# EEEB UN3005/GR5005
# Lab - Week 03 - 11 and 13 February 2019

*Xun Zhao, xz2827*

## Data Visualization

### Exercise 1: Importing and Cleaning Snake Capture Data

In class you've already seen the `ebay_snake_captures` dataset which shows snake capture results for approximately a year of sampling at one wetland in South Carolina, Ellenton Bay. You can find this data as a CSV file (`ebay_snake_captures.csv`) on both the class CourseWorks and GitHub site. Download this data, and import it into R, assigning it to an object called `e`. Rename the six columns of this data as follows: "date", "time", "trap_type", "species", "count", "comments". Use `head()` to confirm your change of column names.

```
e = read.csv('ebay_snake_captures.csv')
colnames(e) = c('date', 'time', 'trap_type', 'species', 'count', 'comments')
head(e)
```

```
##         date time  trap_type                species count comments
## 1 2003-03-02 1645 snake trap       Nerodia fasciata     1
## 2 2003-03-05 1400 snake trap    Thamnophis sauritus     1
## 3 2003-03-05 1730 snake trap Agkistrodon piscivorus     1
## 4 2003-03-06 1045 snake trap Agkistrodon piscivorus     1
## 5 2003-03-06 1045 snake trap Agkistrodon piscivorus     1
## 6 2003-03-10 1030 snake trap       Nerodia fasciata     1
```

### Exercise 2: Working With Dates

Dates can be very tricky to work with in R. Think about the general issues we might have. In everyday usage, we sometimes refer to dates using the names of days of the week and months of the year. In other cases, we represent similar data using just numbers (e.g., days 1-31, months 1-12). And in different parts of the world, people use different conventions when writing out dates (e.g., some put the month first, others the day).

Use the function `str()` to examine the structure of the `date` column in the `e` data. How is R currently representing this data?

```
str(e$date)
```

```
##  Factor w/ 195 levels "2003-03-02","2003-03-05",..: 1 2 2 3 3 4 4 5 6 6 ...
```

**Answer:** The R represents these dates as factors. And for every date, it represents a different class (or factor label).

Create a modified date column in `e` called `date_mod` using the following code: `as.Date(as.character(e$date), format = "%d-%b-%y")`. Can you figure out what the `format` argument is doing here? What is the structure of the new `date_mod` column?

```
e$date_mod = as.Date(as.character(e$date), format = "%Y-%m-%d")
str(e$date_mod)
```

```
##  Date[1:457], format: "2003-03-02" "2003-03-05" "2003-03-05" "2003-03-06" "2003-03-06
```

**Answer:** The format is used to tell the function what the date strings looks like, namely, "Day" - "Three Letters Month" - "Two Digit Year".

## Exercise 3: Creating Monthly Summary Capture Counts

Given that the `e` data contains information on snake captures throughout the year, one might naturally be interested in how snake captures vary over time. One sensible way to do this would be to summarize how many of each snake species were captured in a given month. However, right now, our `date_mod` variable represents an even finer scale of date data (i.e., specific days rather than just months).

To get data appropriate for downstream use, first, create a new variable in your `e` data frame called `month_of_capture` that indicates the month associated with a given `date_mod` value using the following code: `as.numeric(format(e$date_mod, "%m"))`. Note, if you plan to work extensively with dates in R, the `lubridate` package has a number of convenient functions for this purpose. The `lubridate` function `month()` would allow us to do this same operation, for example.

Next, create a data frame called `e2` that represents each unique month-snake species combination found within `e` and the associated total capture count (call this variable `monthly_capture_count`).

```
e$month_of_capture = as.numeric(format(e$date_mod, '%m'))
e2 = summarize(group_by(e, month_of_capture, species), monthly_capture_count = n())
e2
```

```
## # A tibble: 86 x 3
## # Groups:   month_of_capture [?]
##    month_of_capture species                monthly_capture_count
##               <dbl> <fct>                                  <int>
##  1                1 Nerodia fasciata                           5
##  2                1 Thamnophis sauritus                        1
##  3                3 Agkistrodon piscivorus                    18
##  4                3 Coluber constrictor                        9
##  5                3 Crotalus horridus                          1
```

```
##  6                3 Farancia erytrogramma                      3
##  7                3 Nerodia erythrogaster                      1
##  8                3 Nerodia fasciata                           3
##  9                3 Thamnophis sauritus                        1
## 10                4 Agkistrodon piscivorus                     5
## # ... with 76 more rows
```

Should `e2` have more or fewer rows of data than `e`? Can you show this is the case? Use some summary functions to investigate `e2` and ensure you have the appropriate dataset for further analyses.

**Answer:** e2 should have fewer rows. The result is shown below.

```
print(ifelse(nrow(e2) > nrow(e), 'e2 has more rows than e', 'e2 has less rows than e'))
```

```
## [1] "e2 has less rows than e"
```

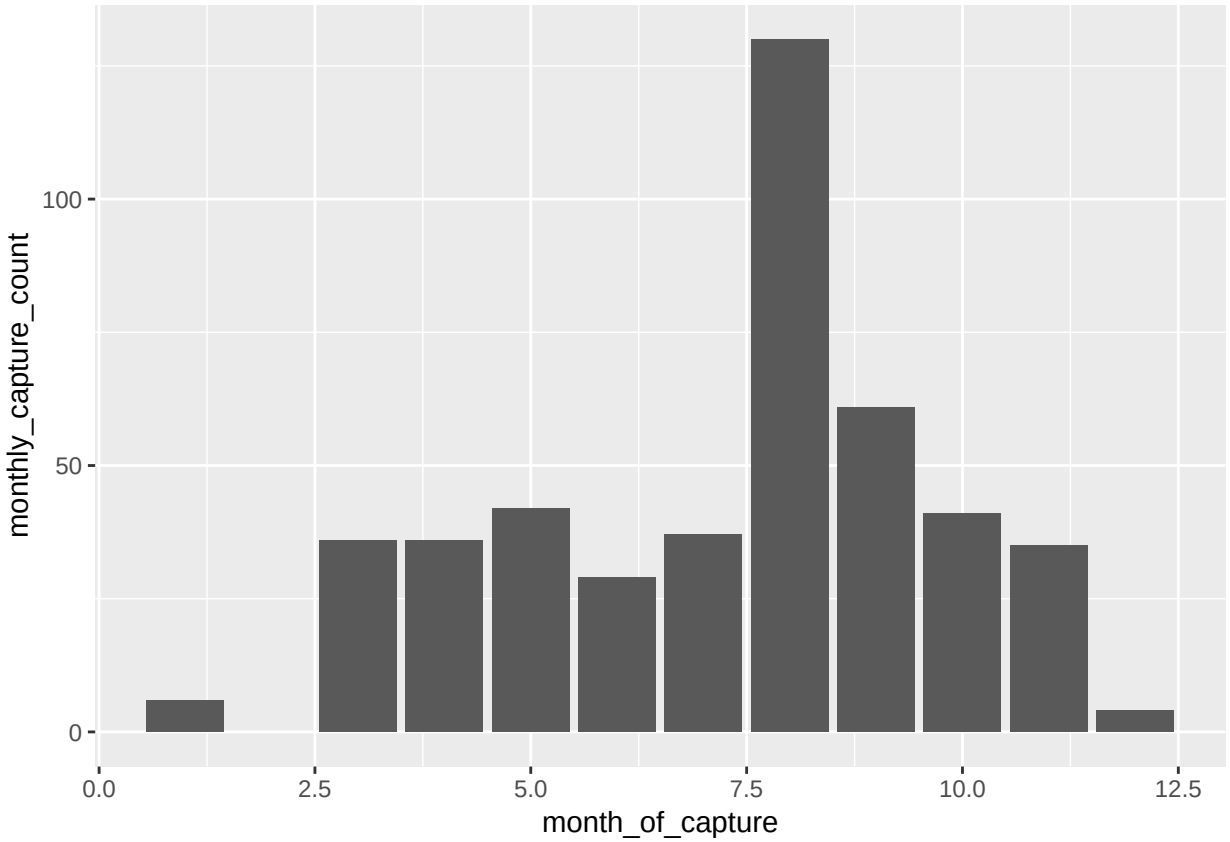
```
str(e2)
```

```
## Classes 'grouped_df', 'tbl_df', 'tbl' and 'data.frame':  86 obs. of  3 variables:
##  $ month_of_capture     : num  1 1 3 3 3 3 3 3 3 4 ...
##  $ species              : Factor w/ 21 levels "Agkistrodon contortrix",..: 12 20 2 4
##  $ monthly_capture_count: int  5 1 18 9 1 3 1 3 1 5 ...
##  - attr(*, "vars")= chr "month_of_capture"
##  - attr(*, "drop")= logi TRUE
```

## Exercise 4: Bar Charts and Histograms

One way you might want to visualize snake captures over time in the `e2` data is with a bar chart. Generate a bar chart using `ggplot()`. The month of capture should appear on the x-axis and capture counts on the y-axis.

```
ggplot(e2, aes(x = month_of_capture, y = monthly_capture_count)) + geom_col()
```




What is the maximum bar height you see displayed? How does this compare with the maximum value of `monthly_capture_count` in the `e2` data? Why?

```
print(list('max of e2:', max(e2$monthly_capture_count)))
```

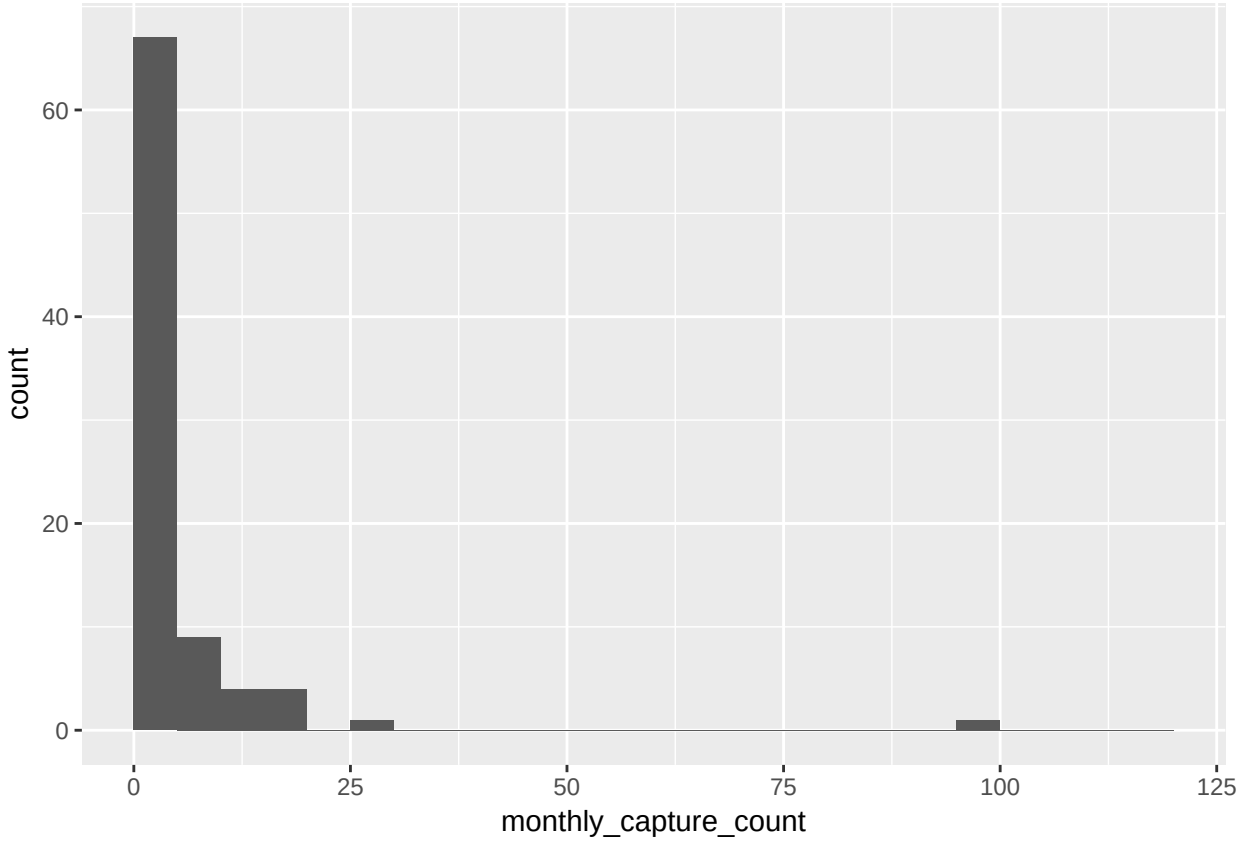
```
## [[1]]
## [1] "max of e2:"
##
## [[2]]
## [1] 100
```

**Answer:** The maximum height is slightly larger than 125, where the max of e2 is 100. It is because the maximum of e2 is specified with month and species, while the every bar in the plot is the sum of all species in a month.

Now, create a histogram of your `monthly_capture_count` variable using `ggplot()` and `geom_histogram()`. Note that `geom_histogram()` does not accept a y-axis variable. You only need to specify the x-axis variable. Within `geom_histogram()`, set the `breaks` argument equal to `seq(from = 0, to = 120, by = 5)` just to create a nicer looking plot.

What is this histogram showing you? What does the y-axis represent? Are there any noticeable outliers in the data?

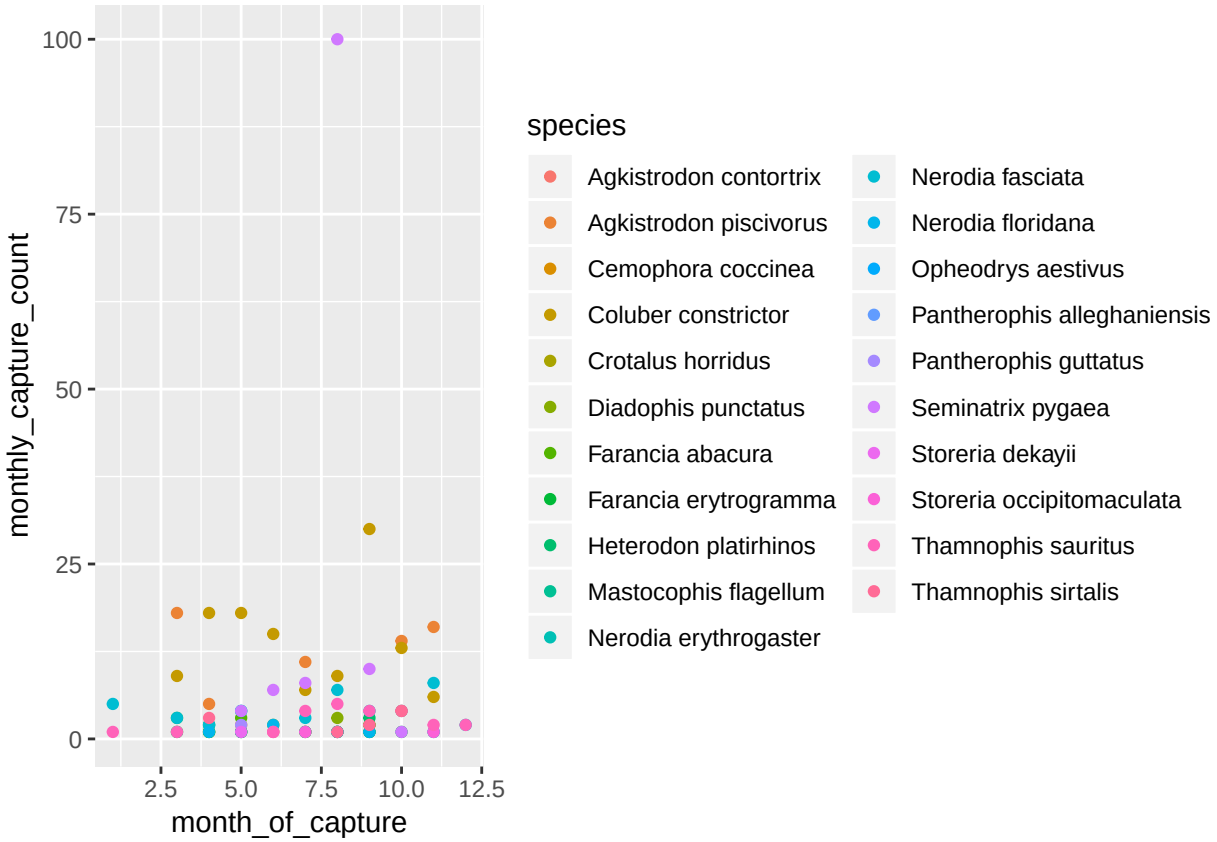
```
ggplot(e2, aes(x = monthly_capture_count)) +
    geom_histogram(breaks = seq(from = 0, to = 120, by = 5))
```




## Exercise 5: Scatter Plots

Now let's examine the `e2` data with scatter plots. Using `ggplot()`, generate a scatter plot of `monthly_capture_count` against `month_of_capture`. Additionally, build the plot such that the color of the data points corresponds to `species`.

```
ggplot(e2, aes(x = month_of_capture, y = monthly_capture_count,
    color = species)) +
    geom_point()
```



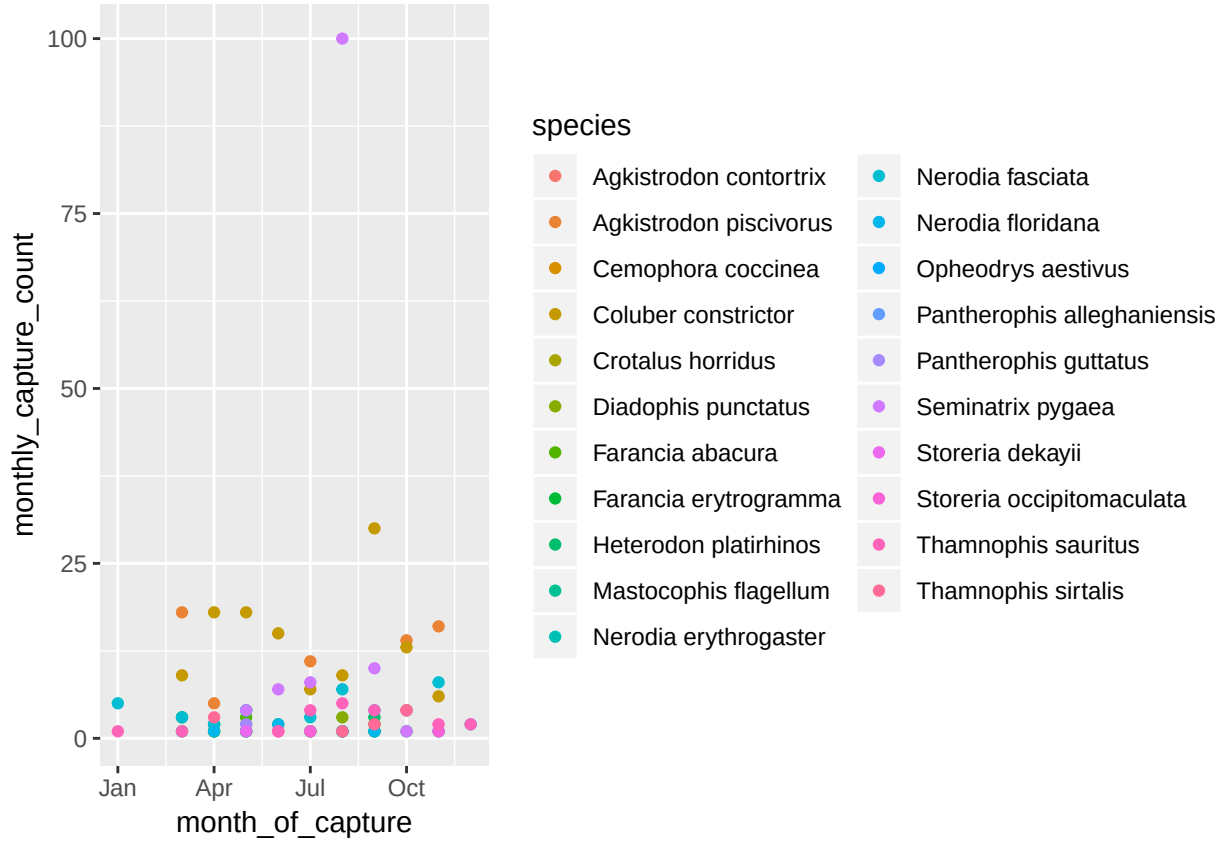

Notice how the x-axis is a little ugly? Let's work step-by-step to make this look better. First, generate a vector named `my.breaks` that contains the numbers 1, 4, 7, and 10. Next, examine the `month.abb` vector that is built into R. Can you generate a vector called `my.labels` that contains the first, fourth, seventh, and tenth elements of `month.abb`?

```
my.breaks = c(1, 4, 7, 10)
my.labels = month.abb[my.breaks]
print(data.frame(my.breaks, my.labels))
```

```
##   my.breaks my.labels
## 1         1       Jan
## 2         4       Apr
## 3         7       Jul
## 4        10       Oct
```

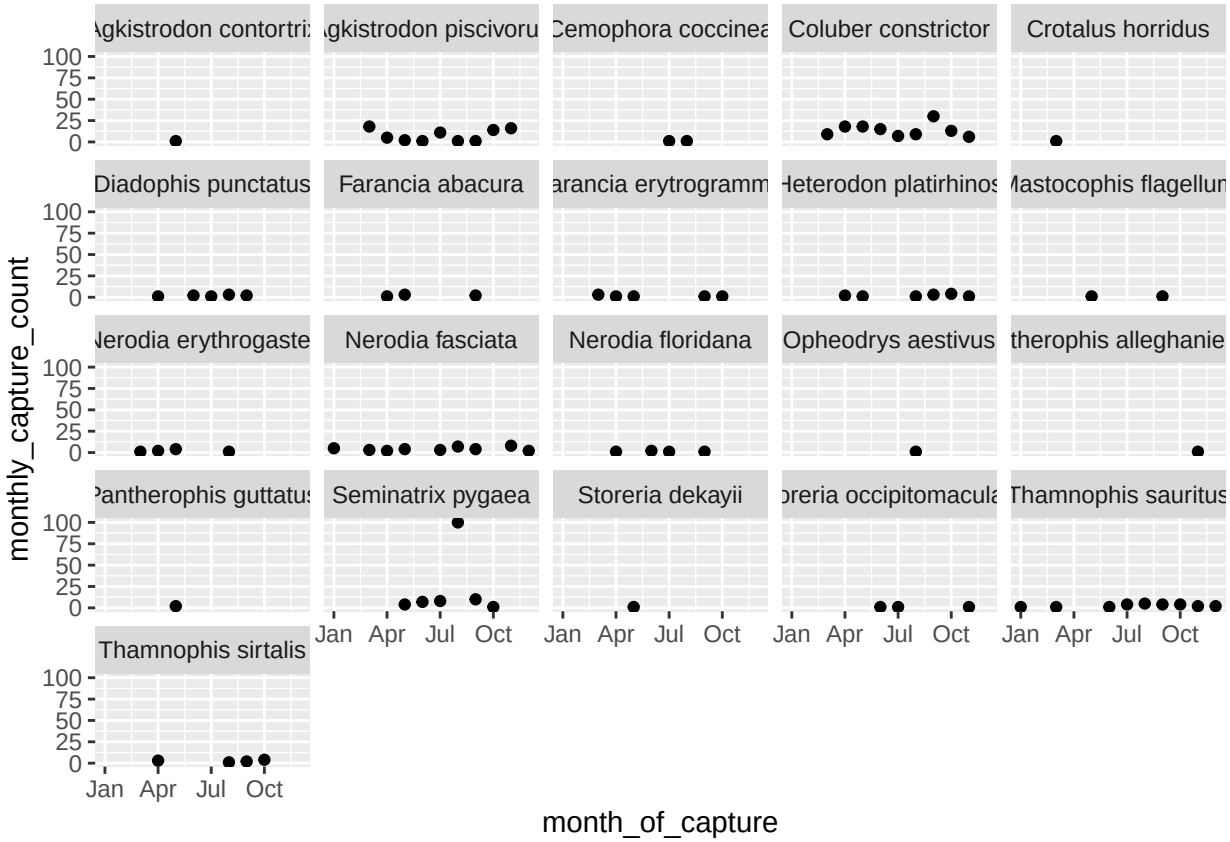
Now that we have these vectors, let's use them to modify the look of our plot. We can specify where on our plot the x-axis labels should appear and what they should be labelled using the layer `scale_x_continuous()`. The relevant arguments are `breaks` (controlling where the x-axis labels land) and `labels` (controlling what the labels read). Regenerate your previous plot but with `scale_x_continuous()` added, with `breaks` equal to `my.breaks` and `labels` equal to `my.labels`.

```
ggplot(e2, aes(x = month_of_capture, y = monthly_capture_count,
    color = species)) +
    geom_point() +
    scale_x_continuous(breaks = my.breaks, labels = my.labels)
```



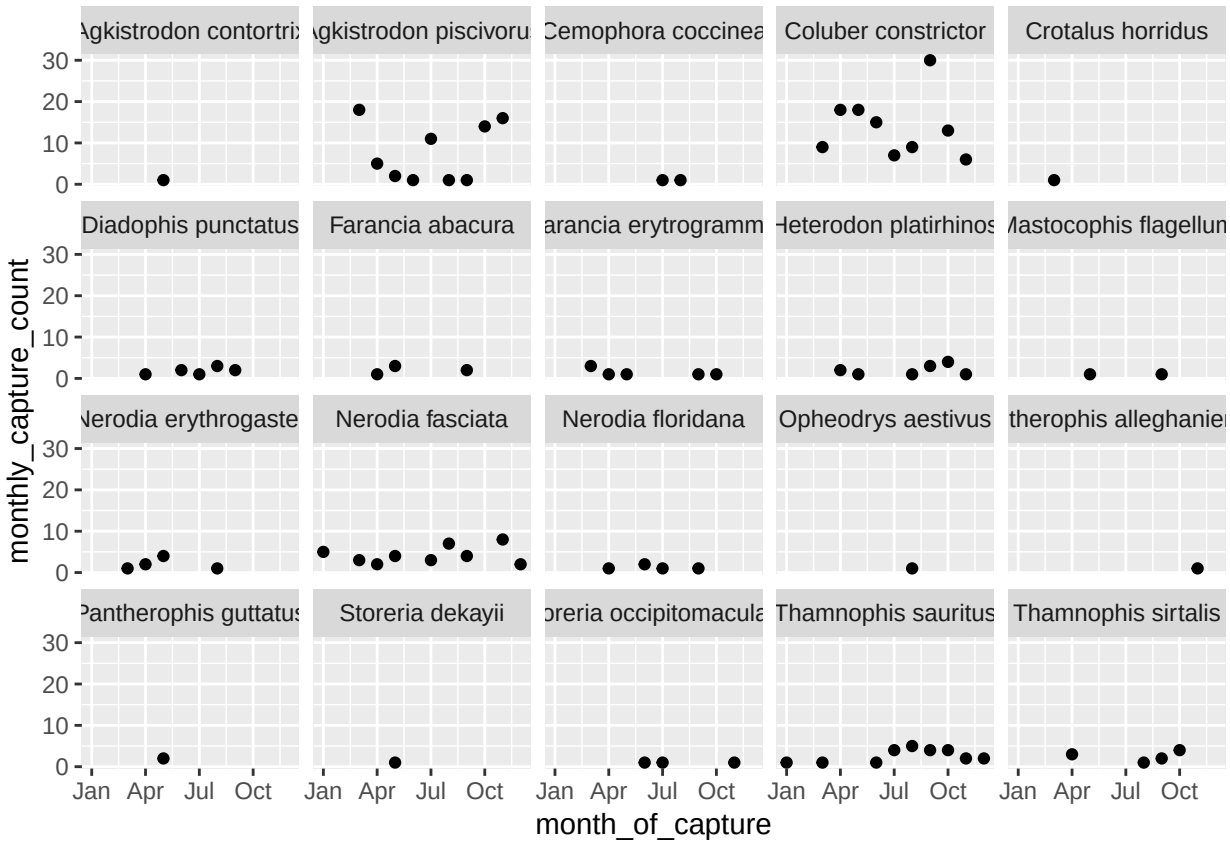

Instead of distinguishing species based on color, create a facetted plot for each species in the dataset.

```
ggplot(e2, aes(x = month_of_capture, y = monthly_capture_count)) +
    geom_point() +
    scale_x_continuous(breaks = my.breaks, labels = my.labels) +
    facet_wrap(~species)
```




Look at the y-axes of the various plots you've produced. By default, `ggplot()` will show all facets with the same y-axis range. However, you can see that one species in the dataset, *Seminatrix pygaea* (check 'em out here) has by far the highest monthly capture count, which means all other species' data is relatively difficult to inspect by comparison. Modify your previous plot to exclude *Seminatrix pygaea* so that any variation in other species' data will be more apparent.

```
e2 %>%
filter(species != 'Seminatrix pygaea') %>%
ggplot(aes(x = month_of_capture, y = monthly_capture_count)) +
    geom_point() +
    scale_x_continuous(breaks = my.breaks, labels = my.labels) +
    facet_wrap(~species)
```




## Bonus Exercise: Install `rethinking`

If you have not yet installed the `rethinking` package, now would be a good time to try to do so, using the instructions at https://github.com/rmcelreath/rethinking.