F     J     NR             10.3
## 7 M     J     NR             10.1
## 8 F     A     P              10
## 9 M     J     TD              9.83
```

Combine these functions to explore your data!

Find all groups bigger than a threshold

```
bats %>%
  group_by(sex, age) %>%
  filter(mass > 15)
```

```
## # A tibble: 24 x 7
## # Groups:   sex, age [3]
##    age   sex   condition   RFA  mass moonlight avg_temp
##    <chr> <chr> <chr>     <dbl> <dbl>     <dbl>    <dbl>
##  1 A     F     L          46.6  22          85     25.0
##  2 A     M     NR         46.9  17.5        10     17.2
##  3 A     M     NR         46.2  20          10     17.2
##  4 A     F     L          43.1  16          27     22.8
##  5 A     F     L          41    16          57     21.4
##  6 A     F     L          41.5  15.8        98     24.4
##  7 A     F     L          44    22.5        81     27.3
##  8 A     M     NR         46.4  20.5        81     27.3
##  9 A     F     PL         48.7  21          81     27.3
## 10 A     F     PL         51.9  27          81     27.3
## # ... with 14 more rows
```