HW 13: Strings and things

Graphical Analysis of Biological Data

By the end of this assignment, you should be able to achieve the following tasks in R:

- use R notebooks and R markdown;
- insert, write, and evaluate code chucks;
- use pipes;
- manipulate strings using string functions,
- use regular expressions to work with complex strings, and
- produce plots with `ggplot2`;
- customize plots to improve visualization.
- use a typical workflow to wrangle and plot data;
- confidently stage, commit, and push with Git.

These achievements belong to Learning Outcomes 2, 3, 4, 5, 6.

Click on any blue text to visit the external website.

This assignment has four parts.

Note: If you cannot get your code to run, open an issue in the public discussion forum,, and describe the problem. Include the code that is not working and also tell me what you have tried.

Preparation

- Read R4ds Chapter 14: Strings sections 1-4 for reference. You are not required to run the examples or answer questions. **BUT!** I suggest you at least run the examples and answer the questions for section 14.3 and section 14.4. You will be glad you did when it comes time to work with regex patterns.
- Read R4ds Chapter 12.4.1: Separate for details of separating a single column into multiple columns.
- Read R4ds Chapter 13.4: Mutating joins for details of how to merge two tibbles. You will use an inner join later.
- Read R4ds Chapter 28: Graphics for communication for reference.
- Open your `.Rproj` project file in RStudio.
- Right-click and save these files to your `data` folder.
  - beta_gliadin.txt
  - ants.csv
  - flower_size.csv
  - flower_pollinators.csv
- Create an `hw13` folder inside the same folder as your project file.
- Add the YAML header as usual.
- Install the `ggthemes` and `gghighlight` packages. Do I have to tell you to not include the code in your notebook?
- Load the `tidyverse`, `here`, `ggthemes`, and `gghighlight` libraries.

- Install the `RColorBrewer` and `viridis` packages. *Do not load them.* `ggplot` will use these for some color palettes.
- Be sure your code format follows the guidelines.
- Commit early. Commit often. Push regularly.
- Stringr cheatsheet
- Regular expression cheatsheet
- Data visualization cheatsheet I've linked to this cheatsheet before.

Save the results from step to step except where noted.

Part 1: DNA strings

Gluten is a wheat protein that gives bread its structure. Gluten is composed of two smaller classes of proteins, glutenins and gliadins. Gliadins are a primary trigger of Celiac's Disease. For this assignment, you will import a text file with DNA sequence from β-gliadin Genbank Accession K03075.1. The sequence contains the entire gene plus the flanking regions up- and downstream of the gene.

You will use `stringr` functions and regular expressions (regex) to identify the promotor region just upstream from the start of the gene.

Import the data

- Import `beta_gliadin.txt` using the `read_lines()` function. Skip the first line. `read_lines` just reads in the raw text. Each line is treated as a distinct string.
- Print the imported data to get a sense of the format. Here are the first six lines. Yours should look the same.

```
## [1] "\t\t1 aatttgacat gcacagctcc tgaccctgcc aatattgttg cagctgcgct cgcaagcctt"
## [2] "       61 tgcgtagatg atcactttat atgatttgtg taaaaccaaa ataagatcta caaacgaata"
## [3] "      121 gaagctagag cgtaccttgg cgtgcacaca cattgcaagc catacctaac cttgataagt"
## [4] "      181 gttaatgact tgtacaacat atacatcact taagacaagt aaaagcgatt tgatgagtca"
## [5] "      241 tggtctatca aagcaagcca cattactagt ctaatccatc ttaacaggtc acgcatgatt"
## [6] "      301 acaatcttgt ttgtgtgcaa gtcaagccta tctagtttac acgtaacaac ttgtaagaac"
```

From many strings to one

Each line of the text is a single string. Each string contains the DNA sequence and unwanted spaces, tabs (`\t`) and numbers. The unwanted characters will be removed by the next several steps, which you can do in any order. You will need to use `str_replace_all` and regular expressions. The stringr and regex cheatsheets show some helpful shortcuts to represent whitespace and digits. So does the text.

- Replace all of the whitespace with nothing.
- Replace all of the digits with nothing.
- Print the results. You should have 56 strings. Here are the first six

```
## [1] "aatttgacatgcacagctcctgaccctgccaatattgttgcagctgcgctcgcaagcctt"
## [2] "tgcgtagatgatcactttatatgatttgtgtaaaaccaaaataagatctacaaacgaata"
## [3] "gaagctagagcgtaccttggcgtgcacacacattgcaagccatacctaaccttgataagt"
## [4] "gttaatgacttgtacaacatatacatcacttaagacaagtaaaagcgatttgatgagtca"
```

```
## [5] "tggtctatcaaagcaagccacattactagtctaatccatcttaacaggtcacgcatgatt"
## [6] "acaatcttgtttgtgtgcaagtcaagcctatctagtttacacgtaacaacttgtaagaac"
```

- Use `str_c` to *collapse* the strings. Print the variable to ensure you have just one string. Here are the first 100 characters.

```
## [1] "aatttgacatgcacagctcctgaccctgccaatattgttgcagctgcgctcgcaagcctttgcgtagatgatcactttatatgatttgtgtaaaa
```

This would also be a good time to stage and commit. And push it.

DNA sequence is typically represented in upper case letters.

- Convert it to upper case with `str_to_upper()`.
- `str_length()` returns the length of a string. How long is the total DNA sequence? You should get 3310 characters. If not, try to figure out where you went wrong.

Do not save the results of any string length calculations.

Sequence analysis: find the promoter region

These steps will identify the part of the promoter region immediately upstream of the start codon (ATG) of the β-gliadin gene.

DNA consists of four nucleotides, A, C, G, and T. However, the nucleotides in a sequence cannot always be identified positively. Nucleotides that could not be identified are represented by other letters, standardize by the IUPAC. The letters most commonly used are Y (pyrimidines, C and T), R (purines, A and G), and N (any of the four DNA nucleotides), but there are others, as shown in the list linked above.

- Use `str_count()` and regex to count the number of IUPAC letters in the sequence that are *not* A, C, G, or T. *Hint:* I just gave you one.

Do not save the result of any string counts.

The start codon for nuclear genes is `ATG` and the stop codon is either `TGA` or `TAA`.

- Count how many possible start codons are in the sequence.
- Count how many possible stop codons are located in the sequence. For full credit, I'll be more impressed if you search for both stop codons with a single regular expression. *Hint:* `|`

Well, isn't that special? The DNA sequence contains 66 start codons and 99 stop codons. That's too many to work with directly.

- Stage, commit and push.

Let's find the promotor region that is just upstream from the true start codon. The promoter region for many eukaryotic genes have two distinctive motifs, or short sequences that are important for transcription. The first motif is the "CAAT box" and has a sequence of CCAAT. The second motif is the "TATA box" and has a sequence of CTATAA. [1]

- Assign the two motif sequences to variables `caat_box` and `tata_box`, respectively.
- Count how many times each of these motifs occurs in the DNA sequence.

The CAAT box occurs 4 times. The TATA box occurs 1 time. The TATA box should be associated with one of the CAAT boxes.

Comparisons of multiple nuclear genes have shown that the CAAT box is usually within 100-150 nucleotides of the start codon. The TATA box tends to be within 50-100 nucleotides of the start codon (at least in gliadin genes).

[1] The motifs are slightly variable among eukaryotes but these are the specific sequences in this promoter region.

- Make a regex search string called `caat_seq` that begins with "CCAAT", ends with "ATG"", and allows for a stretch of 100-150 nucleotides between the two. *Hint:* Remember `{start, stop}`.
- Count how many times this possible sequence occurs.

Cool. You cut your candidate CAAT boxes in half.

- Make a regex search string called `tata_seq` that begins with "CTATAA", ends with "ATG", and allows for a stretch of 50-100 nucleotides between the two. *Think about this carefully.*
- Count how many times this possible sequence occurs.

Your searches found twice as many `caat_seq` as `tata_seq` in the DNA sequence. The next step is to determine which, if any, `caat_seq` sequences have a `tata_seq` sequence.

- Use `str_extract_all` to extract all of the `caat_seq` sequences and save them in a variable called `caat_seqs_all`. Use the `simplify = TRUE` argument in your function call (`?str_extract_all` for help).
- How long is each extracted sequence? Make a mental note of the length of each one. You should have two sequences that differ in length by nine nucleotides.

```
## [1] 139 148
```

- Use `str_which()` to find which `caat_seq` has the `tata_box` sequence. Use that value to extract the full promotor sequence into a variable called `promotor`.
- Print `promotor` and determine it's length.

```
## [1] "CCAATTGTGAAAGAGATCATGCCATGACAGCTATAAATAGGCCCGCATCGATGATGATCATCCTTCCTCATCCATCATTCTCATAAGTAGAGTGC
```

It should be the sequence that matches that shown and is 139 nucleotides long.

- Stage, commit, and push.

For fun, compare your promoter sequence to the GenBank sequence. Scroll down to "FEATURES", and then click on the CDS link. The last three nucleotides of your sequence are the first three nucleotides highlighted. If you start working upstream (to the left), you will find your promoter sequence.

Part 2: Ants

Wood and her students (unpubl.) studied how managed habitat burn strategies affected ant communities. The first steps of the analyses involved gathering the data and shortening the species names for plotting.

The purpose here is two-fold. First, I want to show you a different way to identify columns for gathering. You *could* hard-code the column names but that is not efficient. What if species were forgotten? Or deleted? What if names changed? These and more are all possible.

While no code is perfect, regular expression are a powerful tool that allows your code to work under the greatest range of circumstances.

Import the data

- Import `ants.csv` and print the first few rows.

```
## # A tibble: 6 x 27
##    year season seas.code trt   plot  sample Aphaenogaster.t~ Brachymyrmex.de~
##   <dbl> <chr>      <dbl> <chr> <chr>  <dbl>            <dbl>            <dbl>
## 1  2010 May            1 Pre-~ 6A         1                0                1
```

```
## 2  2010 May            1 Pre-~ 6A         2                0                0
## 3  2010 May            1 Pre-~ 6A         3                0                0
## 4  2010 May            1 Pre-~ 6A         4                0                0
## 5  2010 May            1 Pre-~ 6A         5                0                1
## 6  2010 May            1 Pre-~ 6A         6                0                0
## # ... with 19 more variables: Camponotus.castaneus <dbl>,
## #   Crematogaster.cerasi <dbl>, Crematogaster.lineolata <dbl>,
## #   Crematogaster.missuriensis <dbl>, Dolichoderus.plagiatus <dbl>,
## #   Dorymyrmex.flavus <dbl>, Forelius.pruinosus <dbl>, Formica.dolosa <dbl>,
## #   Hypoponera.opacior <dbl>, Monomorium.minimum <dbl>,
## #   Neivamyrmex.nigrescens <dbl>, Neivamyrmex.opacithorax <dbl>,
## #   Nylanderia.terricola <dbl>, Nylanderia.trageri <dbl>,
## #   Pheidole.morrisi <dbl>, Strumigenys.louisianae <dbl>,
## #   Tapimoma.sessile <dbl>, Temnothorax.pergandei <dbl>,
## #   Trachymyrmex.septentrionalis <dbl>
```

The first few columns reflect the experimental design. Ants were sampled during May, August, and September of 2010 and 2011. Multiple samples were taken from each of several plots. The abundance of each ant species is recorded in a separate column.

The species column names follow a distinct pattern: "Genus.species". The first letter of genus is capitalized and separated from the specific epithet by a period. The `seas.code` column has a period but it does not begin with a capital letter.

Wrangle the data

The data are untidy, so pivot the data to a longer format. Your `pivot_longer()` statement should follow this framework:

```
pivot_longer(names_to = "species", values_to = "abundance", matches(regex, ignore.case = FALSE))
```

I have given you everything you need except the `regex` pattern. You have to figure out the pattern to select all of the species columns. *Remember to use two backslashes to escape the period that separates Genus and species in the column name.*

```
## # A tibble: 6 x 8
##    year season seas.code trt       plot  sample species                abundance
##   <dbl> <chr>      <dbl> <chr>     <chr>  <dbl> <chr>                      <dbl>
## 1  2010 May            1 Pre-burn~ 6A         1 Aphaenogaster.treatae          0
## 2  2010 May            1 Pre-burn~ 6A         1 Brachymyrmex.depilis           1
## 3  2010 May            1 Pre-burn~ 6A         1 Camponotus.castaneus           0
## 4  2010 May            1 Pre-burn~ 6A         1 Crematogaster.cerasi           0
## 5  2010 May            1 Pre-burn~ 6A         1 Crematogaster.lineola~         2
## 6  2010 May            1 Pre-burn~ 6A         1 Crematogaster.missuri~         2
```

Optional: Use `select()` to remove `season` through `sample` columns. You need only year, species, and abundance.

This part might be the hardest part of the assignment. Break it down into small chunks. Run the code without saving the results until you are satisified you have the correct results. Then, save the results to a variable.

Later, you will plot abundances for each species. The species names are long, which will make the plot awkward. You will use `str_replace` and regex patterns to reduce the long species names to eight letter CEP names. The CEP name is made from the first four letters of the genus and the first four letters of the species. For example, *Mephitis mephitis* becomes Mephmeph.

The `mutate()` framework to use is

```
mutate(species = str_replace(species, pattern, replacement))
```

Use `(...)` to capture and backreference groups. You need to capture two patterns, one for the first four letters of the genus and one for the first four letters of the species. Use `{n}` to limit your matches.[2]

For the genus, you want to *capture* a pattern in parentheses that starts with an uppercase letter followed by up to 3 lowercase letters. For the species you want to *capture* a pattern of the first four lower case letters.

```
## # A tibble: 6 x 3
##    year species  abundance
##   <dbl> <chr>        <dbl>
## 1  2010 Aphatrea         0
## 2  2010 Bracdepi         1
## 3  2010 Campcast         0
## 4  2010 Cremcera         0
## 5  2010 Cremline         2
## 6  2010 Cremmiss         2
```

- Use mutate to turn the year into an order factor with levels "2010" and "2011". Then, group by year and species, and summarize the total abundance of each species. `sum()` will add up the abundance of each species.

Plot the data

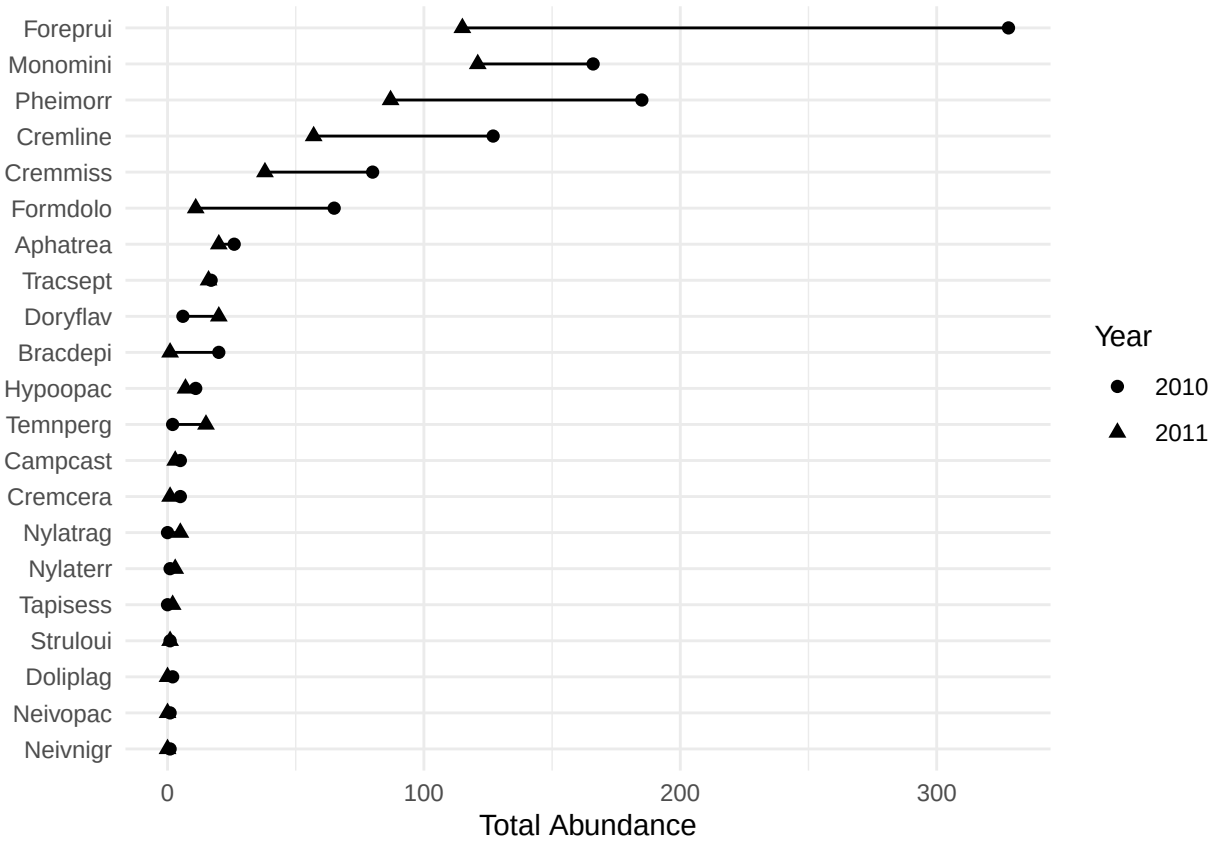
Ants were sampled during 2010 and 2011. When data are sampled across time, you should plot your data over time to look for changes. Here, you will plot the total abundance of ants for each year. I hope you thought immediately of producing a dot plot. You probably thought about using `facet_wrap` on year to produce two plots. That would work but we can do something better.

We will make a Cleveland dot plot of total abundance of each species, similar to the top figure shown here.. Build the plot from your summary data using the following guide.

- Use the `aes` aesthetic in `ggplot` for mapping. Use `x = reorder(species, total)` to sort the species order from most to least abundant. `total` should be on the Y-axis.
- This `aes` should also include `group = species`. This will keep the abundance of each year for each species grouped togehter.
- Add a line geometry layer. You do not need to add aesthetics or anything else to this layer. You may optionally add `color = "gray"` but do not put it inside an aesthetic.
- Add a point geometry layer. Add `shape = f_year` in the `aes` aesthetic for this layer. (Use the column name you used for the ordered years; I used `f_year`). Set the point size to 2, but do this outside of the aesthetic. This will use a different shape for each year and enlarge the slightly. *Ignore the warning.*
- Add a `coord_flip()` layer.
- Add a `labs` layer that removes the x-axis label and changes the `y-axis` label to "Total Abundance". Also add `shape = "Year"`, which will change the title of the legend.
- Add a `theme_minimal()` layer. This simplifies the plot.

Your plot should look like this.

[2]Ideally, you want to use {1,3} or {1,4} to match genera or species names that are fewer than four characters but that is not an issue here so {3} or {4} works fine. Yes, this is a hint for you, but is not *entirely* complete for the problem at hand.

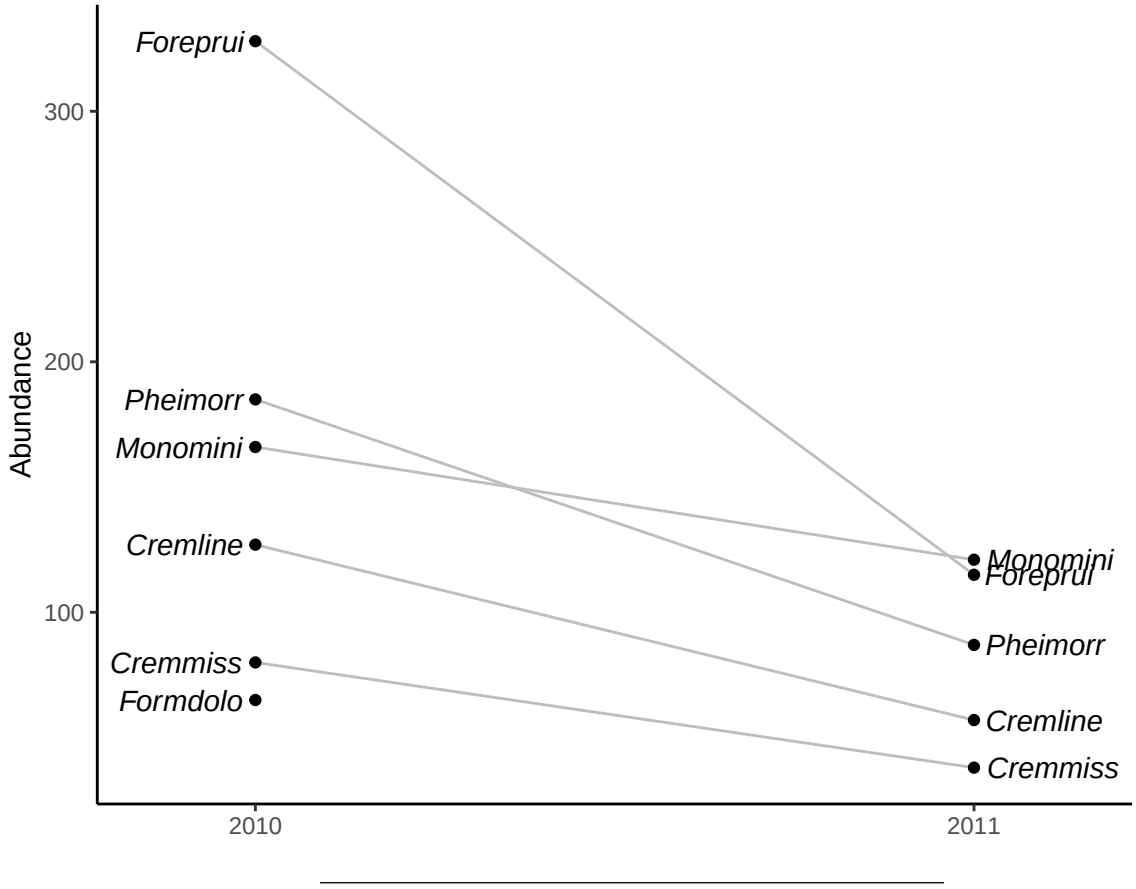


Cleveland plots are, in my opinion, one of the first types of plots that should be considered for data visualization. In this case, though, because we plotted all 21 species, the lower right corner of the plot is a big empty space. The code below shows you how to make a slope graph. Slope graphs are great a showing changes for many variables over distinct time periods or other categories. In the example here, I (arbitrarily) removed species with fewer than 30 total individuals. The plot still needs tweaks but it gives you a sense of what slope graphs can do. Notice how it leaves less empty space compared to the Cleveland plot.

You do not have to reproduce this code but I encourage you to think about what is happening at each step.

```
# Code for slope graph
ants_summary %>%
  filter(total >= 30) %>%  # Arbitrary value
  ggplot(aes(x = f_year, y = total, group = species)) +
  geom_line(color = "gray") +
  geom_point() +
  theme_classic() +
  geom_text(data = ants_summary %>%
              filter(f_year == "2010",
                     total >= 30),
            aes(x = f_year,
                y = total,
                label = species,
                fontface = "italic"),
            hjust = 1.1) +
  geom_text(data = ants_summary %>%
              filter(f_year == "2011",
                     total >= 30),
```

```
            aes(x = f_year,
                y = total,
                label = species,
                fontface = "italic"),
            hjust = -0.1) +
  labs(x = NULL,
       y = "Abundance") +
  theme(legend.position = "none") +
  scale_x_discrete(expand = c(0.2, 0.02))
```



Part 3: Featuring Phlox Flowers

Landis et al. 2018 studied how the interaction of flower traits and pollinators influenced the evolution of flowering plants in the family Polomoniaceae (phloxes).

For this part, you will import and wrangle to separate files, and then join them together into a single data file for plotting.

Import and wrangle the first data set

- Import `flower_size.csv`
- Filter out rows where `Flower number` is not `NA`.

- Select the "Species", "Corolla length (cm)", and "Corolla width throat (cm)" columns. Make the column names syntactic, either by renaming them as part of the `select()` function (best) or by using `rename()` (acceptable).

```
## # A tibble: 6 x 3
##   species               cor_length throat_width
##   <chr>                      <dbl>        <dbl>
## 1 Acanthogilia gloriosa       3.55        0.393
## 2 Acanthogilia gloriosa       3.70        0.455
## 3 Acanthogilia gloriosa       2.66        0.37
## 4 Acanthogilia gloriosa       2.87        0.375
## 5 Aliciella caespitosa        1.65        0.241
## 6 Aliciella caespitosa        1.66        0.148
```

- Use `separate` to separate the species column into `genus` and `species` columns, dropping everything else. Notice I used lower case letters for `genus` and `species` columns. I'll do that again on the second data set.

```
## # A tibble: 6 x 4
##   genus        species    cor_length throat_width
##   <chr>        <chr>           <dbl>        <dbl>
## 1 Acanthogilia gloriosa         3.55        0.393
## 2 Acanthogilia gloriosa         3.70        0.455
## 3 Acanthogilia gloriosa         2.66        0.37
## 4 Acanthogilia gloriosa         2.87        0.375
## 5 Aliciella    caespitosa       1.65        0.241
## 6 Aliciella    caespitosa       1.66        0.148
```

- Group the data by genus and species, and then use `summarize` to summarize the `mean()` and `max()` lengths and widths for each species.

```
## Warning in max(throat_width, na.rm = TRUE): no non-missing arguments to max;
## returning -Inf
```

```
## # A tibble: 6 x 6
## # Groups:   genus [2]
##   genus        species        mean_length mean_width max_length max_width
##   <chr>        <chr>                <dbl>      <dbl>      <dbl>     <dbl>
## 1 Acanthogilia gloriosa              3.20       0.398      3.70      0.455
## 2 Aliciella    caespitosa            1.73       0.197      1.88      0.241
## 3 Aliciella    formosa               2.12       0.297      2.24      0.386
## 4 Aliciella    haydenii              1.58       0.209      2.21      0.321
## 5 Aliciella    heterostyla           1.11       0.145      1.47      0.184
## 6 Aliciella    hutchinsifolia        0.818      0.158      0.957     0.272
```

Import and wrangle the second data set.

The second data set lists many of the same species as the first data set. Where known, flower color, and pollination syndrome are include. The syndrome tells whether the species is self-pollinating ("autogamous") or is pollinated by an animal (bees, hummingbirds, etc.) are included. For this analysis, you will use the only the `Species` and `Pollinator` columns.

- Import `flower_pollinators.csv` into a variable called `pollinators_raw`.
- Use `select()` to remove columns that start with "Source" and to remove the `Color` column.
- Use `filter()` to remove rows where `Pollinator` is `NA`.

- Use `separate()` as you did above to separate `Species` into `genus` and `species` columns. Again, I used lower case letters for `genus` and `species`.
- Save these last three steps in a variable called `pollinators`.

```
## # A tibble: 6 x 3
##   genus        species        Pollinator
##   <chr>        <chr>          <chr>
## 1 Acanthogilia gloriosa       hummingbird
## 2 Aliciella    caespitosa     hummingbird
## 3 Aliciella    hutchinsifolia autogamous
## 4 Aliciella    leptomeria     autogamous
## 5 Aliciella    micromeria     autogamous
## 6 Aliciella    pinnatifida    bee
```

Some species of flowers have multiple pollinators. More than one pollinator may be listed, in the form of "bee, beefly, butterfly" or "bees primary, hawkmoths secondary" or similar. We will assume that the pollinator listed first is the major pollinator and discard the others. The first pollinator is separated from other pollinators by either a comma or a space.

- Use `separate()` to separate the first pollinator from all other words. The `sep` argument must be a regex pattern that finds either a comma or a space. Save the result in a column called `Syndrome`.

```
## # A tibble: 6 x 3
##   genus        species        Syndrome
##   <chr>        <chr>          <chr>
## 1 Acanthogilia gloriosa       hummingbird
## 2 Aliciella    caespitosa     hummingbird
## 3 Aliciella    hutchinsifolia autogamous
## 4 Aliciella    leptomeria     autogamous
## 5 Aliciella    micromeria     autogamous
## 6 Aliciella    pinnatifida    bee
```

Join the datasets

You have two separate tibbles. The `flower_size_summary` tibble has 6 rows. The pollinators tibble has 3 rows. We want to make a `phlox` data set using an `inner_join` so that only species found in the both data sets are included. Species that do not have pollinator data will not be included.

- Use `inner_join` to join the smaller data set to the larger set. As long as your column names for genus and species are identical in both tibbles (e.g., `genus` and `species`), the join function will automatically match both columns when merging the data.

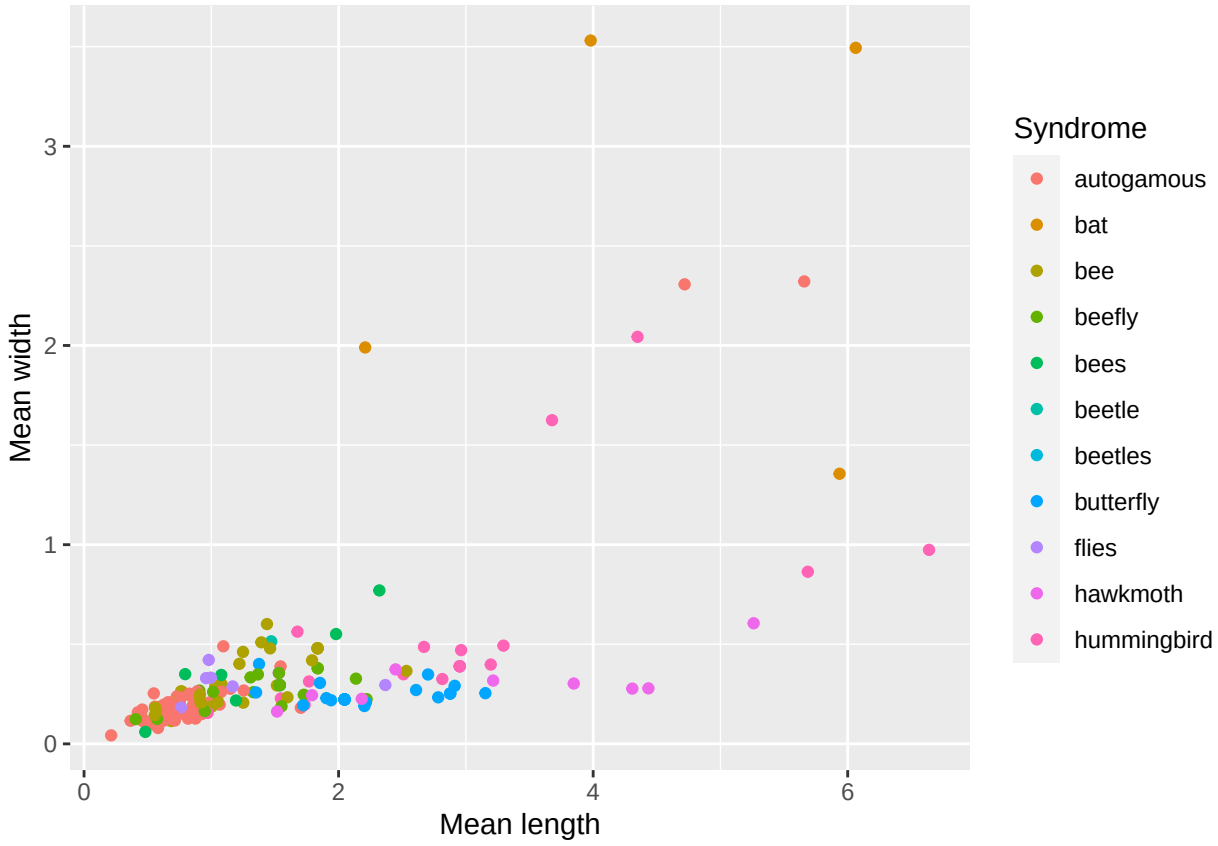
```
## # A tibble: 6 x 7
## # Groups:   genus [2]
##   genus      species      mean_length mean_width max_length max_width Syndrome
##   <chr>      <chr>              <dbl>      <dbl>      <dbl>     <dbl> <chr>
## 1 Acanthogi~ gloriosa            3.20      0.398       3.70      0.455 hummingbi~
## 2 Aliciella  caespitosa          1.73      0.197       1.88      0.241 hummingbi~
## 3 Aliciella  hutchinsifo~        0.818     0.158       0.957     0.272 autogamous
## 4 Aliciella  leptomeria          0.518     0.0963      0.731     0.141 autogamous
## 5 Aliciella  micromeria          0.213     0.0427      0.224     0.049 autogamous
## 6 Aliciella  pinnatifida         0.609     0.162       0.666     0.196 bee
```

Stage, commit, and push.

Plotting

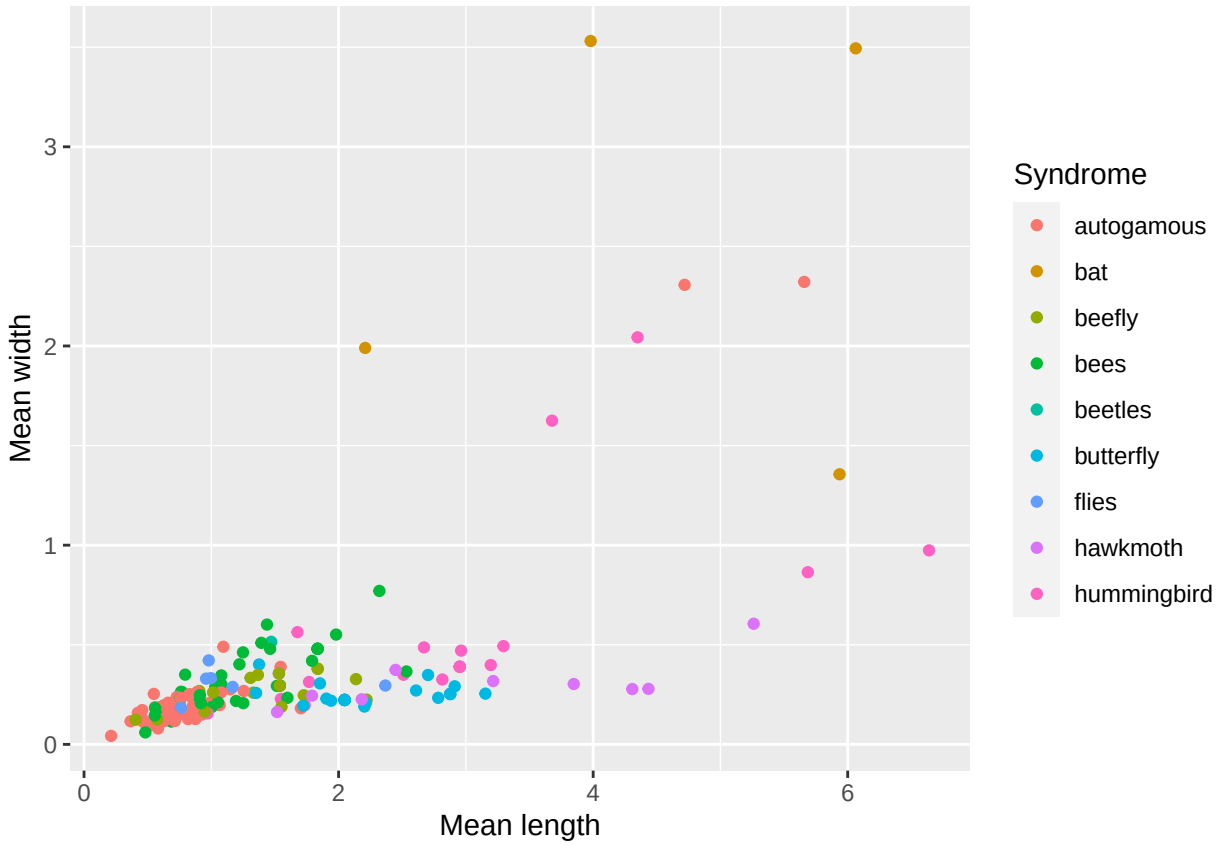
The data are now ready for plotting. You will make several variations of the same graph, slowly improving it. The goal is to show you a typical work flow, as you think about the best way to present the data. These steps are representative but not definitive. Each situation is unique.

- Make a scatterplot to show the relationship between `mean_length` (x-axis) and `mean_width` (y-axis). Color the points by `Syndrome`.



The legend lists "bee" and "bees", and "beetle" and "beetles". They were not coded consistently in the original data set. They need to be fixed.

- Use `mutate` with `str_replace()` and regex patterns to replace all occurences of "beetle" with "beetles," and "bee" with "bees". *Be careful.* If you don't do this right, then you could end up with "beetless". Make use of the `$` match character.
- Replot your results once you are sure you replaced the names correctly.



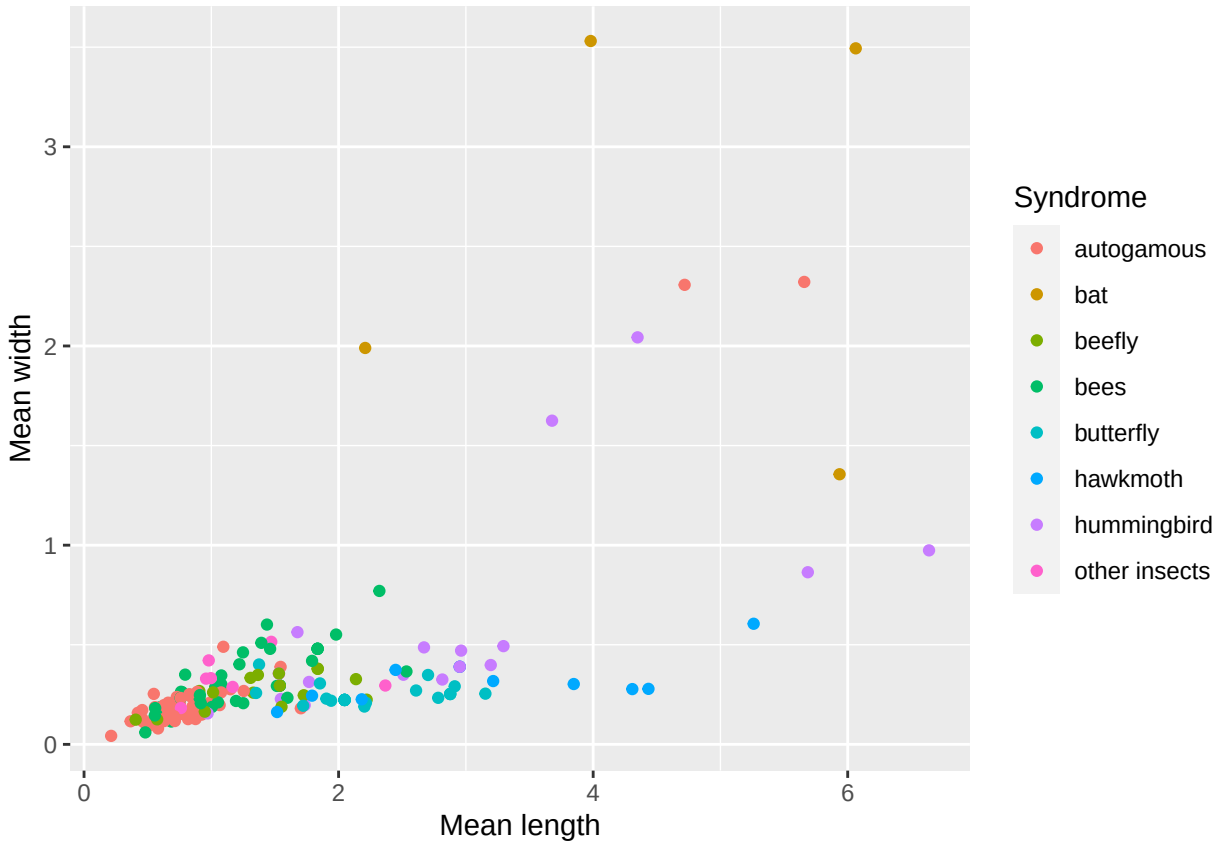
The bees and beetles problem is fixed but that is still a long list of pollination syndromes. If you run the following code, you see that there are not many species that are pollinated by bats, beetles, or flies.

```
phlox %>%
  group_by(Syndrome) %>%
  count(Syndrome)
```

```
## # A tibble: 9 x 2
## # Groups:   Syndrome [9]
##   Syndrome        n
##   <chr>       <int>
## 1 autogamous     55
## 2 bat             4
## 3 beefly         17
## 4 bees           41
## 5 beetles         3
## 6 butterfly      24
## 7 flies           7
## 8 hawkmoth       10
## 9 hummingbird    19
```

Bats are unusual pollinators (relative to other species on the list) so you want to keep them separate. You decide to lump beetles and flies together as `other insects`.

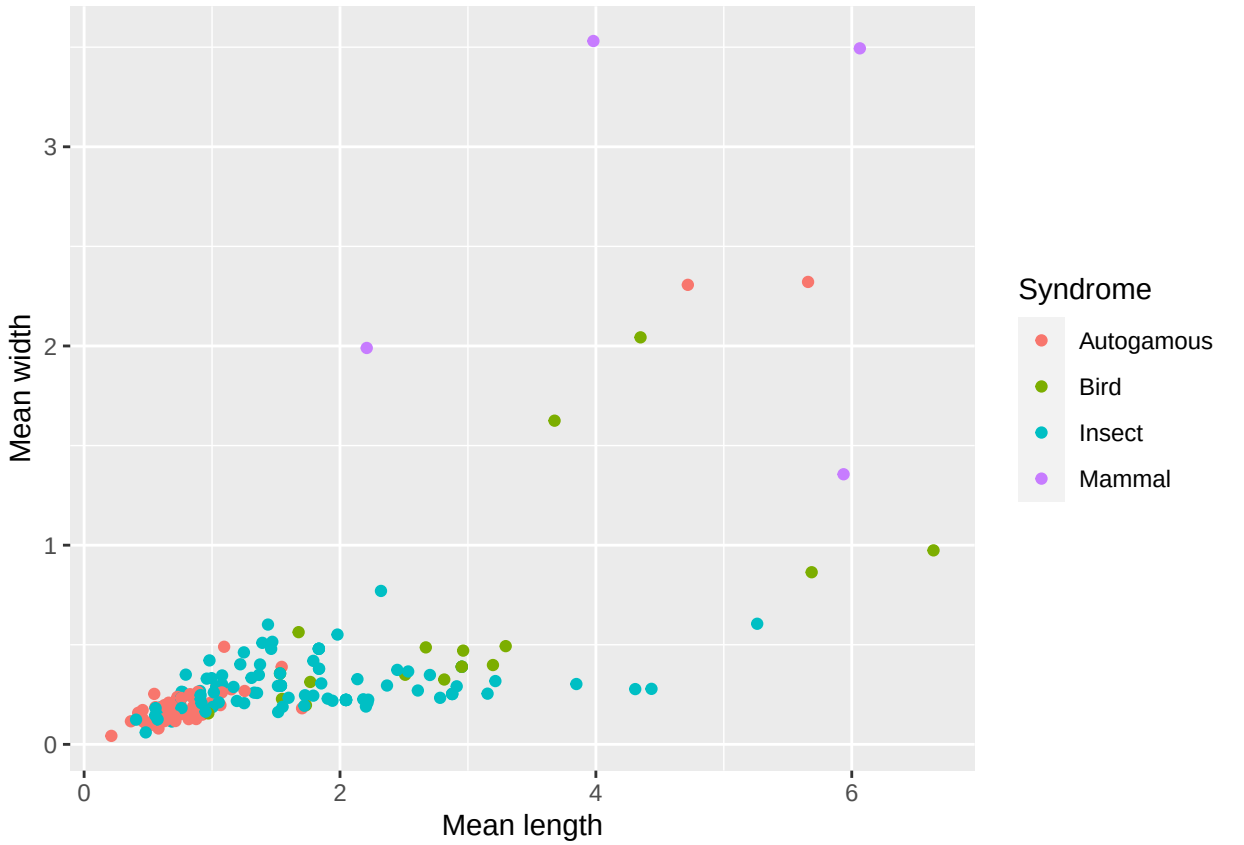
- Once again, use `mutate()` with `str_replace` and a regex pattern to replace all instances of "beetles" or "flies" with "other insects".
- Plot the results again.



Better but still not ideal. You decide you want to keep "autogamous" as a distinct sydrome but you want to group the animal pollinators into "Mammal", "Bird", and "Insect".

- Use `mutate` and `case_when` to create a new column called `syndrome_group` and to change `autogamous` to `Autogamous` (upper case A), `bat` to `Mammal`, `hummingbird` to `Bird`, and everything else to `Insect`. Creating a new column would allow you to access the individual syndromes for future plots, when necessary.
- Replot the results.

```
## # A tibble: 6 x 8
## # Groups:   genus [2]
##   genus species mean_length mean_width max_length max_width Syndrome
##   <chr> <chr>         <dbl>      <dbl>      <dbl>     <dbl> <chr>
## 1 Acan~ glorio~       3.20      0.398       3.70      0.455 humming~
## 2 Alic~ caespi~       1.73      0.197       1.88      0.241 humming~
## 3 Alic~ hutchi~       0.818     0.158       0.957     0.272 autogam~
## 4 Alic~ leptom~       0.518     0.0963      0.731     0.141 autogam~
## 5 Alic~ microm~       0.213     0.0427      0.224     0.049 autogam~
## 6 Alic~ pinnat~       0.609     0.162       0.666     0.196 bees
## # ... with 1 more variable: syndrome_group <chr>
```



The patterns in the data are easier to see. Large flowers tend to be pollinated by large animals. The smallest flowers tend to be self-pollinating, with the exception of the two large, autogamous species.

Highlighting groups with `gghighlight`

The different pollination syndromes can be distinguished through the use of colors. You can also highlight one or more groups with the `gghighlight` package. `gghighlight()` will highlight groups based on your logical criteria. The other groups are automatically given a light shade of gray, although this can be customized.

`gghighlight` works as a layer added to your `ggplot`. The basic format for adding a `gghighlight` layer is:

For example, if I want to highlight just the insect syndrome group, I add the line `gghighlight(syndrome_group == "Insect")` to the ggplot commands. Note that you still need to use `color` mapping in your `geom_point` layer so that `gghighlight` can properly highlight your group of interest.

Notice that the legend went away. This is a default behavior of `gghighlight` which can be overridden by adding the `use_direct_label = FALSE` argument to `gghighlight()`. This brings back the legend with only the highlighted group listed.

Read through the gghighlight vignette to learn more about how to use it.

Change the above plot so that it highlights both the mammal and bird syndrome groups. Hint: `|`.

For the rest of this assignment, do not use `gghighlight()`.

- Stage, commit, push.

Other improvements: colors and themes

One of the first things I change in a plot is default colors (when colors are needed). The default colors used by `ggplot` are carefully chosen so that they do not emphasize some data over others and can be distinguished by people with the most common forms of color blindness. These qualities are very important when using colors in figures, but I am not a particular fan of the default `ggplot` colors.

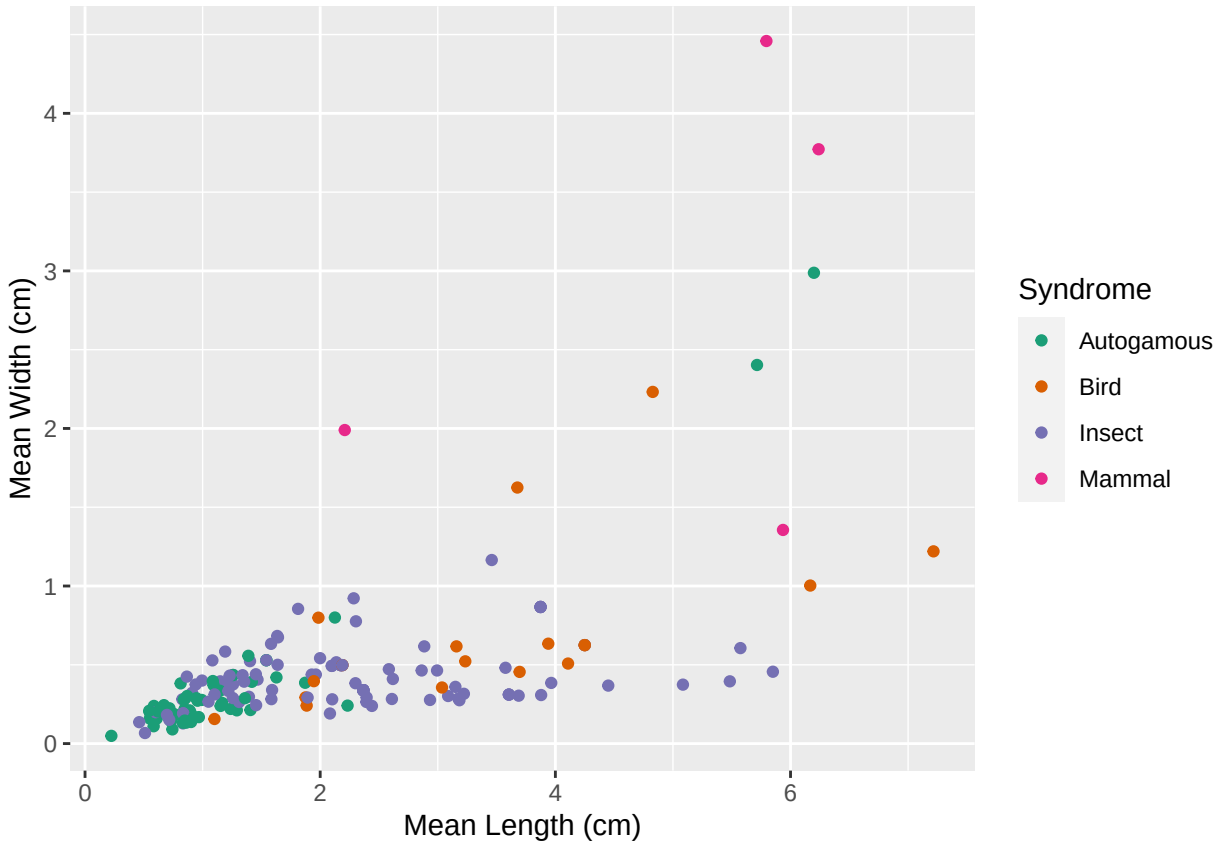
Other color sets are available with the desirable qualities of even emphasis and distinguishability by people with different visual abilities.

Dr. Cynthia Brewer developed several color palettes that are useful for plotting. The palettes were developing for cartography but some qualitative palettes work well with `ggplot`. Her palettes are available via the RColorBrewer package. Dr. Simon Garnier and colleages developed another set of palettes available via the viridis package.

For plots that use discrete categories like the pollination syndromes, I prefer using the brewer palettes. You can see the available palettes in the last page of this color cheatsheet.

You can access the brewer palettes by adding a `scale_color_brewer` layer like that shown here. I like the "Dark2" and "Set1" palettes but you should try different qualitative palettes to find what appeals to you. You should also try the diverging and sequential palettes to see why you should not use them for categorical data.

```
phlox %>%
  ggplot() +
  geom_point(aes(x = max_length,
                 y = max_width,
                 color = syndrome_group)) +
#  scale_color_viridis_d(option = "viridis")
  scale_color_brewer(type = "qual", palette = "Dark2") +
  labs(x = "Mean Length (cm)",
       y = "Mean Width (cm)",
       color = "Syndrome")
```



You can access the discrete viridis palettes using `scale_color_viridis_d` (use `_c` instead of `_d` to access the continuous palettes). Replace `scale_color_brewer` with `scale_color_viridis_d` and try different options, like "magma", "inferno", and "plasma". For me, the viridis palettes include one color that is very hard to see in a discrete setting. The viridis palettes work well for maps, which you will see in a later assignment.

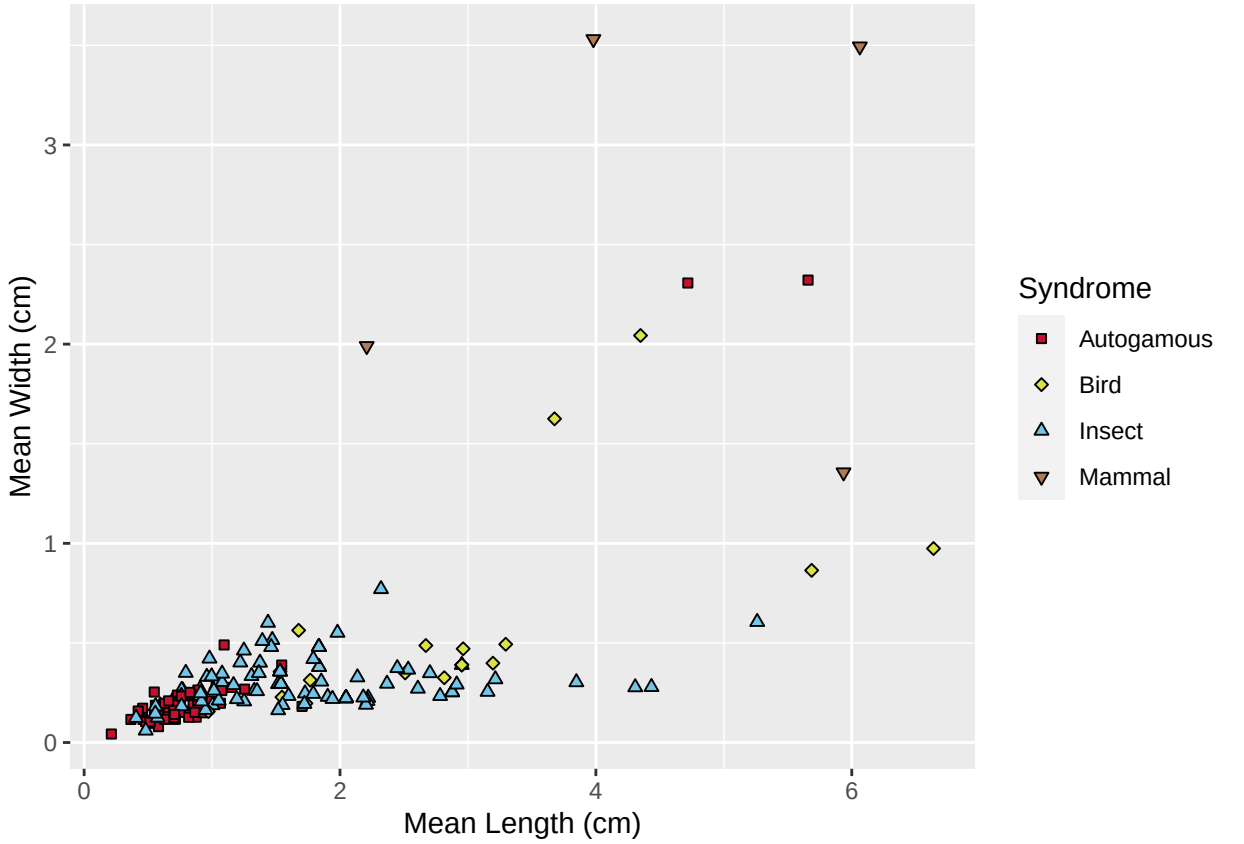
Shapes are another way of distinguishing categorical data. Here, I use four of the "fillable" shapes available in R. You can change the fill color but leave that border black. That allows even light colors like yellow to stand out from the background.

To change the fill color with the brewer or viridis palettes, you have to change `scale_color_...` to `scale_fill_...`, like that shown below. You can also use `scale_fill_manual` and specify your own colors, as shown below using official Southeast colors. Choose custom colors carefully because you must ensure they have equal emphasis and can be distinguished by as many people as possible.

```
# Custom color palette, using
# Southeast Red, Gum Tree, Fountain, Copper
semo_palette <- c("#C8102E", "#DBE245", "#71C5E8", "#B07C57")

phlox %>%
  ggplot() +
  geom_point(aes(x = mean_length,
                 y = mean_width,
                 shape = syndrome_group,
                 fill = syndrome_group)) +
  scale_shape_manual(values = c(22:25)) +
#  scale_fill_brewer(palette = "Dark2")
#  scale_fill_viridis_d(option = "viridis")
  scale_fill_manual(values = semo_palette) +
```

```
  labs(x = "Mean Length (cm)",
       y = "Mean Width (cm)",
       shape = "Syndrome",
       fill = "Syndrome")
```

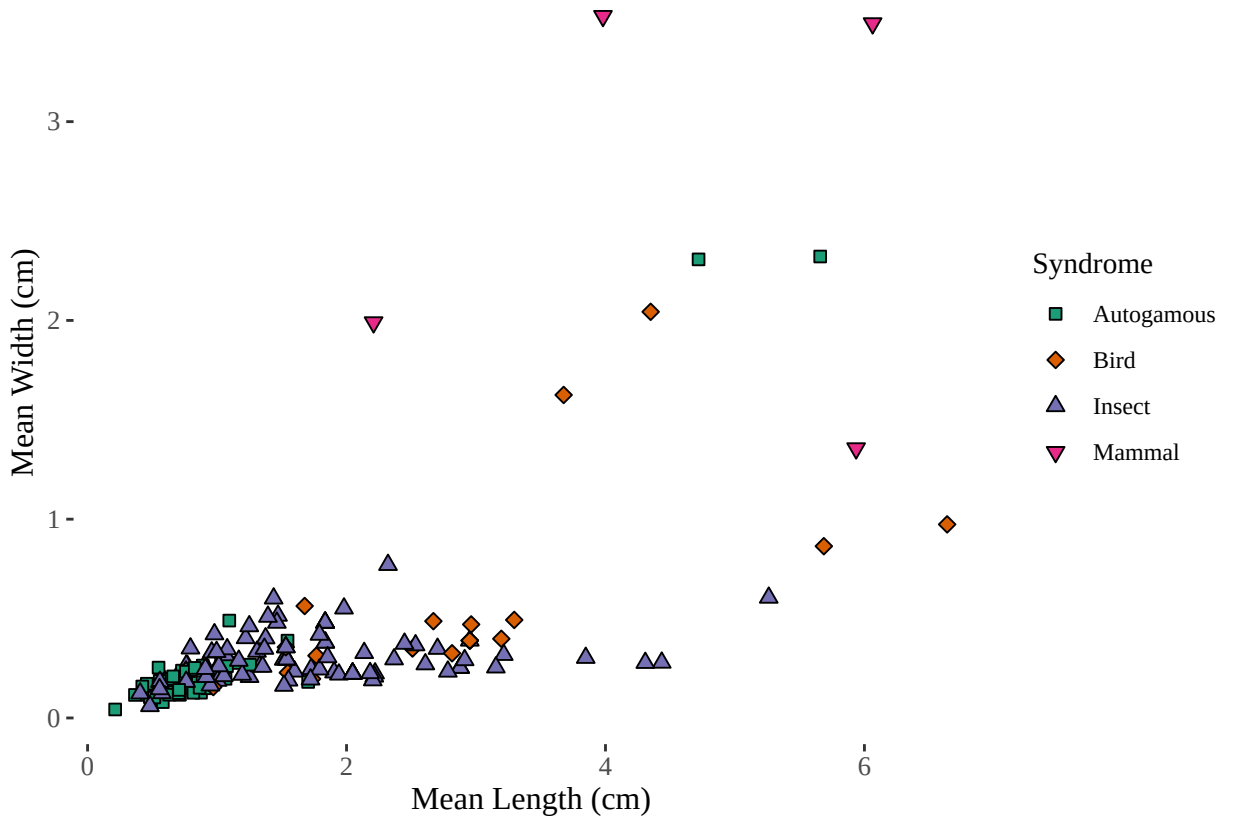


`ggplot` is highly customizable so you can make plots to suit very specific needs. `ggplot` includes several themes that change the overall look of the plot. Themes that I find useful are `theme_minimal`, `theme_bw` and `theme_classic`. The `ggthemes` library includes many additional themes. As I mentioned early in the course, I like the philosphy of Edward Tufte. Here is our plot using `theme_tufte()`, the `Dark2` brewer palette, `size = 2` to enlarge the plotted points a bit, and some font size customization to increase slightly the fonts used for the axis titles and the numbers at the tick marks.

Note: Some themes (I think) set default colors. If you add a `theme` layer after using one of the `scale` options, it is possible that your color choices will be overridden. Its a good idea to change the theme, and then make your color and other changes.

```
phlox %>%
  ggplot() +
  geom_point(aes(x = mean_length,
                 y = mean_width,
                 shape = syndrome_group,
                 fill = syndrome_group),
             size = 2) +
  labs(shape = "Syndrome",
       fill = "Syndrome",
       x = "Mean Length (cm)",
       y = "Mean Width (cm)") +
  theme_tufte() +
```

```
  scale_shape_manual(values = c(22:25)) +
  scale_fill_brewer(palette = "Dark2") +
  theme(axis.text = element_text(size = 10),
        axis.title = element_text(size = 12))
```



- Stage, commit, and push.

Part 4: Customize your plots

The assignment and examples above plotted mean width against mean length. I want you to make four scatterplots of max width as a function of max length. I want you to try different themes and palettes. Each plot should use a different combination of theme and color palettes.

For this exercise, I don't care about color choices. You can play with the brewer and viridis palettes, make your own, or even try one of the dozens of other palettes available via different packages. Any Wes Anderson fans? Pirate fans?

For fun, try to make at least one really good looking figure and one obnoxiously ugly one.

-Stage, commit, push.

---

et Vóila