Week 4: Aggregation

Ellen Bledsoe

2024-02-06

Aggregation

Setup

First, we need to load out packages that we will be using for our lesson. Again, we will need `readr` and `dplyr`.

```
library(readr)
library(dplyr)
```

```
##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

Next, we read in our surveys data using the `read_csv()` function.

```
surveys <- read_csv("surveys.csv")
```

```
## Rows: 35549 Columns: 9
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr (2): species_id, sex
## dbl (7): record_id, month, day, year, plot_id, hindfoot_length, weight
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

Split, Apply, Combine with `group_by()`

One common way we analyze data is through something we call the "split, apply, combine" approach. This means that we:

- *split* data up into groups via some type of categorization
- *apply* some type of analysis to each group independently and
- *combine* the data back together

The `group_by()` function lets us do this. It is most often used in combination with `mutate()` or `summarize()`.

For example, we can use this method to calculate values for every year from the surveys data frame. In this case, we would group by the year column.

Let's see what happens to the `surveys` dataframe when we group by the year column.

```
surveys %>%
  group_by(year)
```

```
## # A tibble: 35,549 x 9
## # Groups:   year [26]
##    record_id month   day  year plot_id species_id sex   hindfoot_length weight
##        <dbl> <dbl> <dbl> <dbl>   <dbl> <chr>      <chr>           <dbl>  <dbl>
##  1         1     7    16  1977       2 NL         M                  32     NA
##  2         2     7    16  1977       3 NL         M                  33     NA
##  3         3     7    16  1977       2 DM         F                  37     NA
##  4         4     7    16  1977       7 DM         M                  36     NA
##  5         5     7    16  1977       3 DM         M                  35     NA
##  6         6     7    16  1977       1 PF         M                  14     NA
##  7         7     7    16  1977       2 PE         F                  NA     NA
##  8         8     7    16  1977       1 DM         M                  37     NA
##  9         9     7    16  1977       1 DM         F                  34     NA
## 10        10     7    16  1977       6 PF         F                  20     NA
## # i 35,539 more rows
```

The `group_by()` function doesn't seem to change the data frame visually in any way. However, you will notice that next to the information about the tibble (number of rows and columns), there is now an addditional bit of information that tells us that this is now a grouped dataframe: grouped by the year column, and there are 26 groups.

Combining `group_by()` and `summarize()`

Often times, we are interested in calculating summary statistics for our data, such as the mean, standard deviation, minimums, maximums, etc.

Fortunately, the `dplyr` has a handy-dandy function to make this easy to do with data frames. The `summarize()` function creates a new dataframe with columns and values we give it.

Similar to `mutate()`, what is on the left of the `=` is the name of the new column, and what is on the right of the `=` is the value(s) to put in the new column.

```
surveys %>%
  summarise(min_weight = min(weight, na.rm = TRUE))
```

```
## # A tibble: 1 x 1
##   min_weight
##        <dbl>
## 1          4
```

After grouping a data frame, we can pipe it into a `summarize()` function to calculate values for *each group*.

```
surveys %>%
  group_by(year) %>%
  summarise(min_weight = min(weight, na.rm = TRUE))
```

```
## # A tibble: 26 x 2
##     year min_weight
##    <dbl>      <dbl>
##  1  1977          4
##  2  1978          6
##  3  1979          6
##  4  1980          5
##  5  1981          4
##  6  1982          4
##  7  1983          4
##  8  1984          7
##  9  1985          4
## 10  1986          7
## # i 16 more rows
```

As another example, we can use a new function (`n()`), which will count up the number of rows per group. That will give us the number of rodents caught during that year, which we will consider the abundance.

```
surveys %>%
  group_by(year) %>%
  summarise(abundance = n())
```

```
## # A tibble: 26 x 2
##     year abundance
##    <dbl>     <int>
##  1  1977       503
##  2  1978      1048
##  3  1979       719
##  4  1980      1415
##  5  1981      1472
##  6  1982      1978
##  7  1983      1673
##  8  1984       981
##  9  1985      1438
## 10  1986       942
## # i 16 more rows
```

Grouping by Multiple Columns

To calculate the number of individuals caught in each plot for each year, we will want to group by both the year column and the plot_id column.

Let's start by putting only the group by function.

```
surveys %>%
  group_by(year, plot_id)
```

```
## # A tibble: 35,549 x 9
## # Groups:   year, plot_id [622]
##    record_id month   day  year plot_id species_id sex   hindfoot_length weight
##        <dbl> <dbl> <dbl> <dbl>   <dbl> <chr>      <chr>           <dbl>  <dbl>
##  1         1     7    16  1977       2 NL         M                  32     NA
##  2         2     7    16  1977       3 NL         M                  33     NA
##  3         3     7    16  1977       2 DM         F                  37     NA
##  4         4     7    16  1977       7 DM         M                  36     NA
##  5         5     7    16  1977       3 DM         M                  35     NA
##  6         6     7    16  1977       1 PF         M                  14     NA
##  7         7     7    16  1977       2 PE         F                  NA     NA
##  8         8     7    16  1977       1 DM         M                  37     NA
##  9         9     7    16  1977       1 DM         F                  34     NA
## 10        10     7    16  1977       6 PF         F                  20     NA
## # i 35,539 more rows
```

We can see that there are now 622 groups! Let's add our summarize function.

```
surveys %>%
  group_by(year, plot_id) %>%
  summarize(abundance = n())
```

```
## `summarise()` has grouped output by 'year'. You can override using the
## `.groups` argument.
```

```
## # A tibble: 622 x 3
## # Groups:   year [26]
##     year plot_id abundance
##    <dbl>   <dbl>     <int>
##  1  1977       1        22
##  2  1977       2        40
##  3  1977       3        18
##  4  1977       4        22
##  5  1977       5        26
##  6  1977       6        18
##  7  1977       7        12
##  8  1977       8        15
##  9  1977       9        27
## 10  1977      10         7
## # i 612 more rows
```

Let's Practice

Start working on Question 1 and Question 2a-b.

Some Reminders

We can perform multiple calculations within the summarize function.

We'll calculate the number of individuals in each plot year combination and their average weight.

```
surveys %>%
  group_by(year, plot_id) %>%
  summarize(abundance = n(),
            avg_weight = mean(weight))
```

```
## `summarise()` has grouped output by 'year'. You can override using the
## `.groups` argument.
```

```
## # A tibble: 622 x 4
## # Groups:   year [26]
##     year plot_id abundance avg_weight
##    <dbl>   <dbl>     <int>      <dbl>
##  1  1977       1        22         NA
##  2  1977       2        40         NA
##  3  1977       3        18         NA
##  4  1977       4        22         NA
##  5  1977       5        26         NA
##  6  1977       6        18         NA
##  7  1977       7        12         NA
##  8  1977       8        15         NA
##  9  1977       9        27         NA
## 10  1977      10         7         NA
## # i 612 more rows
```

```
# remove NAs
surveys %>%
  group_by(year, plot_id) %>%
  summarize(abundance = n(),
            avg_weight = mean(weight, na.rm = TRUE))
```

```
## `summarise()` has grouped output by 'year'. You can override using the
## `.groups` argument.
```

```
## # A tibble: 622 x 4
## # Groups:   year [26]
##     year plot_id abundance avg_weight
##    <dbl>   <dbl>     <int>      <dbl>
##  1  1977       1        22       37.8
##  2  1977       2        40       39.2
##  3  1977       3        18       29.6
##  4  1977       4        22       60.6
##  5  1977       5        26       58.9
##  6  1977       6        18       38.5
##  7  1977       7        12       33.7
##  8  1977       8        15       54.1
##  9  1977       9        27       55.9
## 10  1977      10         7      NaN
## # i 612 more rows
```

How do we remove the NA values? We need to add the `na.rm = TRUE` argument to the `mean()` function.

You'll note that the data frame till has `NaN`. This is for cases where no individuals in that group have a weight. We can remove those values using `!is.na()`.

```
# remove NAs using filter
surveys %>%
  group_by(year, plot_id) %>%
  summarize(abundance = n(),
            avg_weight = mean(weight, na.rm = TRUE)) %>%
  filter(!is.na(avg_weight))
```

```
## `summarise()` has grouped output by 'year'. You can override using the
## `.groups` argument.
```

```
## # A tibble: 618 x 4
## # Groups:   year [26]
##     year plot_id abundance avg_weight
##    <dbl>   <dbl>     <int>      <dbl>
##  1  1977       1        22       37.8
##  2  1977       2        40       39.2
##  3  1977       3        18       29.6
##  4  1977       4        22       60.6
##  5  1977       5        26       58.9
##  6  1977       6        18       38.5
##  7  1977       7        12       33.7
##  8  1977       8        15       54.1
##  9  1977       9        27       55.9
## 10  1977      11        34       67.6
## # i 608 more rows
```

Note the message about "grouped output." It says that the resulting data frame is grouped by year. When we group by more than one column, the resulting data frame is grouped by all but the last group.

This can be useful in some more complicated circumstances, but it can also make things not work if functions that we want to use later don't support grouped data frames.

If needed, we can remove these groups by adding an `ungroup()` function at the end of our pipeline.

```
surveys %>%
  group_by(plot_id, year) %>%
  summarize(abundance = n(),
            avg_weight = mean(weight, na.rm = TRUE)) %>%
  filter(!is.na(avg_weight)) %>%
  ungroup()
```

```
## `summarise()` has grouped output by 'plot_id'. You can override using the
## `.groups` argument.
```

```
## # A tibble: 618 x 4
##    plot_id  year abundance avg_weight
##      <dbl> <dbl>     <int>      <dbl>
##  1       1  1977        22       37.8
##  2       1  1978        58       84.1
##  3       1  1979        27       76.4
##  4       1  1980        75       75.7
##  5       1  1981        79       79.9
##  6       1  1982       109       63.1
```

```
##  7       1  1983       130       63.8
##  8       1  1984        51       49.3
##  9       1  1985       102       66.4
## 10       1  1986        57       77.9
## # i 608 more rows
```

The message still prints because it happens as part of the summarize step, but looking at the resulting data frame shows us that the final data frame is ungrouped.

Let's Practice!

Try working on Question 2c.

Using `group_by()` with `mutate()`

While we most commonly will use grouping before the summarize function, there are some occassions where using groups with the `mutate()` function can be particularly helpful.

I won't be asking you to do something like this in your assignment, but I at least want you to know that it is possible!

Let's say we want to calculate the relative abundance of each species per year. As a reminder, the relative abundance is the total number of individuals of a species caught divided by the total number of rodents caught that year.

We will want want to calculate (a) the abundance of each species in each year, (b) the total number of rodents caught in that year, and (c) divide them.

```
surveys %>%
  group_by(year, species_id) %>%
  # calculate the total number of individuals per species per year
  summarise(abundance = n()) %>%
  # remove groups based on species_id (leave groups for each year)
  mutate(total_abund = sum(abundance), # total number caught per year
         relative_abund = abundance / total_abund) # relative abundance
```

```
## `summarise()` has grouped output by 'year'. You can override using the
## `.groups` argument.
```

```
## # A tibble: 535 x 5
## # Groups:   year [26]
##     year species_id abundance total_abund relative_abund
##    <dbl> <chr>          <int>       <int>          <dbl>
##  1  1977 DM               264         503         0.525
##  2  1977 DO                12         503         0.0239
##  3  1977 DS                98         503         0.195
##  4  1977 NL                31         503         0.0616
##  5  1977 OL                10         503         0.0199
##  6  1977 OT                17         503         0.0338
##  7  1977 OX                 7         503         0.0139
##  8  1977 PE                 6         503         0.0119
##  9  1977 PF                31         503         0.0616
## 10  1977 PP                 7         503         0.0139
## # i 525 more rows
```

Let's Practice

Keep working on using `group_by()` and `summarise()` together with some other `dplyr` functions. Tackle Question 3.