Lucas Busta

Integrated Bioanalytical Chemistry

Contents

integrated bioanalytical chemistry

1. Analytical chemists separate, identify, and quantify matter. To connect this data with the world around us and answer scientific questions, multiple chemical entities must be separated, quantified, and identified. Challenge 1: As our ability to collect analytical data expands, so must our ability to effectively analyze that data - whether it's 10 data points or 10,000.

Question 1: A psychologist administers a test intended to measure intelligence. Participants complete different puzzles and answer different questions.

From a measurement theory point of view, what kind of variable is intelligence in this context?

A latent variable

An observed variable

A dependent variable

A measurement variable

Question 2: In OLS regression, a model is fit to the individual participant data. By contrast, regression in structural equation modeling fits a model to the observed covariance matrix.

TRUE

FALSE

Question 3: A model with more degrees of freedom is…

more complex

more simple

Question 4: Multiple regression and ANCOVA are statistically equivalent.

TRUE

FALSE

Question 5: A model with more degrees of freedom will…

fit the data better

fit the data worse

Question 6: The error term (ϵ_i) in regression equations reflects how much the observed scores of individuals differ from their predicted scores.

FALSE

TRUE

Question 7: What are degrees of freedom?

The number of unique pieces of information

The number of parameters

Number of participants minus number of parameters

The number of unique pieces of information minus number of parameters

Question 8: In the equation $Y_i = a + bX_i + e_i$, what are the ‘model parameters’?

a and bX_i

Y and X

a and b

Y_i, X_i and e_i

2. One of the largest obstacles facing scientists is communicating about our work with non-scientists. Challenge 2: We must practice oral and written science communication in both technical and non-technical formats.

This course is a set of first steps toward meeting both challenges outlined above. In the first half, we'll explore, critique, and practice methods of handling and communicating about the data generated in large analytical chemistry projects. In the second half, we'll apply the methods to large datasets and hone our writing skills by developing mini manuscripts that incorporate our large datasets.

Part I

data visualization

0.1 installation

0.1.1 R

R is the computing language we will use to run our chemometric analyses and produce high quality plots. If you already have R installed (you will need at least version 4.1.1), you can go straight to installing RStudio. If not, follow these steps to install R:

1. Go to https://cran.r-project.org/
2. Click on "Download R for <your operating system>" (see footnote), depending on your operating system you will select "Download R for Linux", "Download R for (Mac) OS X", or "Download R for Windows".

We will use <this notation> quite a bit. It indicates a place where you should insert information, data, or something similar that corresponds to your particular situation. In this example it means insert "your operating system", i.e. Linux, (Mac) OS X, or Windows.

3. For Mac: download the .pkg file for the latest release. For PC: click "install R for the first time", then click "Download R <version> for Windows".
4. After the executable finishes downloading (in Windows, it is a file with .exe extension; for Mac, it is a .dmg file or a .dmg inside a .pkg file), open the file as an administrator, and follow the installation instructions. R should install without any problems. You can click OK for all of the windows that pop-up during installation, and choose a "regular" installation (if given the choice).

If you have trouble installing R please google "Install R Mac" or "Install R PC" and follow one the many video tutorials out there. If you have tried this and are still having trouble, please contact me.

0.1.2 RStudio

Once we install R, we can install RStudio, which is essentially a convenient way of interacting with R. Some people do not like RStudio and prefer to interact with R directly. This is fine, but many beginning R users find RStudio helpful, so I recommend it. Follow these steps to install RStudio:

1. Go to https://rstudio.com/

2. Click “DOWNLOAD” at the top of the page.
3. Click the “DOWNLOAD” button that corresponds to RStudio Desktop with the free Open Source License.
4. The page may automatically detect which operating system you are using and recommend a version for you. If it does, download that file (.exe for PC or .dmg for Mac). If not, scroll down to the “All Installers” section and download the file that is right for you. Open the file as an administrator, and follow the installation instructions. RStudio should install without any problems. You can click OK for all of the windows that pop-up during installation, and choose a “regular” installation (if given the choice).

If you have trouble installing RStudio please google “Install RStudio Mac” or “Install RStudio PC” and following one the many video tutorials out there. If you have tried this and are still having trouble, please contact me.

0.1.3 Verification

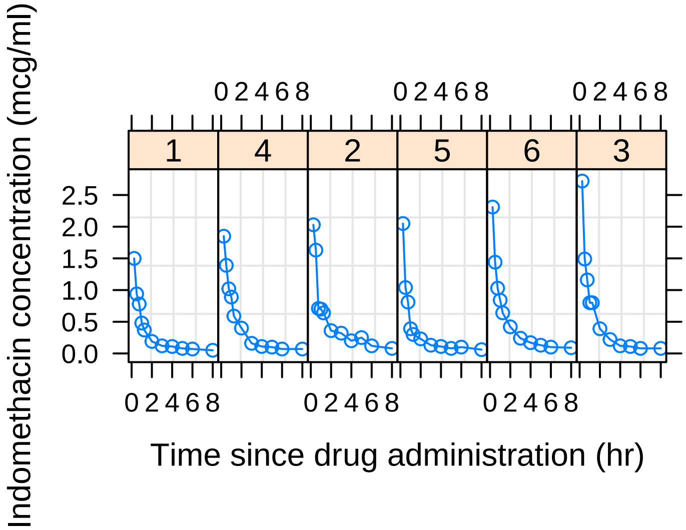
Open RStudio by clicking on the appropriate file in your applications folder, or wherever it is saved on your computer. You will see several windows. One is the Code Editor, one is the R Console, one is the Workspace and History, and one is the Plots and Files window.

The R Console window should have a > in it. Type `head(Indometh)`. This should display the first six lines of a data set describing the pharmacokinets of indomethacin. This is one of the built in datasets in R - you do not need any additional files to run this test.

```
head(Indometh)
## Grouped Data: conc ~ time | Subject
##   Subject time conc
## 1       1 0.25 1.50
## 2       1 0.50 0.94
## 3       1 0.75 0.78
## 4       1 1.00 0.48
## 5       1 1.25 0.37
## 6       1 2.00 0.19
```

Next, type `plot(Indometh)` into the R Console. This will plot the indomethacin dataset in a basic way.

```
plot(Indometh)
```



If both the above commands (`head(Indometh)` and `plot(Indometh)`) worked and there were no error messages during installation, then you should be ready to proceed.

0.1.4 tidyverse

For us to run our analyses, we need to install a set of add-on functions that expand R's capabilities. These functions are collected in something called the tidyverse, a very well-known and widely-used R package. You do not need to manually download anything to complete this installation - R will do it for you. In the R Console, type `install.packages("tidyverse", repos = "http://cran.us.r-project.org")` to install the tidyverse. Let's try it:

RSudio might ask you: "Do you want to install from sources the packages which need compilation? (Yes/no/cancel)", for now, type `no` and press enter.

```
install.packages("tidyverse", repos = "http://cran.us.r-project.org")
```

Let's make sure your version of the tidyverse is installed correctly. To do this, we will load the `tidyverse` library/package inside of an R session. We can do this using `library(tidyverse)`. Let's try it:

```
library(tidyverse)
```

If the library load correctly - then you are set to go! If not, try updating your R / RStudio installations, the re installing the tidyverse. If this still fails, please contact me.

0.1.5 TeX

In this class we will generate high quality reports suitable for submission to supervisors, academic journals, etc. For this, we need the typesetting engine TeX. There are a few ways to do this. The easiest way is using the following commands:

```
install.packages(c('tinytex', 'rmarkdown'))
```

If you are on Mac, you may get an error about "not being able to write to a path" or something like that. In that case you probably need to open your terminal and run the following two commands:

sudo chown -R ‘whoami‘:admin /usr/local/bin

and then

~/Library/TinyTeX/bin/*/tlmgr path add

Then, on both Mac and PC, you then need to do:

```
tinytex::install_tinytex()
```

Other options are: if you have Windows, download and install MikTeX. If you have OSX, you can download and install MacTeX.

0.1.6 phylochemistry

In addition to the tidyverse, there are a variety of other packages we will need, as well as some datasets and custom functions. These call all be loaded by doing the following.

First, attempt to load phylochemistry:

```
source("http://thebustalab.github.io/phylochemistry/phylochemistry.R")
```

The first time you try this, it will very likely say: "You need to install the following packages before proceeding […] Run: installPhylochemistry() to automatically install the required packages."

This means that some of the prerequisite packages that phylochemistry needs are not installed. If this happens, run the following:

```
installPhylochemistry()
```

Sometimes when you run `installPhylochemistry()` you will get a message:

`Update all/some/none? [a/s/n]:`

In this case, it is generally advisable to enter `a` into the console and then press enter, indicating to R that you wish to update anything and everything that can be updated.

Other times you may get this message:

```
Do you want to install from sources the packages which need
compilation? (Yes/no/cancel)
```

You can reply `yes` if you wish, but for simplicity's sake it is okay to say `no`. I usually start with saying `no`, only reverting to `yes` if things don't work down the line.

Once that is complete, and assuming no errors are displayed, attempt to load phylochemistry again:

```
source("http://thebustalab.github.io/phylochemistry/phylochemistry.R")
```

0.1.7 xcms

If you wish to run the GC-MS integration app that comes with phylochemistry, please also install XCMS by running the following in your RStudio console:

```
if (!requireNamespace("BiocManager", quietly = TRUE))
    install.packages("BiocManager")

BiocManager::install("xcms")
```

0.1.8 Updating R and R Packages

To update R:

```
install.packages('devtools') #assuming it is not already installed

library(devtools)

install_github('andreacirilloac/updateR')

library(updateR)

updateR()
```

0.2 ggplot and markdown

0.2.1 Objects

Ok, we've got the installation out of the way. Let's get down to working with data and generating reports! In R, data is stored in objects. You can think of objects as if they were “files” inside an R session. `phylochemistry` provides a variety of objects for us to work with.

Let’s look at how to create an object. For this, we can use an arrow: `<-` . The arrow will take something and store it inside an object. For example:

```
new_object <- 1
```

Now we’ve got a new object called `new_object`, and inside of it is the number 1. To look at what’s inside an object, we can simply type the name of the object into the console:

```
new_object
## [1] 1
```

Easy! Let’s look at one of the objects that comes with our class code base. What are the dimensions of the “algae_data” data set?

```
algae_data
## # A tibble: 180 x 5
##    replicate algae_strain harvesting_regime chemical_species
##        <dbl> <chr>        <chr>             <chr>
##  1         1 Tsv1         Heavy             FAs
##  2         1 Tsv1         Heavy             saturated_Fas
##  3         1 Tsv1         Heavy             omega_3_polyuns~
##  4         1 Tsv1         Heavy             monounsaturated~
##  5         1 Tsv1         Heavy             polyunsaturated~
##  6         1 Tsv1         Heavy             omega_6_polyuns~
##  7         1 Tsv1         Heavy             lysine
##  8         1 Tsv1         Heavy             methionine
##  9         1 Tsv1         Heavy             essential_Aas
## 10         1 Tsv1         Heavy             non_essential_A~
## # ... with 170 more rows, and 1 more variable:
## #   abundance <dbl>
```

0.2.2 Functions

Excellent - we’ve got data. Now we need to manipulate it. For this we need functions:

- A function is a command that tells R to perform an action!
- A function begins and ends with parentheses: `this_is_a_function()`
- The stuff inside the parentheses are the details of how you want the function to perform its action: `run_this_analysis(on_this_data)`

Let’s illustrate this with an example. `algae_data` is a pretty big object. For our next chapter on visualization, it would be nice to have a smaller dataset object to work with. Let’s use another `tidyverse` command called `filter` to filter the `algae_data` object. We will need to tell the filter command what to filter out using "logical predicates" (things like equal to: `==`, less than: `<`, greater than: `>`, greater-than-or-equal-to: `<=`, etc.). Let's filter `algae_data` so that only rows where the `chemical_species` is equal to `FAs` (fatty acids) is preserved. This will look like `chemical_species == "FAs"`. Here we go:

```
filter(algae_data, chemical_species == "FAs")
## # A tibble: 18 x 5
##    replicate algae_strain harvesting_regime chemical_species
##        <dbl> <chr>        <chr>             <chr>
##  1         1 Tsv1         Heavy             FAs
##  2         2 Tsv1         Heavy             FAs
##  3         3 Tsv1         Heavy             FAs
##  4         1 Tsv1         Light             FAs
##  5         2 Tsv1         Light             FAs
##  6         3 Tsv1         Light             FAs
##  7         1 Tsv2         Heavy             FAs
##  8         2 Tsv2         Heavy             FAs
##  9         3 Tsv2         Heavy             FAs
## 10         1 Tsv2         Light             FAs
## 11         2 Tsv2         Light             FAs
## 12         3 Tsv2         Light             FAs
## 13         1 Tsv11        Heavy             FAs
## 14         2 Tsv11        Heavy             FAs
## 15         3 Tsv11        Heavy             FAs
## 16         1 Tsv11        Light             FAs
## 17         2 Tsv11        Light             FAs
## 18         3 Tsv11        Light             FAs
## # ... with 1 more variable: abundance <dbl>
```

Cool! Now it's just showing us the 18 rows where the chemical_species is fatty acids (FAs). Let's write this new, smaller dataset into a new object. For that we use `<-`, remember?

```
algae_data_small <- filter(algae_data, chemical_species == "FAs")
algae_data_small
## # A tibble: 18 x 5
##    replicate algae_strain harvesting_regime chemical_species
##        <dbl> <chr>        <chr>             <chr>
##  1         1 Tsv1         Heavy             FAs
##  2         2 Tsv1         Heavy             FAs
##  3         3 Tsv1         Heavy             FAs
##  4         1 Tsv1         Light             FAs
##  5         2 Tsv1         Light             FAs
##  6         3 Tsv1         Light             FAs
##  7         1 Tsv2         Heavy             FAs
##  8         2 Tsv2         Heavy             FAs
##  9         3 Tsv2         Heavy             FAs
## 10         1 Tsv2         Light             FAs
## 11         2 Tsv2         Light             FAs
## 12         3 Tsv2         Light             FAs
## 13         1 Tsv11        Heavy             FAs
## 14         2 Tsv11        Heavy             FAs
## 15         3 Tsv11        Heavy             FAs
## 16         1 Tsv11        Light             FAs
## 17         2 Tsv11        Light             FAs
## 18         3 Tsv11        Light             FAs
## # ... with 1 more variable: abundance <dbl>
```

0.2.3 ggplot

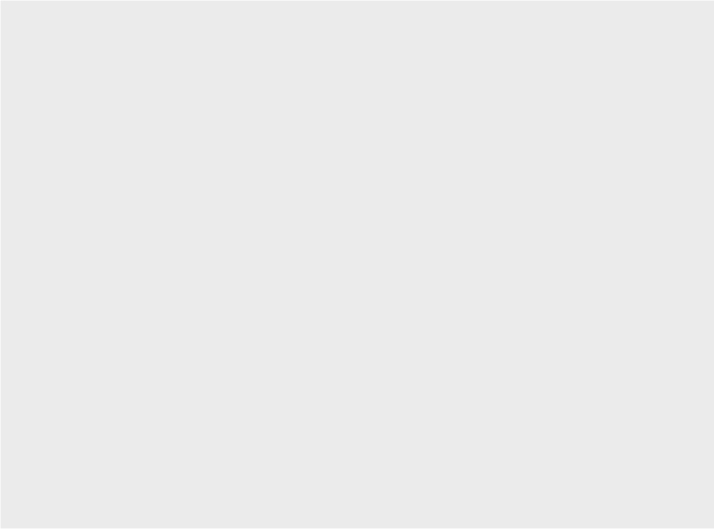
Now we have a nice, small table that we can use to practice data visualization. For visualization, we're going to use `ggplot2` - a powerful set of commands for plot generation.

There are three steps to setting up a ggplot:

1. **Define the data you want to use.**

We do this using the ggplot function's data argument. When we run that line, it just shows a grey plot space. Why is this? It's because all we've done is told ggplot that (i) we want to make a plot and (ii) what data should be used. We haven't explained how to represent features of the data using ink.

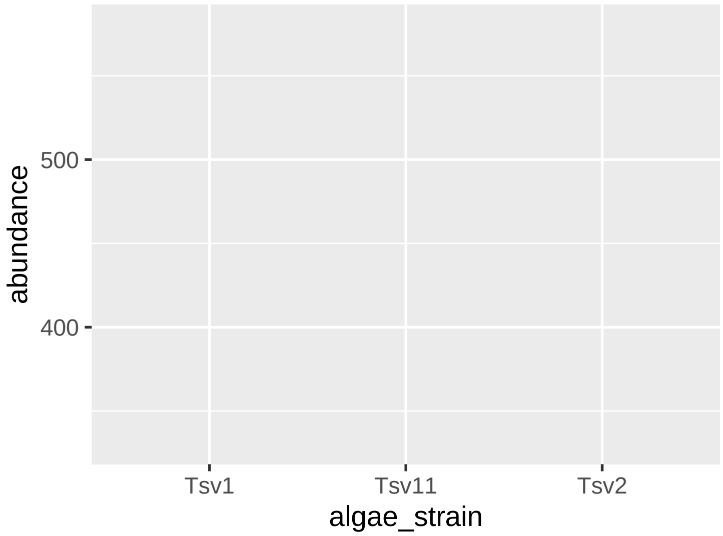
```
ggplot(data = algae_data_small)
```



2. **Define how your variables map onto the axes.**

This is called aesthetic mapping and is done with the aes() function. aes() should be placed inside the ggplot command. Now when we run it, we get our axes!

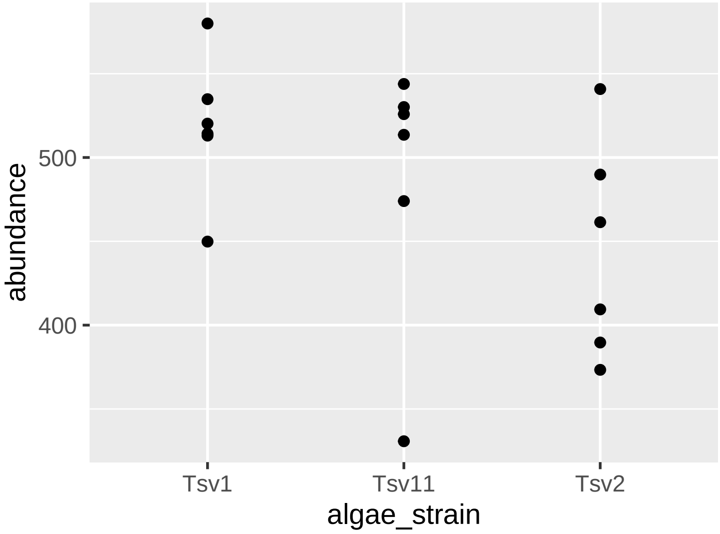
```
ggplot(data = algae_data_small, aes(x = algae_strain, y = abundance))
```



3. **Use geometric shapes to represent other variables in your data.**

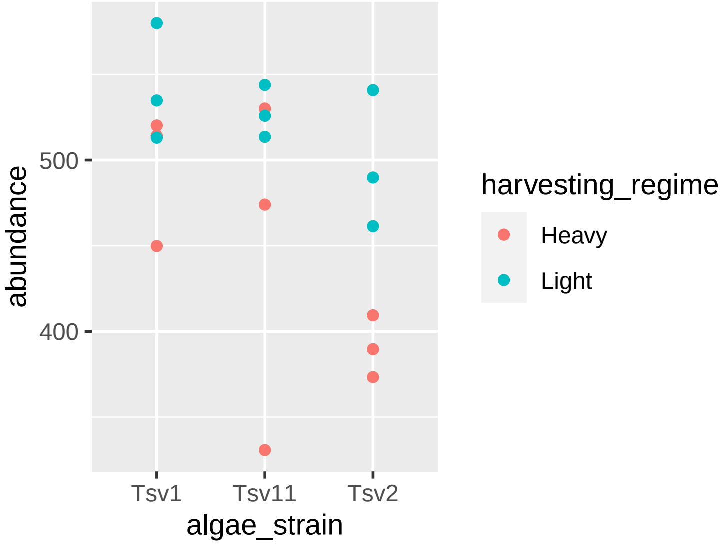
Map your variables onto the geometric features of the shapes. To define which shape should be used, use a geom_* command. Some options are, for example, geom_point(), geom_boxplot(), and geom_violin(). These functions should be added to your plot using the + sign. We can use a new line to keep the code from getting too wide, just make sure the + sign is at the end fo the top line. Let's try it:

```
ggplot(data = algae_data_small, aes(x = algae_strain, y = abundance)) +
  geom_point()
```



In the same way that we mapped variables in our dataset to the plot axes, we can map variables in the dataset to the geometric features of the shapes we are using to represent our data. For this, again, use `aes()` to map your variables onto the geometric features of the shapes:

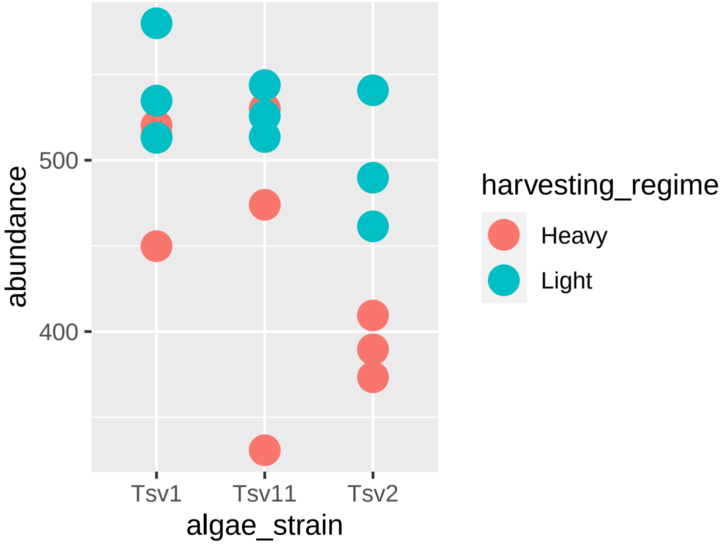
```
ggplot(data = algae_data_small, aes(x = algae_strain, y = abundance)) +
  geom_point(aes(color = harvesting_regime))
```



0.2.3.1 modifying geoms

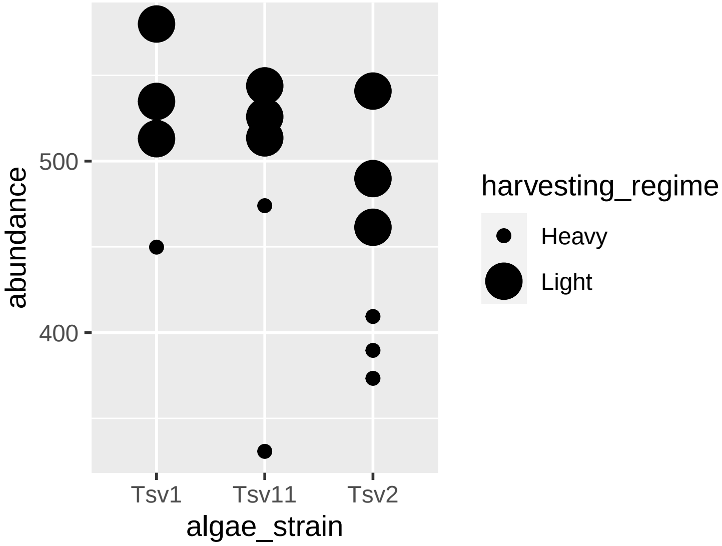
In the last plot in the previous section, the points were a bit small, how could we fix that? We can modify the features of the shapes by adding additional arguments to the `geom_*()` functions. To change the size of the points created by the `geom_point()` function, this means that we need to add the `size =` argument. Here's an example:

```
ggplot(data = algae_data_small, aes(x = algae_strain, y = abundance)) +
  geom_point(aes(color = harvesting_regime), size = 5)
```



One powerful aspect of `ggplot` is the ability to quickly change mappings to see if alternative plots are more effective at bringing out the trends in the data. For example, we could modify the plot above by switching how harvesting_regime is mapped:

```
ggplot(data = algae_data_small, aes(x = algae_strain, y = abundance)) +
  geom_point(aes(size = harvesting_regime), color = "black")
```

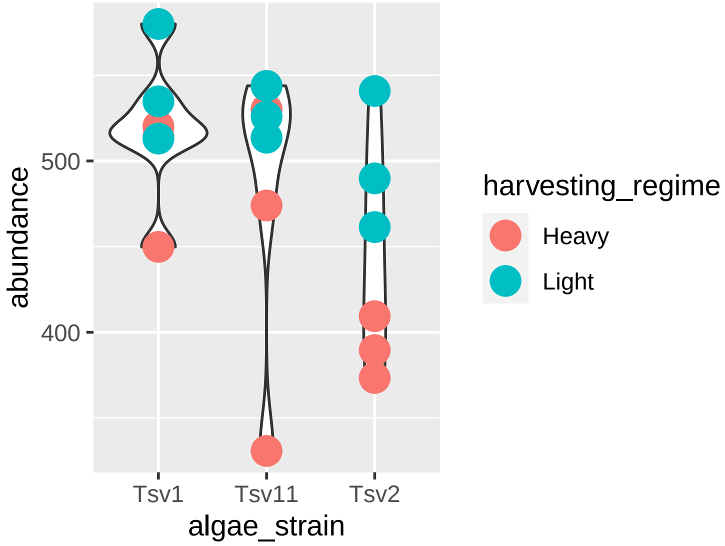


** Important note: Inside the `aes()` function, map aesthetics (the features of the geom's shape) to a *variable*. Outside the `aes()` function, map aesthetics to *constants*. You can see this in the above two plots - in the first one, color is inside `aes()` and mapped to the variable called harvesting_regime, while size is outside the `aes()` call and is set to the constant 5. In the second plot, the situation is reversed, with size being inside the `aes()` function and mapped to the variable harvesting_regime, while color is outside the `aes()` call and is mapped to the constant "black".

0.2.3.2 multiple geoms

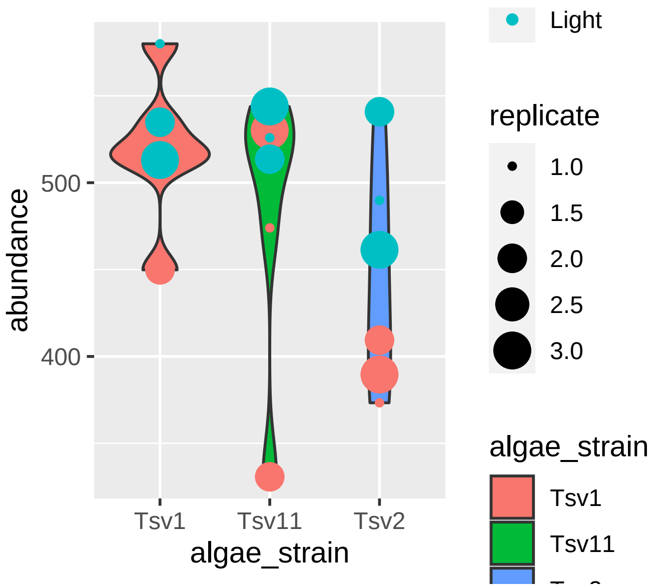
We can also stack geoms on top of one another by using multiple + signs. We also don't have to assign the same mappings to each geom.

```
ggplot(data = algae_data_small, aes(x = algae_strain, y = abundance)) +
  geom_violin() +
  geom_point(aes(color = harvesting_regime), size = 5)
```

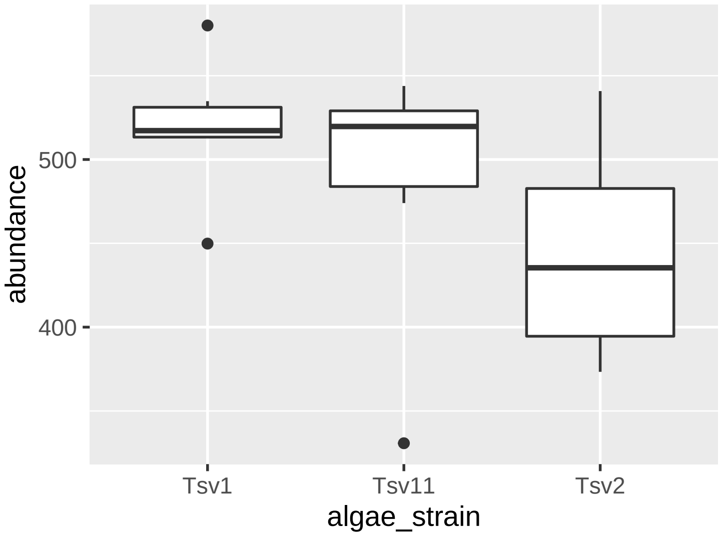


As you can probably guess right now, there are lots of mappings that can be done, and lots of different ways to look at the same data!

```r
ggplot(data = algae_data_small, aes(x = algae_strain, y = abundance)) +
  geom_violin(aes(fill = algae_strain)) +
  geom_point(aes(color = harvesting_regime, size = replicate))
```



```r
ggplot(data = algae_data_small, aes(x = algae_strain, y = abundance)) +
  geom_boxplot()
```



0.2.4 markdown

Now that we are able to filter our data and make plots, we are ready to make reports to show others the data processing and visualization that we are doing. For this, we will use R Markdown. You can open a new markdown document in RStudio by clicking: `File -> New File -> R Markdown`. You should get a template document that compiles when you press "knit".

Customize this document by modifying the title, and add `author: "your_name"` to the header. Delete all the content below the header, then compile again. You should get a page that is blank except for the title and the author name.

You can think of your markdown document as a stand-alone R Session. This means you will need to load our class code base into each new markdown doument you create. You can do this by adding a "chunk" or R code. That looks like this:

You should notice a few things when you compile this document:

1. Headings: When you compile that code, the "# My first analysis" creates a header. You can create headers of various levels by increasing the number of hashtags you use in front of the header. For example, "## Part 1" will create a subheading, "### Part 1.1" will create a sub-subheading, and so on.
2. Plain text: Plain text in an R Markdown document creates a plan text entry in your compiled document. You can use this to explain your analyses and your figures, etc.

We can also run R chunks right in markdown and create figures. Dr. Busta will show you how to do this in class.

0.2.5 exercises

In this set of exercises we're going to practice filtering and plotting data in R Markdown. We're going to work with two datasets: (i) algae_data and (ii) alaska_lake_data. **For these exercises, you will write your code and answers to all questions in an R Markdown report, compile it as a pdf, and submit it on Canvas. If you have any questions please let me know**

Some pointers:

- If your code goes off the page, don't be afraid to wrap it across multiple lines, as shown in some of the examples.
- Don't be afraid to put the variable with the long elements / long text on the y-axis and the continuous variable on the x-axis.

0.2.5.1 Algae Chemistry Dataset

1. **Filtering 1**

You will have `algae_data` stored in an object called `algae_data` as soon as you run `source("http://thebustalab.github.io/phylochemistry/phylochemistry.R")`. For this question, filter the data so that only entries are shown for which the `chemical_species` is "FAs" (remember that quotes are needed around FAs here!). What are the dimensions (i.e. number of rows and columns) of the resulting dataset?

2. **Filtering 2**

Now filter the dataset so that only entries for the `algae_strain` "Tsv1" are shown. What are the dimensions of the resulting dataset?

3. **Filtering 3**

Now filter the dataset so that only entries with an abundance greater than 250 are shown. Note that `>` can be used in the filter command instead of `==`, and that numbers inside a filter command do not require quotes around them. What are the dimensions of the resulting dataset?

4. **Plotting**

Make a ggplot that has `algae_strain` on the x axis and `abundance` on the y axis. Remember about `aes()`. Use points (`geom_point()`) to represent each compound. You don't need to color the points.

Which algae strain has the most abundant compound out of all the compounds in the dataset?

5. Plotting

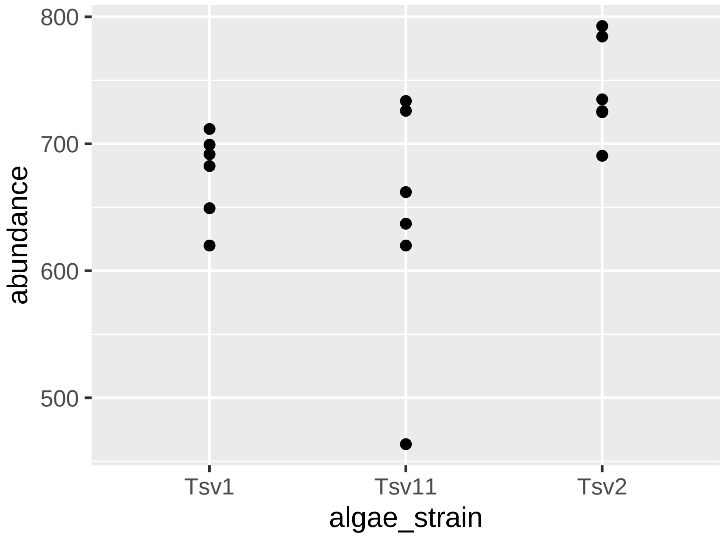
Make a ggplot that has `abundance` on the x axis and `chemical_species` on the y axis. Use points to represent each compound. You don't need to color the points.

Generally speaking, what are the two most abundant classes of chemical species in these algae strains? (FAs/Fas stand for fatty acids, AAs/Aas stand for amino acids.)

6. Filtering and plotting

I am going to show you an example of how you can filter and plot at the same time. To do this, we nest the filter command inside ggplot's data argument:

```
ggplot(
  data = filter(algae_data, chemical_species == "essential_Aas"),
  aes(x = algae_strain, y = abundance)) +
geom_point()
```



Using the above as a template, make a plot that shows just `omega_3_polyunsaturated_Fas`, with algae_strain on the x axis, and abundance on the y axis. Color the points so that they correspond to `harvesting_regime`. Remember that mapping a feature of a shape onto a variable must be done inside `aes()`. Change the plot so that all the points are size = 5. Remember that mapping features of a shape to a constant needs to be done outside `aes()`. Which harvesting regime leads to higher levels of `omega_3_polyunsaturated_Fas`?

7. Filtering and plotting

Use a combination of filtering and plotting to show the abundance of the different chemical species in just the `algae_strain` called "Tsv1". Use an x and y axis, as well as points to represent the measurements. Make point size correspond to the replicate, and color the points according to harvesting regime.

8. Open-ended plotting

Make a plot that checks to see which `chemical_species` were more abundant under light as opposed to heavy `harvesting_regime` in all three replicates. Use filtered data so that just one `algae_strain` is shown, an x and a y axis, and points to represent the measurements. Make the points `size = 5` and also set the point's `alpha = 0.6`. The points should be colored according to harvesting_regime. Make 3 plots, one for each strain of algae.

9. A peek at what's to come...

Take the code that you made for Question 9. Remove the filtering. Add the following line to the end of the plot: `facet_grid(.~algae_strain)`. Remember that adding things to plots is done with the `+` sign, so your code should look something like:

```
ggplot(data = algae_data, aes(x = <something>, y = <something else>)) +
  geom_point(aes(<some things>), <some here too>) +
  facet_grid(.~algae_strain)
```

Also try, instead of `facet_grid(.~algae_strain)`, `facet_grid(algae_strain~.)` at the end of you plot command. (note the swap in the position of the `.~` relative to `algae_strain`). This means your code should look something like:

```
ggplot(data = algae_data, aes(x = <something>, y = <something else>)) +
  geom_point(aes(<some things>), <some here too>) +
  facet_grid(algae_strain~.)
```

What advantages does this one extra line (i.e. facet_grid) provide over what you had to do in question 8?

0.2.6 Alaska Lakes Dataset

1. **Viewing Data**

Use R to view the first few lines of the `alaska_lake_data` dataset. Do your best to describe, in written format, the kind of data that are in this data set.

2. **Objects**

How many variables are in the Alaska lakes dataset?

3. **Filtering**

Filter the data set so only meausurements of free elements (i.e. element_type is "free") are shown. Remember, it's `==`, not `=`. What are the dimensions of the resulting dataset?

4. **Plotting**

Make a plot that shows the water temperatures of each lake. Don't worry if you get a warning message from R about "missing values". Which is the hottest lake? The coolest?

5. **Plotting**

Make a plot that shows the water temperature of each lake. The x axis should be `park`, the y axis `water temp`. Add geom_violin() to the plot first, then geom_point(). Make the points size = 5. Color the points according to water_temp. Which park has four lakes with very similar temperatures?

6. **Filtering and Plotting**

From the plot you made for question 5, it should be apparent that there is one lake in NOAT that is much warmer than the others. Filter the data so that only entries from `park == "NOAT"` are shown (note the double equals sign and the quotes around NOAT…). Combine this filtering with plotting and use geom_point() to make a plot that shows which specific lake that is.

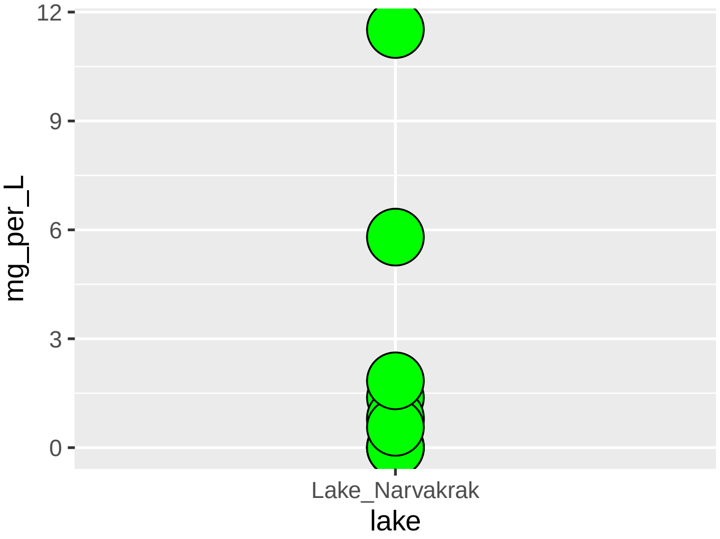
7. **Filtering and Plotting**

Make a plot that shows which lake has the highest abundance of sulfur.

8. **Open-ended Plotting**

Make a plot that uses geom_point(). Set the "shape" aesthetic of the points to 21, i.e. `geom_point(aes(...), shape = 21)`. This gives you access to a new aesthetics: `fill`. It also changes the behaviour of the `color` aesthetic slightly, in that it now controls border color, not the internal color. Here is an example (though it doesn't make a very nice plot):

```
ggplot(
  data = filter(alaska_lake_data, lake == "Lake_Narvakrak"),
  aes(x = lake, y = mg_per_L)
) +
  geom_point(
    shape = 21, size = 10,
    color = "black", fill = "green"
  )
```



Now we have lots of aesthetics we can map to: x, y, size, color, and fill (leave shape set to 21 for now). Make a plot of your own design. It should include filtering, and all the aesthetics listed above, though whether you map them to a variable or a constant is up to you.

When you are done with this plot, take a screen shot of it. Go to THIS GOOGLE SHEET, make a slide for yourself (you don't have to include your name), and paste your screen shot there. Add a small caption that explains how your variables are mapped.

data visualization

Stuff here on deseption with graphics, tufte, etc.

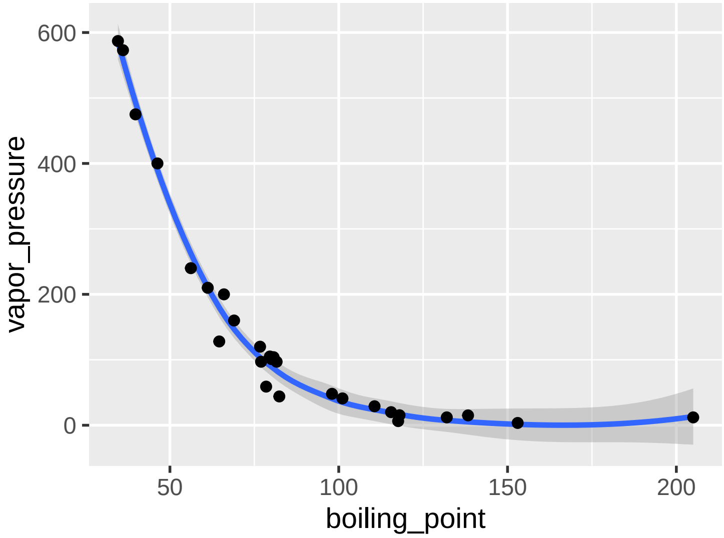
0.3 geoms, facets, scales, themes

We've looked at how to filter data and map variables in our data to geometric shapes to make plots. Let's have a look at a few more things. For these examples, we're going to use the data set called `solvents`.

0.3.1 geoms

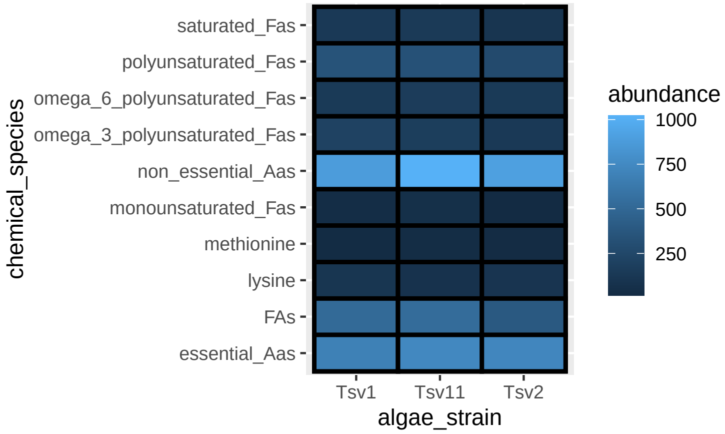
I'd like to introduce you to two new geoms. The first `geom_smooth()` is used when there are two continuous variables. It is particularly nice when geom_point() is stacked on top of it.

```
ggplot(data = solvents, aes(x = boiling_point, y = vapor_pressure)) +
  geom_smooth() +
  geom_point()
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```



Also, please be aware of `geom_tile()`, which is nice for situations with two discrete variables and one continuous variable. `geom_tile()` makes what are often referred to as heat maps. Note that `geom_tile()` is somewhat similar to `geom_point(shape = 21)`, in that it has both `fill` and `color` aesthetics that control the fill color and the border color, respectively.

```
ggplot(
  data = filter(algae_data, harvesting_regime == "Heavy"),
  aes(x = algae_strain, y = chemical_species)
) +
  geom_tile(aes(fill = abundance), color = "black", size = 1)
```



These examples should illustrate that there is, to some degree, correspondence between the type of data you are interested in plotting (number of discrete and continuous variables) and the types of geoms that can effectively be used to represent the data.

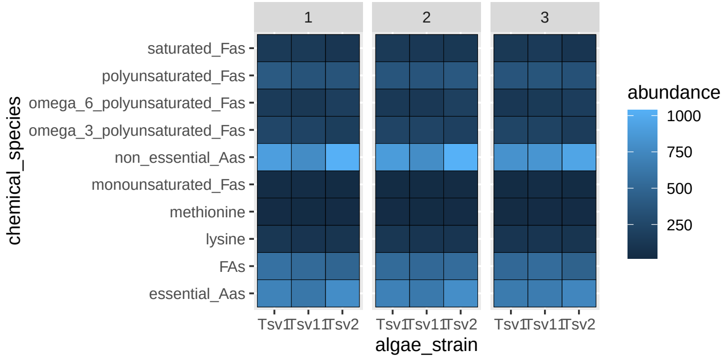
There is a handy cheat sheet that can help you identify the right geom for your situation. Please keep this cheat sheet in mind for your future plotting needs…

0.3.2 facets

As alluded to in Exercises 1, it is possible to map variables in your dataset to more than the geometric features of shapes (i.e. geoms). One very common way of doing this is with facets. Faceting creates small multiples of your plot, each of which shows a different subset of your data based on a categorical variable of your choice. Let's check it out.

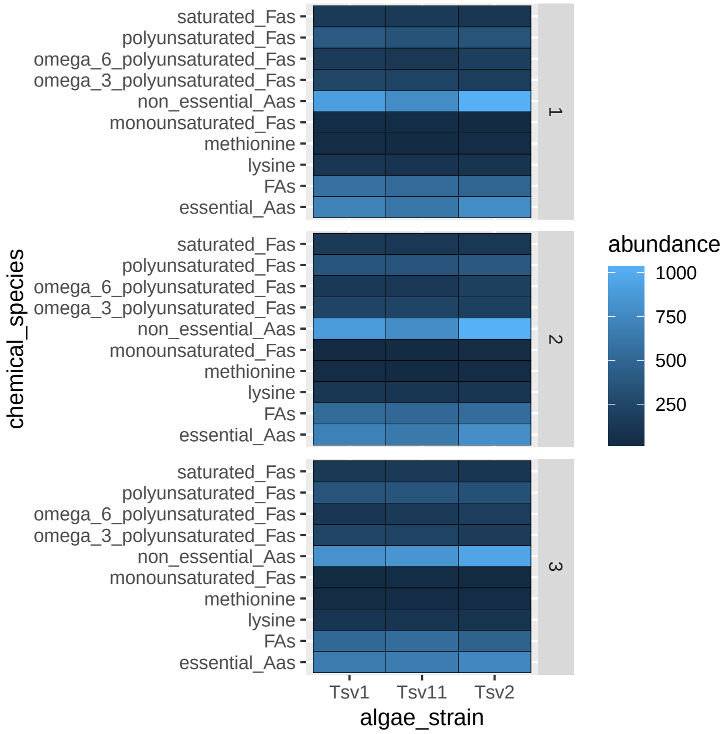
Here, we can facet in the horizontal direction:

```
ggplot(data = algae_data, aes(x = algae_strain, y = chemical_species)) +
  geom_tile(aes(fill = abundance), color = "black") +
  facet_grid(.~replicate)
```



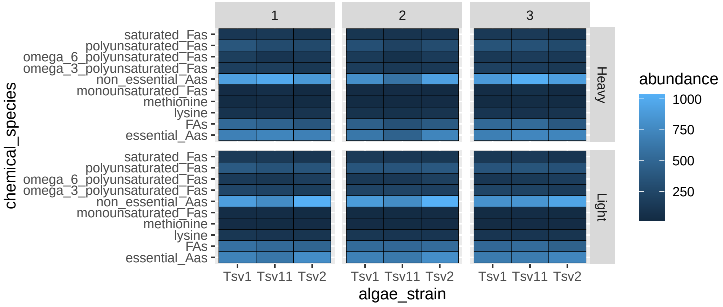
We can facet in the vertical direction:

```
ggplot(data = algae_data, aes(x = algae_strain, y = chemical_species)) +
  geom_tile(aes(fill = abundance), color = "black") +
  facet_grid(replicate~.)
```



And we can do both at the same time:

```
ggplot(data = algae_data, aes(x = algae_strain, y = chemical_species)) +
  geom_tile(aes(fill = abundance), color = "black") +
  facet_grid(harvesting_regime~replicate)
```

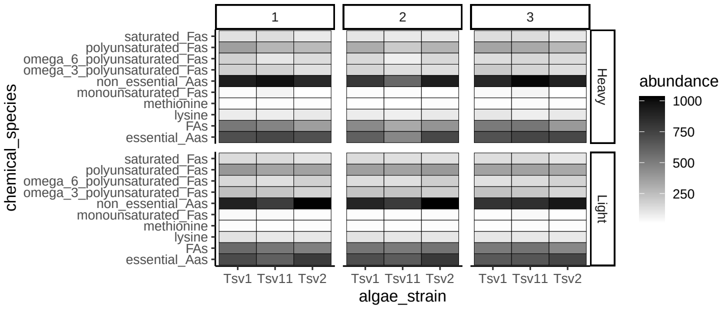


Faceting is a great way to describe more variation in your plot without having to make your geoms more complicated. For situations where you need to generate lots and lots of facets, consider `facet_wrap` instead of `facet_grid`.

0.3.3 scales

Every time you define an aesthetic mapping (e.g. aes(x = algae_strain)), you are defining a new scale that is added to your plot. You can control these scales using the `scale_*` family of commands. Consider our faceting example above. In it, we use `geom_tile(aes(fill = abundance))` to map the abundance variable to the fill aesthetic of the tiles. This creates a scale called `fill` that we can adjust using `scale_fill_*`. In this case, fill is mapped to a continuous variable and so the fill scale is a color gradient. Therefore, `scale_fill_gradient()` is the command we need to change it. Remember that you could always type `?scale_fill_` into the console and it will help you find relevant help topics that will provide more detail. Another option is to google: "How to modify color scale ggplot geom_tile", which will undoubtedly turn up a wealth of help.

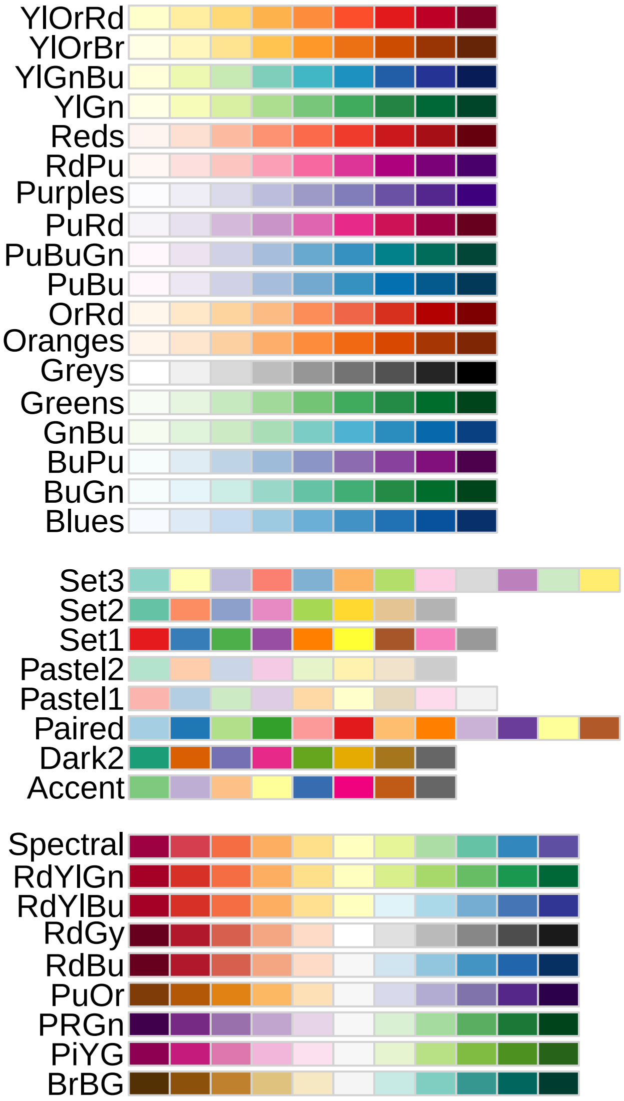
```
ggplot(data = algae_data, aes(x = algae_strain, y = chemical_species)) +
  geom_tile(aes(fill = abundance), color = "black") +
  facet_grid(harvesting_regime~replicate) +
  scale_fill_gradient(low = "white", high = "black") +
  theme_classic()
```



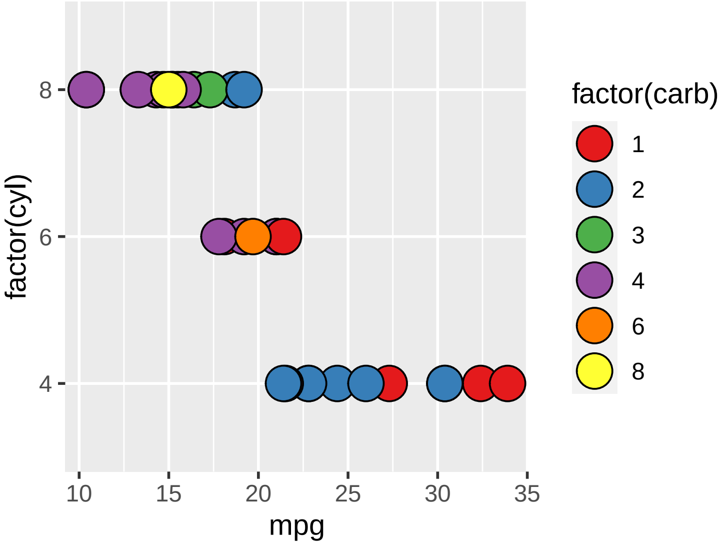
0.3.3.1 RColorBrewer

One particularly useful type of scale are those provided by RColorBrewer:

```
display.brewer.all()
```



```
ggplot(mtcars) +
  geom_point(
    aes(x = mpg, y = factor(cyl), fill = factor(carb)),
    shape = 21, size = 6
  ) +
  scale_fill_brewer(palette = "Set1")
```



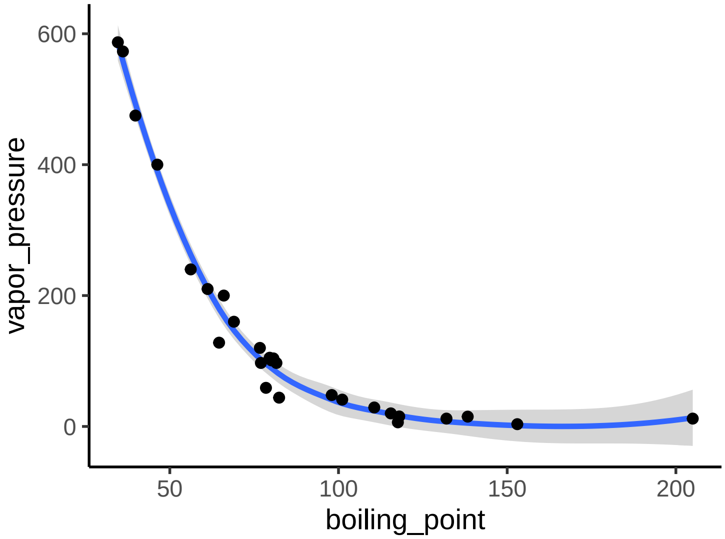
0.3.4 themes

So far we've just looked at how to control the means by which your *data* is represented on the plot. There are also components of the plot that are, strictly speaking, not *data* per se, but rather non-data ink. These are controlled using the `theme()` family of commands. There are two ways to go about this.

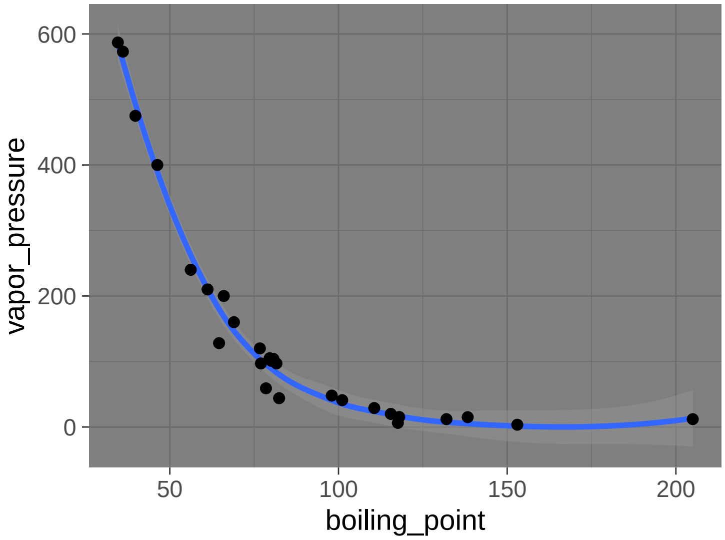
0.3.4.1 Complete themes

`ggplot` comes with a handful of built in "complete themes". These will change the appearance of your plots with respect to the non-data ink. Compare the following plots:

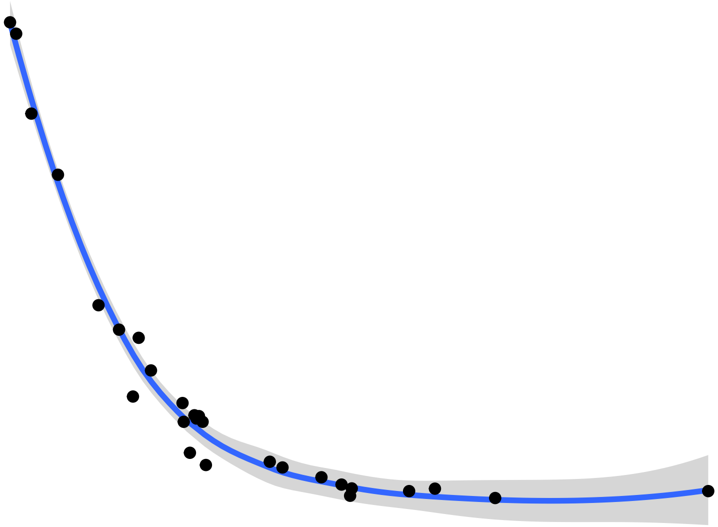
```
ggplot(data = solvents, aes(x = boiling_point, y = vapor_pressure)) +
  geom_smooth() +
  geom_point() +
  theme_classic()
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```



```
ggplot(data = solvents, aes(x = boiling_point, y = vapor_pressure)) +
  geom_smooth() +
  geom_point() +
  theme_dark()
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```



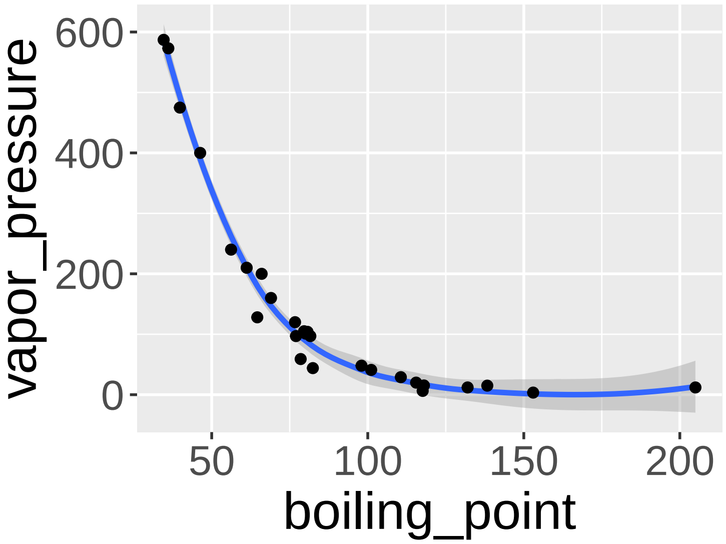
```
ggplot(data = solvents, aes(x = boiling_point, y = vapor_pressure)) +
  geom_smooth() +
  geom_point() +
  theme_void()
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```



0.3.4.2 Theme components

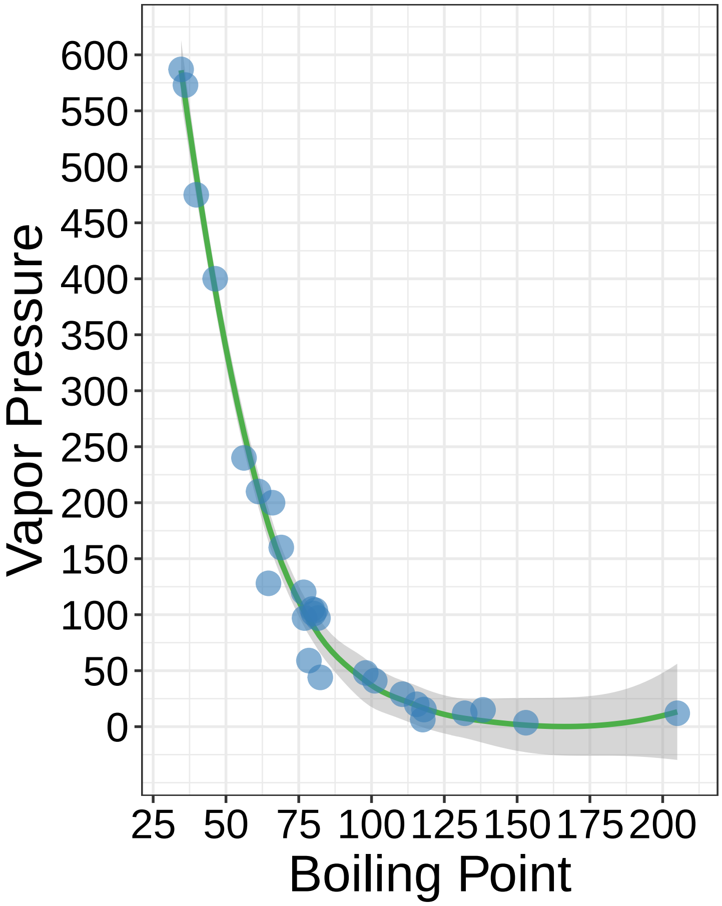
You can also change individual components of themes. This can be a bit tricky, but it's all explained if you run `?theme()`. Hare is an example (and google will provide many, many more).

```
ggplot(data = solvents, aes(x = boiling_point, y = vapor_pressure)) +
  geom_smooth() +
  geom_point() +
  theme(
    text = element_text(size = 20, color = "black")
  )
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```



Last, here is an example of combining `scale_*` and `theme*` with previous commands to really get a plot looking sharp.

```
ggplot(data = solvents, aes(x = boiling_point, y = vapor_pressure)) +
  geom_smooth(color = "#4daf4a") +
  scale_x_continuous(
    name = "Boiling Point", breaks = seq(0,200,25), limits = c(30,210)
  ) +
  scale_y_continuous(
    name = "Vapor Pressure", breaks = seq(0,600,50)
  ) +
  geom_point(color = "#377eb8", size = 4, alpha = 0.6) +
  theme_bw() +
  theme(
    axis.text = element_text(color = "black"),
    text = element_text(size = 20, color = "black")
  )
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```



0.3.5 exercises

In this set of exercises we're going to practice making more plots using the dataset `solvents`. Since you are now familiar with filtering and plotting data, the prompts in this assignment are going to be relatively open ended - I do not care what variables you map to x, y, fill, color, etc. Rather, I expect your submission to demonstrate to me that you have explored each of the new topics covered in the previous chapter. This includes geoms beyond `geom_point()` and `geom_violin()`, facets, scale modifications, and theme adjustments. Be creative! Explore the solvents dataset. Find something interesting! **Show me that you have mastered this material.** Don't forget about the ggplot cheat sheet (see the "Links" section in this book).

As before, for these exercises, you will write your code and answers to any questions in the Script Editor window of your RStudio as an R Markdown document. You will compile that file as a pdf and submit it on Canvas. If you have any questions please let me know.

Some pointers:

- If your code goes off the page, don't be afraid to wrap it across multiple lines, as shown in some of the examples in the previous set of exercises.
- Don't be afraid to put the variable with the long elements / long text on the y-axis and the continuous variable on the x-axis.

1. Create two plots that are identical except that one uses the `scales = "free"` feature of `facet_grid` while the other does not (i.e. one should use `facet_grid(<things>)`, whiel the other uses `facet_grid(<things>, scales = "free")`). Write a single caption that describes *both* plots, highlighting the advantages provided by each plot over the other. For additional tips on writing captions, please see the "Writing" chapter in this book.
2. Create two plots that are identical except that one uses `geom_point()`, while the other uses `geom_jitter()`. Write a single caption that describes *both* plots. The caption should highlight the differences bewteen these two plots and it should describe case(s) in which you think it would be appropriate to use `geom_jitter()` over `geom_point()`.
3. Make a plot that has four aesthetic mappings (x and y mappings count). Use the `scales_*` family of commands to modify some aspect of each scale create by the four mappings. Hint: some scales are somewhat tricky to modify (alpha, linetype, …), and some scales are easier to modify (x, y, color, fill, shape).
4. Make a plot and manually modify at least three aspects of its theme (i.e. do not use one of the build in complete themes such as `theme_classic()`, rather, manually modify components of the theme using `theme()`). This means that inside your `theme()` command, there should be three arguments separated by commas.
5. Identify a relationship between two variables in the dataset. Create a plot that is optimized (see note) to highlight the features of this relationship. Write a short caption that describes the plot *and* the trend you've identified and highlighted. Note: I realize that the word "optimize" is not clearly defined here. That's ok! You are the judge of what is optimized and what is not. Use your caption to make a case for *why* your plot is optimized. *Defend* your ideas with argument!

0.4 import and tidying

0.4.1 data import

To analyze data that is stored on your own computer you can indeed import it into RStudio.

The easiest way to do this is to use the interactive command `readCSV()`, a function that comes with the phylochemistry source command. You run `readCSV()` in your console, then navigate to the data on your hard drive.

Another option is to read the data in from a path. For this, you will need to know the "path" to your data file. This is essentially the street address of your data on your computer's hard drive. Paths look different on Mac and PC.

- On Mac: `/Users/lucasbusta/Documents/sample_data_set.csv` (note the forward slashes!)
- On PC: `C:\\My Computer\\Documents\\sample_data_set.csv` (note double backward slashes!)

You can quickly find paths to files via the following:

- On Mac: Locate the file in Finder. Right-click on the file, hold the Option key, then click "Copy as Pathname"
- On PC: Locate the file in Windows Explorer. Hold down the Shift key then right-click on the file. Click "Copy As Path"

With these paths, we can read in data using the `read_csv` command. We'll run `read_csv("<path_to_your_data>")`. Note the use of QUOTES `""`! Those are necessary. Also make sure your path uses the appropriate direction of slashes for your operating system.

0.4.2 tidy data

https://tidydatatutor.com/vis.html

When we make data tables by hand, it's often easy to make a **wide-style table** like the following. In it, the abundances of 7 different fatty acids in 10 different species are tabulated. Each fatty acid gets its own row, each species, its own column.

```
fadb_sample
## # A tibble: 7 x 11
##   fatty_acid         Agonandra_brasiliensis Agonandra_silva~
##   <chr>                               <dbl>            <dbl>
## 1 Hexadecanoic acid                     3.4              1  
## 2 Octadecanoic acid                     6.2              0.1
## 3 Eicosanoic acid                       4.7              3.5
## 4 Docosanoic acid                      77.4              0.4
## 5 Tetracosanoic acid                    1.4              1
## 6 Hexacosanoic acid                     1.9             12.6
## 7 Octacosanoic acid                     5               81.4
## # ... with 8 more variables: Agonandra_excelsa <dbl>,
## #   Heisteria_silvianii <dbl>, Malania_oleifera <dbl>,
## #   Ximenia_americana <dbl>, Ongokea_gore <dbl>,
## #   Comandra_pallida <dbl>, Buckleya_distichophylla <dbl>,
## #   Nuytsia_floribunda <dbl>
```

While this format is very nice for filling in my hand (such as in a lab notebook or similar), it does not groove with ggplot and other `tidyverse` functions very well. We need to convert it into a **long-style table**. This is done using `pivot_longer()`. You can think of this function as transforming both your data's column names (or some of the column names) and your data matrix's values (in this case, the measurements) each into their own variables (i.e. columns). We can do this for our fatty acid dataset using the command below. In it, we specify what data we want to transform (`data = fadb_sample`), we need to tell it what columns we want to transform (`cols = 2:11`), what we want the new variable that contains column names to be called (`names_to = "plant_species"`) and what we want the new variable that contains matrix values to be called (`values_to = "relative_abundance"`). All together now:

```
pivot_longer(data = fadb_sample, cols = 2:11, names_to = "plant_species", values_to = "rel
## # A tibble: 70 x 3
##    fatty_acid        plant_species           relative_abunda~
##    <chr>             <chr>                              <dbl>
##  1 Hexadecanoic acid Agonandra_brasiliensis               3.4
##  2 Hexadecanoic acid Agonandra_silvatica                  1
##  3 Hexadecanoic acid Agonandra_excelsa                    1.2
##  4 Hexadecanoic acid Heisteria_silvianii                  2.9
##  5 Hexadecanoic acid Malania_oleifera                     0.7
##  6 Hexadecanoic acid Ximenia_americana                    3.3
##  7 Hexadecanoic acid Ongokea_gore                         1
##  8 Hexadecanoic acid Comandra_pallida                     2.3
##  9 Hexadecanoic acid Buckleya_distichophylla              1.6
## 10 Hexadecanoic acid Nuytsia_floribunda                   3.8
## # ... with 60 more rows
```

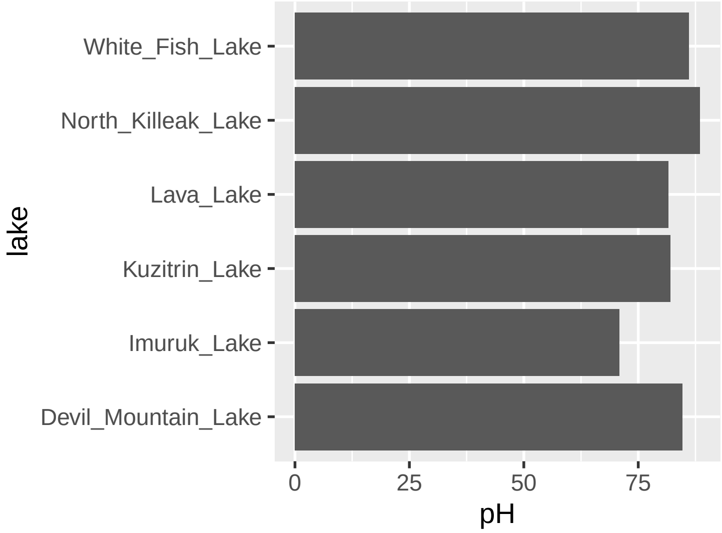
Brilliant! Now we have a tidy, long-style table that can be used with ggplot.

0.5 the pipe and summaries

0.5.1 the pipe (%>%)

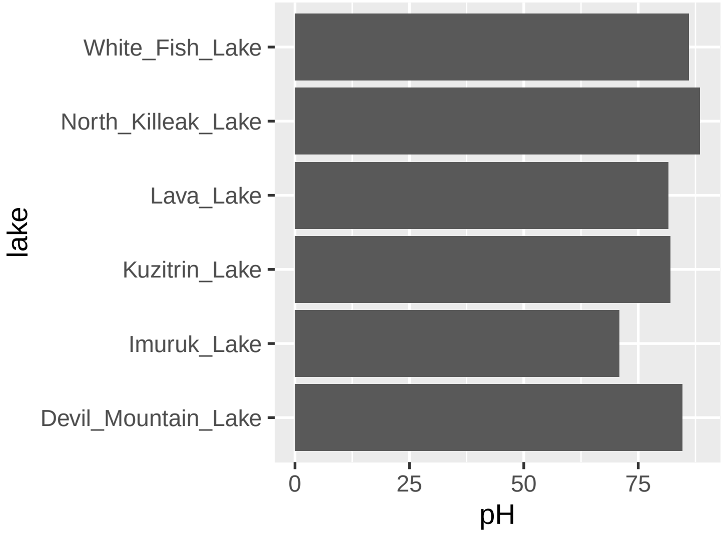
We have seen how to create new objects using <-, and we have been filtering and plotting data using, for example:

```
ggplot(filter(alaska_lake_data, park == "BELA"), aes(x = pH, y = lake)) + geom_col()
```



However, as our analyses get more complex, the code can get long and hard to read. We're going to use the pipe %>% to help us with this. Check it out:

```
alaska_lake_data %>%
  filter(park == "BELA") %>%
  ggplot(aes(x = pH, y = lake)) + geom_col()
```



Neat! Another way to think about the pipe:

The pipe will become more important as our analyses become more sophisticated, which happens very quickly when we start working with summary statistics, as we shall now see…

0.5.2 summary statistics

So far, we have been plotting raw data. This is well and good, but it is not always suitable. Often we have scientific questions that cannot be answered by looking at raw data alone, or sometimes there is too much raw data to plot. For this, we need summary statistics - things like averages, standard deviations, and so on. While these metrics can be computed in Excel, programming such can be time consuming, especially for group statistics. Consider the example below, which uses the `ny_trees` dataset. The NY Trees dataset contains information on nearly half a million trees in New York City (this is after considerable filtering and simplification):

```
ny_trees
## # A tibble: 378,762 x 14
##    tree_height tree_diameter address      tree_loc pit_type
##          <dbl>         <dbl> <chr>        <chr>    <chr>
##  1        21.1             6 1139 57 STR~ Front    Sidewalk~
##  2        59.0             6 2220 BERGEN~ Across   Sidewalk~
##  3        92.4            13 2254 BERGEN~ Across   Sidewalk~
##  4        50.2            15 2332 BERGEN~ Across   Sidewalk~
##  5        95.0            21 2361 EAST  ~ Front    Sidewalk~
##  6        67.5            19 2409 EAST  ~ Front    Continuo~
##  7        75.3            11 1481 EAST  ~ Front    Lawn
##  8        27.9             7 1129 57 STR~ Front    Sidewalk~
##  9       111.             26 2076 EAST  ~ Across   Sidewalk~
## 10        83.9            20 2025 EAST  ~ Front    Sidewalk~
## # ... with 378,752 more rows, and 9 more variables:
## #   soil_lvl <chr>, status <chr>, spc_latin <chr>,
## #   spc_common <chr>, trunk_dmg <chr>, zipcode <dbl>,
## #   boroname <chr>, latitude <dbl>, longitude <dbl>
```

More than 300,000 observations of 14 variables! That's 4.2M data points! Now, what is the average and standard deviation of the height and diameter of each tree species within each NY borough? Do those values change for trees that are in parks versus sidewalk pits?? I don't even know how one would begin to approach such questions using traditional spreadsheets. Here, we will answer these questions with ease using two new commands: `group_by()` and `summarize()`. Let's get to it.

Say that we want to know (and of course, visualize) the mean and standard deviation of the heights of each tree species in NYC. We can see that data in first few columns of the NY trees dataset above, but how to calculate these statistics? In R, mean can be computed with `mean()` and standard deviation can be calculated with `sd()`. We will use the function `summarize()` to calculate summary statistics. So, we can calculate the average and standard deviation of all the trees in the data set as follows:

```
ny_trees %>%
  summarize(mean_height = mean(tree_height))
## # A tibble: 1 x 1
##   mean_height
##         <dbl>
## 1        72.6

ny_trees %>%
  summarize(stdev_height = sd(tree_height))
## # A tibble: 1 x 1
##   stdev_height
##          <dbl>
## 1         28.7
```

Great! But how to do this for each species? We need to subdivide the data by species, then compute the mean and standard deviation, then recombine the results into a new table. First, we use `group_by()`. Note that in ny_trees, species are indicated in the column called `spc_latin`. Once the data is grouped, we can use `summarize()` to compute statistics.

```
ny_trees %>%
  group_by(spc_latin) %>%
  summarize(mean_height = mean(tree_height))
```

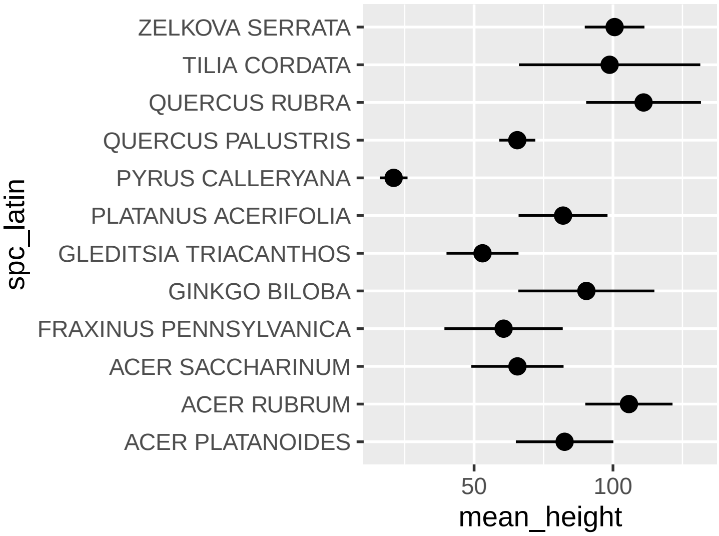
```
## # A tibble: 12 x 2
##    spc_latin              mean_height
##    <chr>                        <dbl>
##  1 ACER PLATANOIDES              82.6
##  2 ACER RUBRUM                  106.
##  3 ACER SACCHARINUM              65.6
##  4 FRAXINUS PENNSYLVANICA        60.6
##  5 GINKGO BILOBA                 90.4
##  6 GLEDITSIA TRIACANTHOS         53.0
##  7 PLATANUS ACERIFOLIA           82.0
##  8 PYRUS CALLERYANA              21.0
##  9 QUERCUS PALUSTRIS             65.5
## 10 QUERCUS RUBRA                111.
## 11 TILIA CORDATA                 98.8
## 12 ZELKOVA SERRATA              101.
```

Bam. Mean height of each tree species. `summarize()` is more powerful though, we can do many summary statistics at once:

```
ny_trees %>%
  group_by(spc_latin) %>%
  summarize(
    mean_height = mean(tree_height),
    stdev_height = sd(tree_height)
  ) -> ny_trees_by_spc_summ
ny_trees_by_spc_summ
## # A tibble: 12 x 3
##    spc_latin              mean_height stdev_height
##    <chr>                        <dbl>        <dbl>
##  1 ACER PLATANOIDES              82.6        17.6
##  2 ACER RUBRUM                  106.         15.7
##  3 ACER SACCHARINUM              65.6        16.6
##  4 FRAXINUS PENNSYLVANICA        60.6        21.3
##  5 GINKGO BILOBA                 90.4        24.5
##  6 GLEDITSIA TRIACANTHOS         53.0        13.0
##  7 PLATANUS ACERIFOLIA           82.0        16.0
##  8 PYRUS CALLERYANA              21.0         5.00
##  9 QUERCUS PALUSTRIS             65.5         6.48
## 10 QUERCUS RUBRA                111.         20.7
## 11 TILIA CORDATA                 98.8        32.6
## 12 ZELKOVA SERRATA              101.         10.7
```

Now we can use this data in plotting. For this, we will use a new geom, `geom_pointrange`, which takes `x` and `y` aesthetics, as usual, but also requires two additional y-ish aesthetics `ymin` and `ymax` (or `xmin` and `xmax` if you want them to vary along x). Also, note that in the aesthetic mappings for `xmin` and `xmax`, we can use a mathematical expression: `mean-stdev` and `mean+stdev`, respectivey. In our case, these are `mean_height - stdev_height` and `mean_height + stdev_height`. Let's see it in action:

```
ny_trees_by_spc_summ %>%
ggplot() +
  geom_pointrange(
      aes(
        y = spc_latin,
        x = mean_height,
        xmin = mean_height - stdev_height,
        xmax = mean_height + stdev_height
      )
    )
```

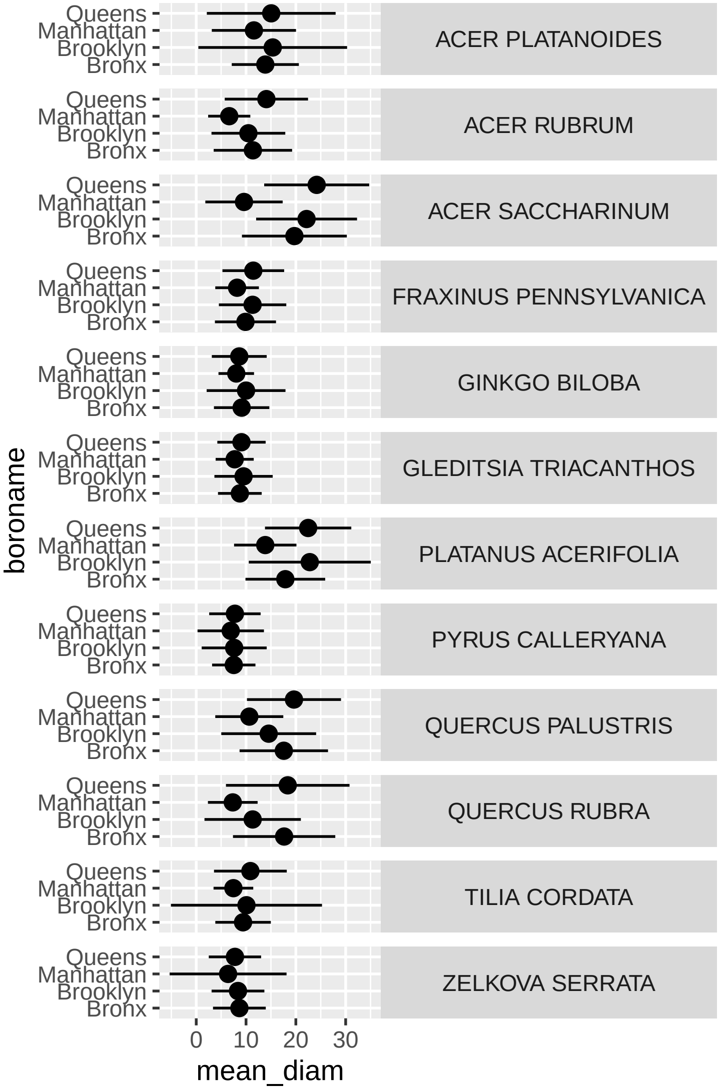


Cool! Just like that, we've found (and visualized) the average and standard deviation of tree heights, by species, in NYC. But it doesn't stop there. We can use `group_by()` and `summarize()` on multiple variables (i.e. more groups). We can do this to examine the properties of each tree species in each NYC borough. Let's check it out:

```
ny_trees %>%
  group_by(spc_latin, boroname) %>%
  summarize(
    mean_diam = mean(tree_diameter),
    stdev_diam = sd(tree_diameter)
  ) -> ny_trees_by_spc_boro_summ
ny_trees_by_spc_boro_summ
## # A tibble: 48 x 4
## # Groups:   spc_latin [12]
##    spc_latin        boroname  mean_diam stdev_diam
##    <chr>            <chr>         <dbl>      <dbl>
##  1 ACER PLATANOIDES Bronx         13.9        6.74
##  2 ACER PLATANOIDES Brooklyn      15.4       14.9
##  3 ACER PLATANOIDES Manhattan     11.6        8.45
##  4 ACER PLATANOIDES Queens        15.1       12.9
##  5 ACER RUBRUM      Bronx         11.4        7.88
##  6 ACER RUBRUM      Brooklyn      10.5        7.41
##  7 ACER RUBRUM      Manhattan      6.63       4.23
##  8 ACER RUBRUM      Queens        14.1        8.36
##  9 ACER SACCHARINUM Bronx         19.7       10.5
## 10 ACER SACCHARINUM Brooklyn      22.2       10.1
## # ... with 38 more rows
```

Now we have summary statistics for each tree species within each borough. This is different from the previous plot in that we now have an additional variable (boroname) in our summarized dataset. This additional variable needs to be encoded in our plot. Let's map boroname to x and facet over tree species, which used to be on x. We'll also manually modify the theme element `strip.text.y` to get the species names in a readable position.

```
ny_trees_by_spc_boro_summ %>%
ggplot() +
  geom_pointrange(
    aes(
      y = boroname,
      x = mean_diam,
      xmin = mean_diam-stdev_diam,
      xmax = mean_diam+stdev_diam
    )
  ) +
  facet_grid(spc_latin~.) +
  theme(
    strip.text.y = element_text(angle = 0)
  )
```



Excellent! And if we really want to go for something pretty:

```
ny_trees_by_spc_boro_summ %>%
ggplot() +
  geom_pointrange(
    aes(
      y = boroname,
      x = mean_diam,
      xmin = mean_diam-stdev_diam,
      xmax = mean_diam+stdev_diam,
      fill = spc_latin
    ), color = "black", shape = 21
  ) +
  labs(
    y = "Borough",
    x = "Trunk diameter",
    caption = str_wrap("Figure 1: Diameters of trees in New York City. Points correspond t
  ) +
  facet_grid(spc_latin~.) +
  guides(fill = "none") +
  scale_fill_brewer(palette = "Paired") +
  theme_bw() +
  theme(
    strip.text.y = element_text(angle = 0),
    plot.caption = element_text(hjust = 0.5)
  )
```

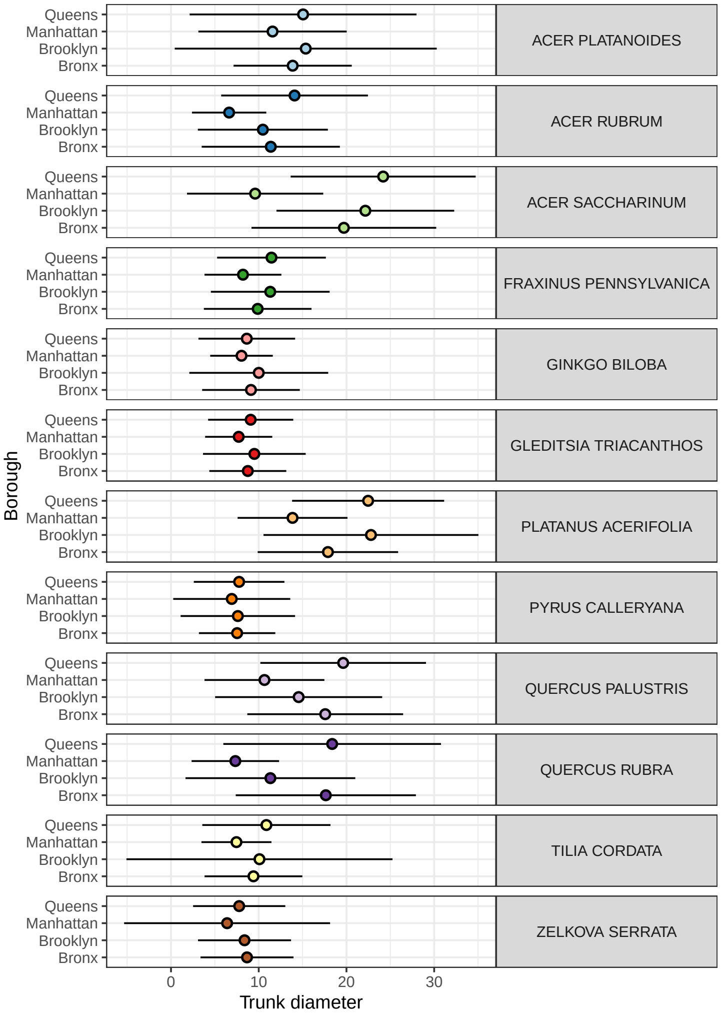


Figure 1: Diameters of trees in New York City. Points correspond to average diameters of each tree species in each borough. Horizontal lines indicate the standard deviation of tree diameters. Points are colored according to tree species.

Now we are getting somewhere. It looks like there are some really big maple trees (Acer) in Queens.

0.5.3 exercises

Isn't seven the most powerfully magical number? *Isn't seven the most powerfully magical number?* Yes… I think the idea of a seven-part assignment would greatly appeal to an alchemist.

In this set of exercises we are going to use the periodic table. After you run source() you can load that dataset using `periodic_table`. Please use that dataset to run analyses and answer the following quetions/prompts. Compile the answers in an R Markdown document, compile it as a pdf, and upload it to the Canvas assignment. Please let me know if you have any questions. Good luck, and have fun!

Some pointers:

- If your code goes off the page, don't be afraid to wrap it across multiple lines, as shown in some of the examples in the previous set of exercises.
- Don't be afraid to put the variable with the long elements / long text on the y-axis and the continuous variable on the x-axis.

1. Make a plot using `geom_point()` that shows the average atomic weight of the elements discovered in each year spanned by the dataset (i.e. what was the average weight of the elements discovered in 1900? 1901? 1902? etc.). You should see a trend, particularly after 1950. What do you think has caused this trend?
2. The column `state_at_RT` indicates the state of each element at room temperate. Make a plot that shows the average first ionization potential of all the elements belonging to each state group indicated in `state_at_RT` (i.e. what is the average 1st ionization potential of all elements that are solid at room temp? liquid? etc.). Which is the highest?
3. Filter the dataset so that only elements with atomic number less than 85 are included. Considering only these elements, what is the average and standard deviation of boiling points for each type of `crystal_structure`? Make a plot using `geom_pointrange()` that shows the mean and standard deviation of each of these groups. What's up with elements that have a cubic crystal structure?
4. Now filter the original dataset so that only elements with atomic number less than 37 are considered. The elements in this dataset belong to the first four periods. What is the average abundance of each of these four *periods* in seawater? i.e. what is the average abundance of all elements from period 1? period 2? etc. Which period is the most abundant? In this context what does "CHON" mean? (not the rock band, though they are also excellent, especially that song that features GoYama)
5. Now filter the original dataset so that only elements with atomic number less than 103 are considered. Filter it further so that elements from group number 18 are excluded. Using this twice-filtered dataset, compute the average, minimum, and maximum values for electronegativiy for each `group_number`. Use `geom_point()` and

geom_errorbar() to illustrate the average, minimum, and maximum values for each group number.

6. Filter the dataset so that only elements with atomic number less than 85 are considered. Group these by color. Now filter out those that have color == "colorless". Of the remaining elements, which has the widest range of specific heats? Use geom_point() and geom_errorbar() to illustrate the mean and standard deviation of each color's specific heats.

7. You have learned many things in this course so far. filter(), ggplot(), and now group_by() and summarize(). Using **all** these commands, create a graphic to illustrate what you consider to be an interesting periodic trend. Use theme elements and scales to enhance your plot: impress me!

Part II

chemometrics

chemometrics

0.6 clustering

0.6.1 theory

"Which of my samples are most closely related?"

So far we have been looking at how to plot raw data, as well as data that has been summarize across samples. This is important stuff and very useful. However, we often have questions about how samples in our datasets relate to one another. For example: in the Alaska lakes dataset, which lake is most similar, chemically speaking, to Lake Narvakrak? Answering this requires calculating numeric distances between samples based on their chemical properties. For this, and other questions, we need to use matrix analyses. For this, we will use `runMatrixAnalysis()`, a function that is loaded into your R Session when you run the source() command.

Matrix analyses can be a bit difficult to set up. There are two things that we can do to help us with this: (i) we will use a template for `runMatrixAnalysis()` (see below) and (ii) it is *critical* that we think about our data in terms of **samples** and **analytes**. Let's consider our Alaska lakes data set:

```
alaska_lake_data
## # A tibble: 220 x 7
##    lake                park  water_temp    pH element mg_per_L
##    <chr>               <chr>      <dbl> <dbl> <chr>      <dbl>
##  1 Devil_Mountain_Lake BELA        6.46  7.69 C          3.4  
##  2 Devil_Mountain_Lake BELA        6.46  7.69 N          0.028
##  3 Devil_Mountain_Lake BELA        6.46  7.69 P          0    
##  4 Devil_Mountain_Lake BELA        6.46  7.69 Cl        10.4  
##  5 Devil_Mountain_Lake BELA        6.46  7.69 S          0.62 
##  6 Devil_Mountain_Lake BELA        6.46  7.69 F          0.04 
##  7 Devil_Mountain_Lake BELA        6.46  7.69 Br         0.02 
##  8 Devil_Mountain_Lake BELA        6.46  7.69 Na         8.92 
##  9 Devil_Mountain_Lake BELA        6.46  7.69 K          1.2  
## 10 Devil_Mountain_Lake BELA        6.46  7.69 Ca         5.73 
## # ... with 210 more rows, and 1 more variable:
## #   element_type <chr>
```

We can see that this dataset is comprised of measurements of various *analytes* (i.e. several chemical elements, as well as water_temp, and pH), in different *samples* (i.e. lakes). We need to tell the `runMatrixAnalysis()` function how each column relates to this samples and analytes structure. See the image below for an explanation.

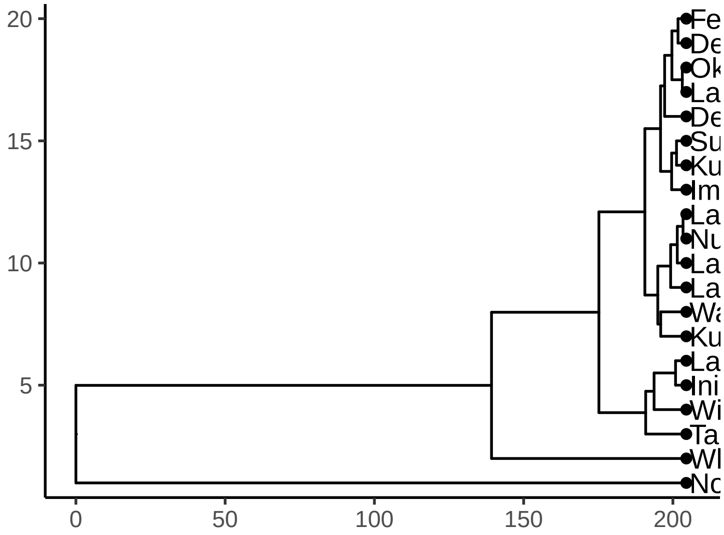
With this in mind, let's try out our template:

```
AK_lakes_clustered <- runMatrixAnalysis(

    data = alaska_lake_data,

    analysis = "hclust",

    column_w_names_of_multiple_analytes = "element",
    column_w_values_for_multiple_analytes = "mg_per_L",

    columns_w_values_for_single_analyte = c("water_temp", "pH"),

    columns_w_additional_analyte_info = "element_type",

    columns_w_sample_ID_info = c("lake", "park")

)
AK_lakes_clustered
## # A tibble: 39 x 25
##    sample_unique_ID  lake   park  parent  node branch.length
##    <chr>             <chr>  <chr>  <int> <int>         <dbl>
##  1 Devil_Mountain_L~ Devil~ BELA      33     1          7.25
##  2 Imuruk_Lake_BELA  Imuru~ BELA      32     2          4.91
##  3 Kuzitrin_Lake_BE~ Kuzit~ BELA      36     3          3.27
##  4 Lava_Lake_BELA    Lava_~ BELA      35     4          3.02
##  5 North_Killeak_La~ North~ BELA      21     5        204.  
##  6 White_Fish_Lake_~ White~ BELA      22     6         65.2 
##  7 Iniakuk_Lake_GAAR Iniak~ GAAR      29     7          3.60
##  8 Kurupa_Lake_GAAR  Kurup~ GAAR      31     8          8.57
##  9 Lake_Matcharak_G~ Lake_~ GAAR      29     9          3.60
## 10 Lake_Selby_GAAR   Lake_~ GAAR      30    10          5.24
## # ... with 29 more rows, and 19 more variables:
## #   label <chr>, isTip <lgl>, x <dbl>, y <dbl>,
## #   branch <dbl>, angle <dbl>, water_temp <dbl>, pH <dbl>,
## #   C <dbl>, N <dbl>, P <dbl>, Cl <dbl>, S <dbl>, F <dbl>,
## #   Br <dbl>, Na <dbl>, K <dbl>, Ca <dbl>, Mg <dbl>
```

It works! Now we can plot our cluster diagram with a ggplot add-on called ggtree. We've seen that ggplot takes a "data" argument (i.e. `ggplot(data = <some_data>) + geom_*()` etc.). In contrast, ggtree takes an argument called `tr`, though if you're using the `runMatrixAnalysis()` function, you can treat these two (`data` and `tr`) the same, so, use: `ggtree(tr = <output_from_runMatrixAnalysis>) + geom_*()` etc.

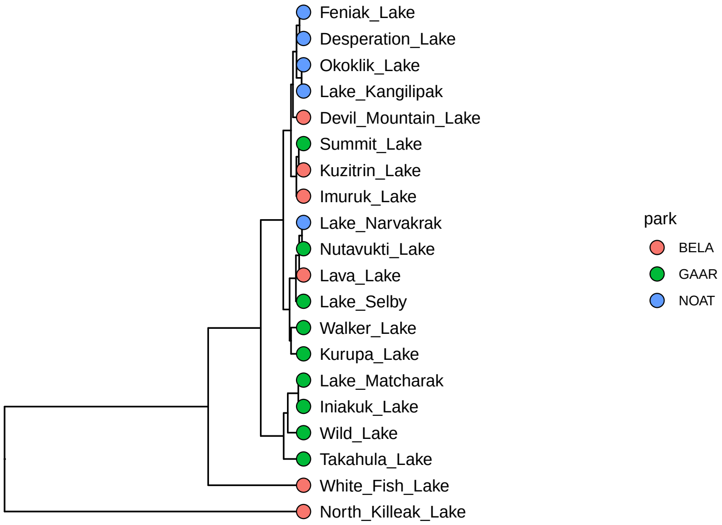
Note that `ggtree` also comes with several great new geoms: `geom_tiplab()` and `geom_tippoint()`. Let's try those out:

```
library(ggtree)
AK_lakes_clustered %>%
ggtree() +
  geom_tiplab() +
  geom_tippoint() +
  theme_classic()
```



Cool! Though that plot could use some tweaking... let's try:

```
AK_lakes_clustered %>%
ggtree() +
    geom_tiplab(aes(label = lake), offset = 10) +
    geom_tippoint(shape = 21, aes(fill = park), size = 4) +
    scale_x_continuous(limits = c(0,400))
```



Very nice!

0.6.2 exercises

For this set of exercises, please use `runMatrixAnalysis()` to run and visualize a hierarchical cluster analysis with each of the main datasets that we have worked with so far, except for NY_trees. This means: `algae_data` (which algae strains are most similar to each other?), `alaska_lake_data` (which lakes are most similar to each other?). and `solvents` (which solvents are most similar to each other?). It also means you should use the periodic table (which elements are most similar to each other?), though please don't use the whole periodic table, rather, use `periodic_table_subset`. For each of these, create (i) a tree diagram that shows how the "samples" in each data set are related to each other based on the numerical data associated with them, (ii) a caption for each diagram, and (iii) describe, in two or so sentences, an interesting trend you see in the diagram. You can ignore columns that contain categorical data, or you can list those columns as "additional_analyte_info".

For this assignment, you may find the `colnames()` function and square bracket-subsetting useful. It will list all or a subset of the column names in a dataset for you. For example:

```
colnames(solvents)
##  [1] "solvent"             "formula"
##  [3] "boiling_point"       "melting_point"
##  [5] "density"             "miscible_with_water"
##  [7] "solubility_in_water" "relative_polarity"
##  [9] "vapor_pressure"      "CAS_number"
## [11] "formula_weight"      "refractive_index"
## [13] "specific_gravity"    "category"

colnames(solvents)[1:3]
## [1] "solvent"       "formula"       "boiling_point"

colnames(solvents)[c(1,5,7)]
## [1] "solvent"             "density"
## [3] "solubility_in_water"
```

0.7 pca

"Which analytes are driving differences among my samples?"

In addition to heirarchical clustering, there is another way to look at our data in a cluster context - i.e. another way to identify clusters of samples that have similar properties based on the analytes in the data set. This method is called k-means, which we will look at later, because for it we first need to have a look at dimensionality reduction techniques, particularly principal components analysis (PCA).

explain pca as drawing a new set of axes pca also answers questions about whether variables are related hclust is "who is most closely related to whom?", PCA is about clustering

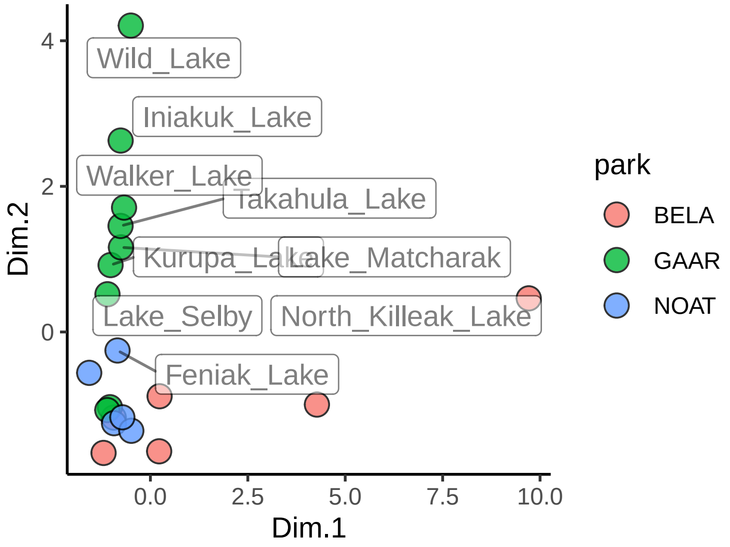
0.7.1 pca

PCA looks at all the variance in a high dimensional data set and chooses new axes within that data set that align with the directions containing highest variance. These new axes are called principal components. Let's look at an example:

In the example above, the three dimensional space can be reduced to a two dimensional space with the principal components analysis. New axes (principal components) are selected (bold arrows on left) that become the x and y axes in the principal components space (right).

We can run and visualize principal components analyses using the `runMatrixAnalysis()` function as in the example below:

```
AK_lakes_pca <- runMatrixAnalysis(
  data = alaska_lake_data,
  analysis = c("pca"),
  column_w_names_of_multiple_analytes = "element",
  column_w_values_for_multiple_analytes = "mg_per_L",
  columns_w_values_for_single_analyte = c("water_temp", "pH"),
  columns_w_additional_analyte_info = "element_type",
  columns_w_sample_ID_info = c("lake", "park")
)

ggplot(data = AK_lakes_pca, aes(x = Dim.1, y = Dim.2)) +
  geom_point(aes(fill = park), shape = 21, size = 4, alpha = 0.8) +
  geom_label_repel(aes(label = lake), alpha = 0.5) +
  theme_classic()
```



Great! In this plot we can see that White Fish Lake and North Killeak Lake, both in BELA park, are quite different from the other parks (they are separated from the others along dimension 1, i.e. the first principal component). At the same time, Wild Lake, Iniakuk Lake, Walker Lake, and several other lakes in GAAR park are different from all the others (they are separated from the others along dimension 2, i.e. the second principal component).

Important question: what makes the lakes listed above different from the others? Certainly some aspect of their chemistry, since that's the data that this analysis is built upon, but how do we determine which analyte(s) are driving the differences among the lakes that we see in the PCA plot?

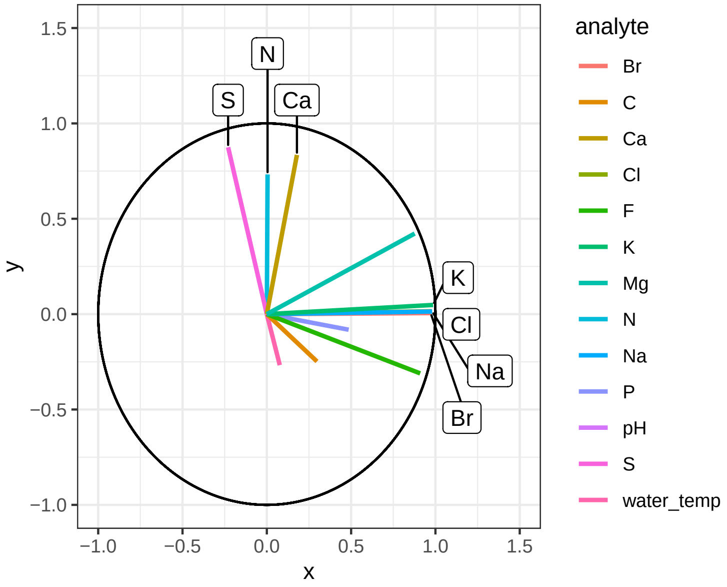
0.7.2 ordination plots

Let's look at how to access the information about which analytes are major contributors to each principal component. This is important because it will tell you which analytes are associated with particular dimensions, and by extension, which analytes are associated with (and are markers for) particular groups in the PCA plot. This can be determined using an ordination plot. Let's look at an example. We can obtain the ordination plot information using `runMatrixAnalysis()` with `analysis = "pca_ord"`:

```
## # A tibble: 6 x 3
##   analyte      Dim.1   Dim.2
##   <chr>        <dbl>   <dbl>
## 1 water_temp 0.0769  -0.267
## 2 pH         0.704    0.0190
## 3 C          0.297   -0.248
## 4 N          0.00446  0.732
## 5 P          0.485   -0.0817
## 6 Cl         0.978    0.0152
```

We can now visualize the ordination plot using our standard ggplot plotting techniques. Note the use of `geom_label_repel()` and `filter()` to label certain segments in the ordination plot. You do not need to use `geom_label_repel()`, you could use the built in `geom_label()`, but `geom_label_repel()` can make labelling your segments easier.

```
AK_lakes_pca_ord <- runMatrixAnalysis(
  data = alaska_lake_data,
  analysis = c("pca_ord"),
  column_w_names_of_multiple_analytes = "element",
  column_w_values_for_multiple_analytes = "mg_per_L",
  columns_w_values_for_single_analyte = c("water_temp", "pH"),
  columns_w_additional_analyte_info = "element_type",
  columns_w_sample_ID_info = c("lake", "park")
)
head(AK_lakes_pca_ord)
## # A tibble: 6 x 3
##   analyte      Dim.1   Dim.2
##   <chr>        <dbl>   <dbl>
## 1 water_temp 0.0769  -0.267
## 2 pH         0.704    0.0190
## 3 C          0.297   -0.248
## 4 N          0.00446  0.732
## 5 P          0.485   -0.0817
## 6 Cl         0.978    0.0152

ggplot(AK_lakes_pca_ord) +
  geom_segment(aes(x = 0, y = 0, xend = Dim.1, yend = Dim.2, color = analyte), size = 1) +
  geom_circle(aes(x0 = 0, y0 = 0, r = 1)) +
  geom_label_repel(
    data = filter(AK_lakes_pca_ord, Dim.1 > 0.9, Dim.2 < 0.1, Dim.2 > -0.1),
    aes(x = Dim.1, y = Dim.2, label = analyte), xlim = c(1,1.5)
  ) +
  geom_label_repel(
    data = filter(AK_lakes_pca_ord, Dim.2 > 0.5),
    aes(x = Dim.1, y = Dim.2, label = analyte), direction = "y", ylim = c(1,1.5)
  ) +
  coord_cartesian(xlim = c(-1,1.5), ylim = c(-1,1.5)) +
  theme_bw()
```



Great! Here is how to read the ordination plot:

1. When considering one analyte's vector: the vector's projected value on an axis shows how much its variance is aligned with that principal component.
2. When considering two analyte vectors: the angle between two vectors indicates how correlated those two variables are. If they point in the same direction, they are highly correlated. If they meet each other at 90 degrees, they are not very correlated. If they meet at ~180 degrees, they are negatively correlated. If say that one analyte is "1.9" with respect to dimension 2 and another is "-1.9" with respect to dimension 2. Let's also say that these vectors are ~"0" with respect to dimension 1.

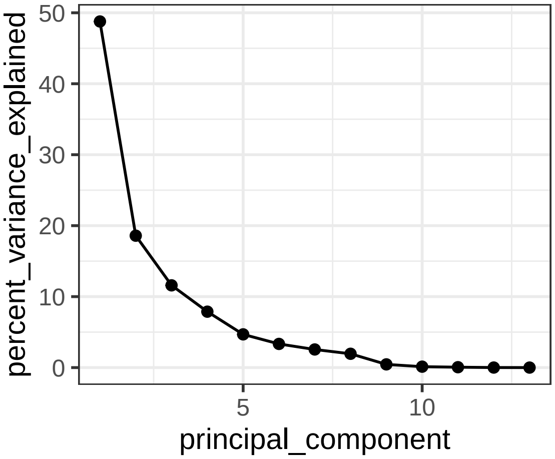
With the ordination plot above, we can now see that the abundances of K, Cl, Br, and Na are the major contributors of variance to the first principal component (or the first dimension). The abundances of these elements are what make White Fish Lake and North Killeak Lake different from the other lakes.

We can also see that the abundances of N, S, and Ca are the major contributors to variance in teh second dimension, whic means that these elements ar what set Wild Lake, Iniakuk Lake, Walker Lake, and several other lakes in GAAR park apart from the rest of the lakes in the data set.

0.7.3 principal components

We also can access information about the how much of the variance in the data set is explained by each principal component, and we can plot that using ggplot:

```
AK_lakes_pca_dim <- runMatrixAnalysis(
  data = alaska_lake_data,
  analysis = c("pca_dim"),
  column_w_names_of_multiple_analytes = "element",
  column_w_values_for_multiple_analytes = "mg_per_L",
  columns_w_values_for_single_analyte = c("water_temp", "pH"),
  columns_w_additional_analyte_info = "element_type",
  columns_w_sample_ID_info = c("lake", "park")
)
head(AK_lakes_pca_dim)
## # A tibble: 6 x 2
##   principal_component percent_variance_explained
##                 <dbl>                      <dbl>
## 1                   1                      48.8
## 2                   2                      18.6
## 3                   3                      11.6
## 4                   4                       7.88
## 5                   5                       4.68
## 6                   6                       3.33

ggplot(
  data = AK_lakes_pca_dim,
  aes(x = principal_component, y = percent_variance_explained)
) +
  geom_line() +
  geom_point() +
  theme_bw()
```



Cool! We can see that the first principal component retains nearly 50% of the variance in the original dataset, while the second dimension contains only about 20%.

0.7.4 exercises

In this set of exercises you will choose to complete ONE of the options below. For either option please refer to Chapter 12 for help with principal component and ordination plots. Also, when you are filling out the `runMatrixAnalysis()` template, you can use the `colnames()` function to help you specify a long list of column names rather than typing them out by hand. For example, in the periodic table data set, we can refer to a set of columns (columns 10 through 20) with the following command:

```
colnames(periodic_table_subset)[10:20]
##  [1] "electronegativity_pauling"
##  [2] "first_ionization_poten_eV"
##  [3] "second_ionization_poten_eV"
##  [4] "third_ionization_poten_eV"
##  [5] "electron_affinity_eV"
##  [6] "atomic_radius_ang"
##  [7] "ionic_radius_ang"
##  [8] "covalent_radius_ang"
##  [9] "atomic_volume_cm3_per_mol"
## [10] "electrical_conductivity_mho_per_cm"
## [11] "specific_heat_J_per_g_K"
colnames(periodic_table_subset)[c(18:20, 23:25)]
## [1] "atomic_volume_cm3_per_mol"
## [2] "electrical_conductivity_mho_per_cm"
## [3] "specific_heat_J_per_g_K"
## [4] "thermal_conductivity_W_per_m_K"
## [5] "polarizability_A_cubed"
## [6] "heat_atomization_kJ_per_mol"
```

We can use that command in the template, as in the example below. With the notation `colnames(periodic_table_subset)[c(5:7,9:25)]`, we can mark columns 5 - 7 and 9 - 25 as columns_w_values_for_single_analyte (note what happens when you run `c(5:7,9:25)` in the console, and what happens when you run `colnames(periodic_table_subset)[c(5:7,9:25)]` in the console). With the notation `colnames(periodic_table_subset)[c(1:4, 8)]` we can mark columns 1 - 4 and column 8 as columns_w_sample_ID_info (note what happens when you run `c(1:4, 8)` in the console, and what happens when you run `colnames(periodic_table_subset)[c(1:4, 8)]` in the console).

0.7.4.1 option 1: human metabolomics.

This first option is to work with a dataset describing metabolomics data (i.e. abundances of > 100 different biochemicals) from each of 93 human patients, some of which have Chronic Kidney Disease. If you choose this option, your task is to discover a biomarker for Chronic Kidney Disease. This means that you will need to determine a metabolite whose abundance is strongly associated with the disease. To do this you should complete the following:

1. Run a PCA analysis on `metabolomics_data` (i.e. `runMatrixAnalysis()` with `analysis = "pca"`)
2. Plot the results of the analysis to determine which principal component (i.e. dimension) separates the healthy and kidney_disease samples.
3. Obtain the ordination plot coordinates for the analytes in the PCA analysis (i.e. `runMatrixAnalysis()` with `analysis = "pca_ord"`).
4. Visualize the ordination plot and determine which of the analytes are strongly associated with the principal component (i.e. dimension) separates the healthy and kidney_disease samples.
5. Bingo! These analytes are associated with Chronic Kidney Disease and could be biomarkers for such.

0.7.4.2 option 2: grape vine chemistry

This second option is to work with a dataset describing metabolomics data (i.e. abundances of > 100 different biochemicals) from 5 different wine grape varieties. If you choose this option, your task is to discover a biomarker for Chardonnay and a biomarker for Cabernet Sauvignon. This means that you will need to identify two metabolites, each of which are associated with one of those two grape varieties. To do this you should complete the following:

1. Run a PCA analysis on `wine_grape_data` (i.e. `runMatrixAnalysis()` with `analysis = "pca"`)
2. Plot the results of the analysis to determine which principal component (i.e. dimension) separates the Chardonnay samples from the other varieties and the Cabernet Sauvignon samples from the other varieties.
3. Obtain the ordination plot coordinates for the analytes in the PCA analysis (i.e. `runMatrixAnalysis()` with `analysis = "pca_ord"`).
4. Visualize the ordination plot and determine which of the analytes are strongly associated with the principal component (i.e. dimension) separates the Chardonnay samples from the other varieties and the Cabernet Sauvignon samples from the other varieties.
5. Bingo! These analytes are associated with those varieites and could be biomarkers for such.

0.8 k-means

"Do my samples fall into definable clusters?"

0.8.1 theory

One of the questions we've been asking is "which of my samples are most closely related?". We've been answering that question using clustering. However, now that we know how to run principal components analyses, we can use another approach. This alternative approach is called k-means, and can help us decide how to assign our data into clusters. It is generally desirable to have a small number of clusters, however, this must be balanced by not having the variance within each cluster be too big. To strike this balance point, the elbow method is used. For it, we must first determine the maximum within-group variance at each possible number of clusters. An illustration of this is shown in **A** below:

One we know within-group variances, we find the "elbow" point - the point with minimum angle theta - thus picking the outcome with a good balance of cluster number and within-cluster variance (illustrated above in **B** and **C**.)

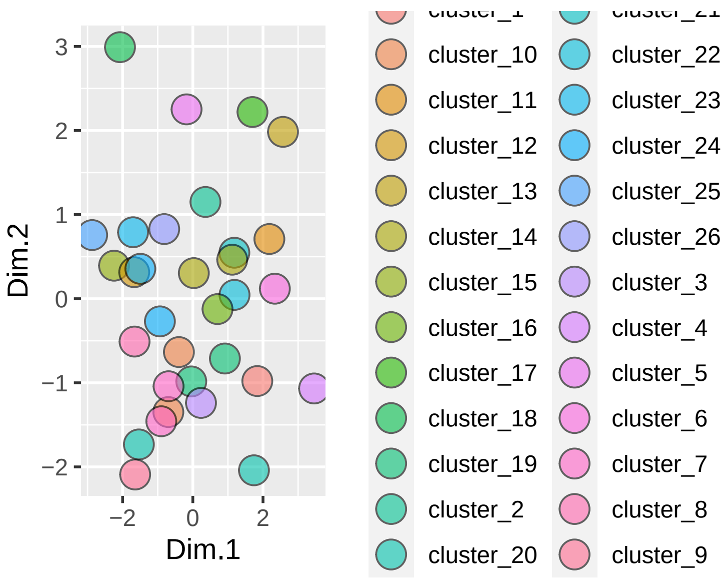
Let's try k-means using `runMatrixAnalysis`. We can use it in conjunction with `analysis = "pca"` or `analysis = "hclust"`. Let's do PCA first. To include k-means, we can just set `kmeans = "auto"`. It's important to note that kmeans cannot handle NAs. We must set something for the `na_replacement` argument. One solution is to ignore variables that have NAs for some values, which can be done by setting `na_replacement = "drop"`.

With `kmeans = "auto"` and `na_replacement = "drop"`, we can now run our analyssis. The output now has an additional column called `kmeans_cluster`, which indicates what cluster each sample is in:

```
solvents_pca_kmeans <- runMatrixAnalysis(
  data = solvents,
  analysis = c("pca"),
  column_w_names_of_multiple_analytes = NULL,
  column_w_values_for_multiple_analytes = NULL,
  columns_w_values_for_single_analyte = colnames(solvents)[c(3:5, 7:9, 11:12)],
  columns_w_additional_analyte_info = NULL,
  columns_w_sample_ID_info = c("solvent", "formula", "miscible_with_water", "CAS_number",
  transpose = FALSE,
  kmeans = "auto",
  na_replacement = "drop"
)
## Dropping any variables in your dataset that have NA as a value.
## Variables dropped:
## solubility_in_water vapor_pressure

solvents_pca_kmeans
## # A tibble: 32 x 15
##    sample_unique_ID       solvent   formula miscible_with_w~
##    <chr>                  <chr>     <chr>   <lgl>
##  1 acetic_acid_C2H4O2_TR~ acetic_a~ C2H4O2  TRUE
##  2 acetone_C3H6O_TRUE_67~ acetone   C3H6O   TRUE
##  3 acetonitrile_C2H3N_TR~ acetonit~ C2H3N   TRUE
##  4 benzene_C6H6_FALSE_71~ benzene   C6H6    FALSE
##  5 benzonitrile_C7H5N_FA~ benzonit~ C7H5N   FALSE
##  6 1-butanol_C4H10O_FALS~ 1-butanol C4H10O  FALSE
##  7 2-butanone_C4H8O_FALS~ 2-butano~ C4H8O   FALSE
##  8 carbon_disulfide_CS2_~ carbon_d~ CS2     FALSE
##  9 carbon_tetrachloride_~ carbon_t~ CCl4    FALSE
## 10 chlorobenzene_C6H5Cl_~ chlorobe~ C6H5Cl  FALSE
## # ... with 22 more rows, and 11 more variables:
## #   CAS_number <chr>, category <chr>, Dim.1 <dbl>,
## #   Dim.2 <dbl>, kmeans_cluster <chr>, boiling_point <dbl>,
## #   melting_point <dbl>, density <dbl>,
## #   relative_polarity <dbl>, formula_weight <dbl>,
## #   refractive_index <dbl>
```

We can plot the results and color them according to the group that kmeans suggested:

```
ggplot(solvents_pca_kmeans) +
  geom_point(aes(x = Dim.1, y = Dim.2, fill = kmeans_cluster), shape = 21, size = 5, alpha
```

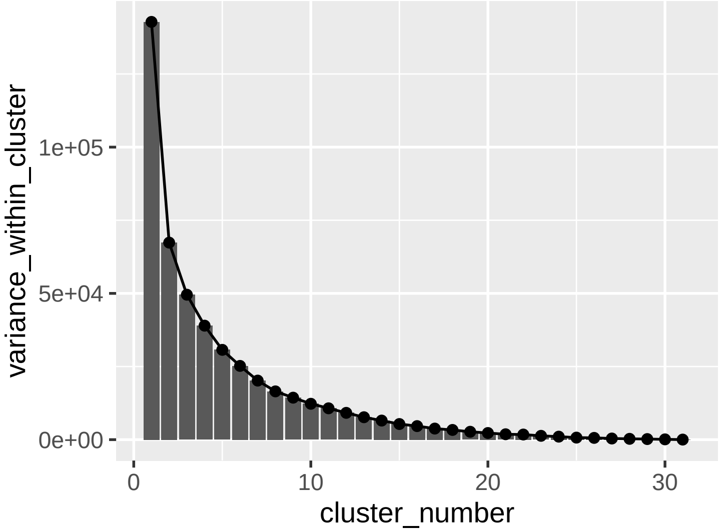


Hmmm, it looks like the elbow algorithm is suggesting lots of clusters. Why is this? Let's look at the elbow plot itself. For this, we can just set `kmeans = "elbow"`:

```
solvents_pca_kmeans_elbow <- runMatrixAnalysis(
  data = solvents,
  analysis = c("pca"),
  column_w_names_of_multiple_analytes = NULL,
  column_w_values_for_multiple_analytes = NULL,
  columns_w_values_for_single_analyte = colnames(solvents)[c(3:5, 7:9, 11:12)],
  columns_w_additional_analyte_info = NULL,
  columns_w_sample_ID_info = c("solvent", "formula", "miscible_with_water", "CAS_number",
  transpose = FALSE,
  kmeans = "elbow",
  na_replacement = "drop"
)
## Dropping any variables in your dataset that have NA as a value.
## Variables dropped:
## solubility_in_water vapor_pressure

solvents_pca_kmeans_elbow
## # A tibble: 31 x 2
##    cluster_number variance_within_cluster
##             <dbl>                   <dbl>
##  1              1                 142804.
##  2              2                  67355.
##  3              3                  49545.
## 4               4                   38964.
## 5               5                   30702.
## 6               6                   25212.
## 7               7                   20188.
## 8               8                   16508.
## 9               9                   14346.
## 10             10                   12265.
## # ... with 21 more rows
```

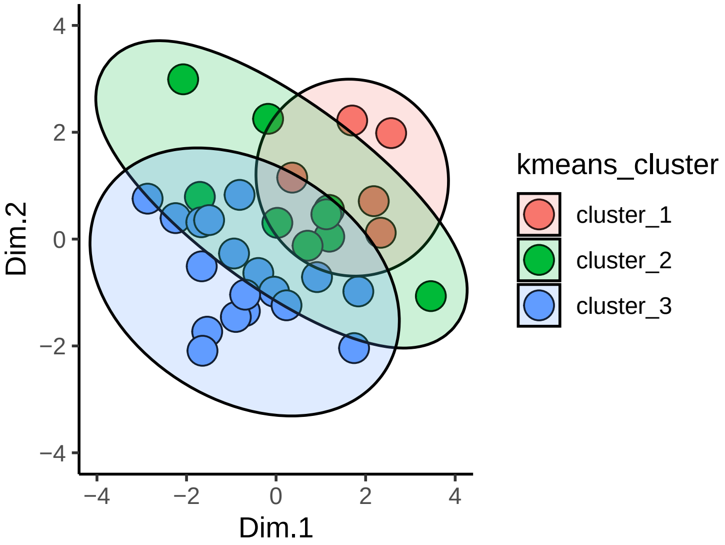
This gives us the maximum variance within a cluster for each number of clusters. Let's plot that:

```
ggplot(
  solvents_pca_kmeans_elbow,
  aes(x = cluster_number, y = variance_within_cluster)
) +
  geom_col() +
  geom_point() +
  geom_line()
```



Hmm, it looks like there aren't any strong elbows in this plot - probably the reason that the elbow method chooses such a high number of clusters. Suppose we want to manually set the number of clusters? We can set `kmeans = 3` if we want three clusters in the output. Below, let's do just that. Let's also plot the results and use `geom_mark_ellipse`.

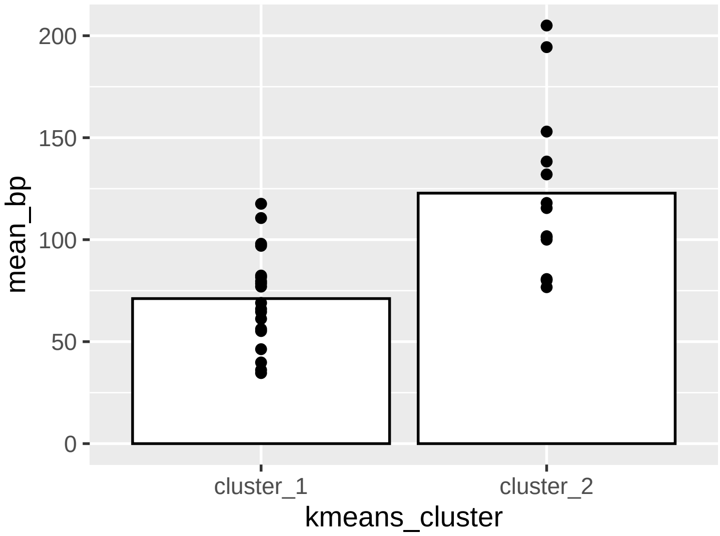
```
runMatrixAnalysis(
  data = solvents,
  analysis = c("pca"),
  column_w_names_of_multiple_analytes = NULL,
  column_w_values_for_multiple_analytes = NULL,
  columns_w_values_for_single_analyte = colnames(solvents)[c(3:5, 7:9, 11:12)],
  columns_w_additional_analyte_info = NULL,
  columns_w_sample_ID_info = c("solvent", "formula", "miscible_with_water", "CAS_number",
  transpose = FALSE,
  kmeans = 3,
  na_replacement = "drop"
) %>%

ggplot(aes(x = Dim.1, y = Dim.2, fill = kmeans_cluster)) +
  geom_point(shape = 21, size = 5) +
  geom_mark_ellipse(aes(label = kmeans_cluster), alpha = 0.2) +
  theme_classic() +
  coord_cartesian(xlim = c(-4,4), ylim = c(-4,4))
## Dropping any variables in your dataset that have NA as a value.
## Variables dropped:
## solubility_in_water vapor_pressure
```



Cool!

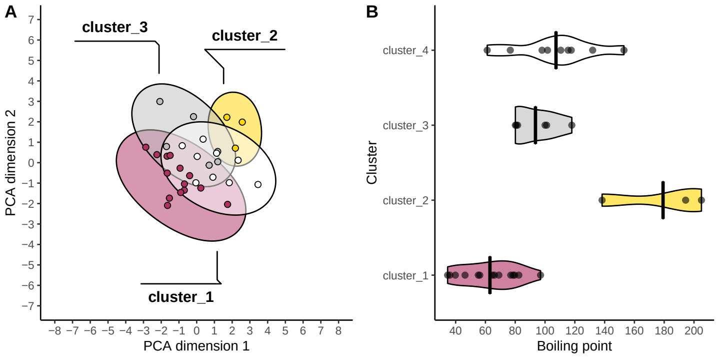
One more important point: when using kmeans, the output of runMatrixAnalysis (specifically the kmeans_cluster column) can be used to create groupings for summary statistics. For example, suppose we want two groups of solvents and we want to calculate the mean and standard deviation in boiling points for each of those groups:

```
solvents_clustered <- runMatrixAnalysis(
  data = solvents,
  analysis = c("pca"),
  column_w_names_of_multiple_analytes = NULL,
  column_w_values_for_multiple_analytes = NULL,
  columns_w_values_for_single_analyte = colnames(solvents)[c(3:5, 7:9, 11:12)],
  columns_w_additional_analyte_info = NULL,
  columns_w_sample_ID_info = c("solvent", "formula", "miscible_with_water", "CAS_number",
  transpose = FALSE,
  kmeans = 2,
  na_replacement = "drop"
)
## Dropping any variables in your dataset that have NA as a value.
## Variables dropped:
## solubility_in_water vapor_pressure

solvents_clustered_summary <- solvents_clustered %>%
  group_by(kmeans_cluster) %>%
  summarize(mean_bp = mean(boiling_point))

ggplot() +
  geom_col(
    data = solvents_clustered_summary,
    aes(x = kmeans_cluster, y = mean_bp),
    color = "black", fill = "white"
  ) +
  geom_point(
    data = solvents_clustered,
    aes(x = kmeans_cluster, y = boiling_point)
  )
```



Very good! Since we can use the outputs of our k-means analyses to run and visualize summary statistics, it's possible that we'll want to see the cluster plot (dendrogram or pca plot) alongside the summary stats plot. For this we can use the `plot_grid` function from the `cowplot` package. Let's check it out:

```
solvents_clustered <- runMatrixAnalysis(
  data = solvents,
  analysis = c("pca"),
  column_w_names_of_multiple_analytes = NULL,
  column_w_values_for_multiple_analytes = NULL,
  columns_w_values_for_single_analyte = colnames(solvents)[c(3:5, 7:9, 11:12)],
  columns_w_additional_analyte_info = NULL,
  columns_w_sample_ID_info = c("solvent", "formula", "miscible_with_water", "CAS_number",
  transpose = FALSE,
  kmeans = 4,
  na_replacement = "drop"
)
## Dropping any variables in your dataset that have NA as a value.
## Variables dropped:
## solubility_in_water vapor_pressure

colors <- c("maroon", "gold", "grey", "white")

pca_plot <- ggplot( data = solvents_clustered, aes(x = Dim.1, y = Dim.2, fill = kmeans_clu
  geom_mark_ellipse(
    aes(label = kmeans_cluster),
    alpha = 0.5, label.lineheight = 0.2, size = 0.5) +
  geom_point(shape = 21, size = 2) +
  theme_classic() +
  guides(fill = "none") +
  scale_x_continuous(name = "PCA dimension 1", breaks = seq(-8,8,1)) +
  scale_y_continuous(name = "PCA dimension 2", breaks = seq(-7,7,1)) +
  scale_fill_manual(values = colors) +
  coord_cartesian(xlim = c(-8,8), ylim = c(-7,7))

solvents_clustered_summary <- solvents_clustered %>%
  group_by(kmeans_cluster) %>%
  summarize(mean_bp = mean(boiling_point))

bar_plot <- ggplot() +
  geom_violin(
    data = solvents_clustered,
    aes(x = kmeans_cluster, y = boiling_point, fill = kmeans_cluster),
    size = 0.5, color = "black", alpha = 0.6, width = 0.5
  ) +
  geom_crossbar(
    data = solvents_clustered_summary,
    aes(x = kmeans_cluster, y = mean_bp, ymin = mean_bp, ymax = mean_bp),
    color = "black", width = 0.5
  ) +
  geom_point(
    data = solvents_clustered,
    aes(x = kmeans_cluster, y = boiling_point),
    size = 2, color = "black", alpha = 0.6
  ) +
  scale_y_continuous(name = "Boiling point", breaks = seq(0,250,20)) +
  scale_x_discrete(name = "Cluster") +
  scale_fill_manual(values = colors) +
  theme_classic() +
  coord_flip() +
  guides(fill = "none") +
  theme(legend.position = "bottom")

cowplot::plot_grid(pca_plot, bar_plot, align = "h", axis = "b", labels = "AUTO")
```



Now we are really rockin!!

0.8.2 exercises

Use the wine grapes dataset (it's stored as `wine_grape_data` after you run the `source(...)` command).

0.8.2.1 Question 1

Run a principal components analysis on the dataset. Use `na_replacement = "drop"` (so that variables with NA values are not included in the analysis) and generate clusters automatically using kmeans by setting `kmeans = "auto"`. Make scatter plot of the results. How many clusters does kmeans recommend?

0.8.2.2 Question 2

Modify your code from Question 1 so that only two clusters are generated. Plot the results. Use `geom_mark_ellipse` to highlight each cluster in your plot (note that the `fill` aesthetic is required to mark groups). Which varieties are put into each of the two clusters?

0.8.2.3 Question 3

Use an ordination plot to determine what chemicals makes Chardonnay so different from the other varieties. To what class of compounds do these chemical belong?

0.8.2.4 Question 4

Modify your code from Question 2 so that five clusters are generated. Plot the results. Use `geom_mark_ellipse` to highlight each cluster in your plot (note that the `fill` aesthetic is required to mark groups). Based on this plot, which grape variety undergoes the least amount of change, chemically speaking, between dry and well-watered conditions?

0.8.2.5 Question 5

Run a heirarchical clustering analysis on the wine grapes data set, using kmeans to create five groups, and also continue using `na_replacement = "drop"`. Plot the results. Which grape variety undergoes the most change in terms of its chemistry between well-watered and dry conditions? (hint: remember that the x-axis shows the distances between nodes and tips, the y-axis is meaningless). Compare the method you used to compare sample shifts in question 4 (i.e. pca+kmeans) versus the method you used in this question (i.e. hclust+kmeans). Which do you is better? Would this change depending on the circumstances?

0.8.2.6 Question 6

Google "Quercetin". What kind of compound is it? Use the clusters created by the heirarchical clustering analysis in question 5 as groups for which to calculate summary statistics. Calculate the mean and standard deviation of the concentration of Quercetin in each group. Plot the result using `geom_pointrange` and adjust axis font sizes so that they are in good proportion with the size of the plot. Also specify a theme (for example, `theme_classic()`).

Does one cluster have a large amount of variation in Quercetin abundance? Why do you think this might be?

0.8.2.7 Question 7

Use `cowplot::plot_grid` to display your plots from questions 4 and 5 next to each other.

0.8.2.8 Challenge (optional)

Use cowplot to display your plots from questions 4, 5, and 6 alongside each other. **Make your combined plot as attractive as possible!** Use each of the following:

`align = TRUE` inside `geom_tiplab()`

`nrow = 1` inside `plot_grid()`

`rel_widths = <your_choice>` inside `plot_grid()`

`name = <your_choice>` inside `scale_*_*`

`label = kmeans_cluster` inside `geom_mark_ellipse()`

`breaks = <your_choice>` inside `scale_x_continuous()` or `scale_y_continuous()` (as an example, `breaks = seq(0,10,1)`)

Also, consider using:

`guides(fill = "none", color = "none")`

Install the RColorBrewer package, and use one of its color schemes. As an example with the color scheme `Set1`:

`scale_fill_brewer(palette = "Set1", na.value = "grey")`

`scale_color_brewer(palette = "Set1", na.value = "grey")`

Save your plot as a png using `ggsave()`.

Maybe something like this:

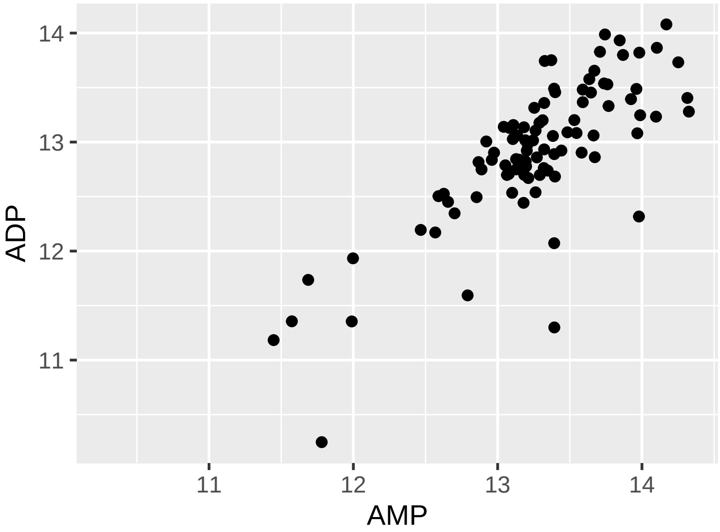
0.9 models

0.9.1 theory

Next on our quest to develop our abilities in analytical data exploration is modeling. We will start with some of the simplest models - linear models. There are a variety of ways to build linear models in R, but we will use a function called `buildLinearModel`. To use it, we simply give it our data, and tell it which to sets of values we want to compare. To tell it what we want to compare, we give it a formula in the form of Y = M x X + B, however, the B term and the M are implicit, so all we need to tell it is Y = X.

Let's look at an example. Suppose we want to know if the abundances of ADP and AMP are related in our metabolomics dataset:

```
ggplot(metabolomics_data) +
  geom_point(aes(x = AMP, y = ADP))
```



It looks like there might be a relationship! Let's build a linear model for that relationship:

```
model <- buildLinearModel(
  data = metabolomics_data,
  formula = "ADP = AMP"
)
str(model, strict.width = "cut")
## List of 2
##  $ metrics:'data.frame': 6 obs. of  4 variables:
##   ..$ variable: chr [1:6] "(Intercept)" "AMP" "median_res"..
##   ..$ value   : num [1:6] 0.7842 0.9142 0.0415 40.3224 15...
##   ..$ type    : chr [1:6] "coefficient" "coefficient" "st"..
##   ..$ p_value : chr [1:6] "0.4375" "0" NA NA ...
##  $ data   :'data.frame': 92 obs. of  7 variables:
##   ..$ input_x  : num [1:92] 13.2 13.5 14.3 13.3 12 ...
##   ..$ input_y  : num [1:92] 12.8 13.1 13.3 13.2 11.9 ...
##   ..$ ADP      : num [1:92] 12.8 13.1 13.3 13.2 11.9 ...
##   ..$ AMP      : num [1:92] 13.2 13.5 14.3 13.3 12 ...
##   ..$ residuals: num [1:92] 0.0312 -0.0217 -0.6014 0.2458 ..
##   ..$ model_y  : num [1:92] 12.8 13.1 13.9 13 11.8 ...
##   ..$ model_x  : num [1:92] 13.2 13.5 14.3 13.3 12 ...
```

The model consists of two thigs: metrics and data. Let's look at the metrics:

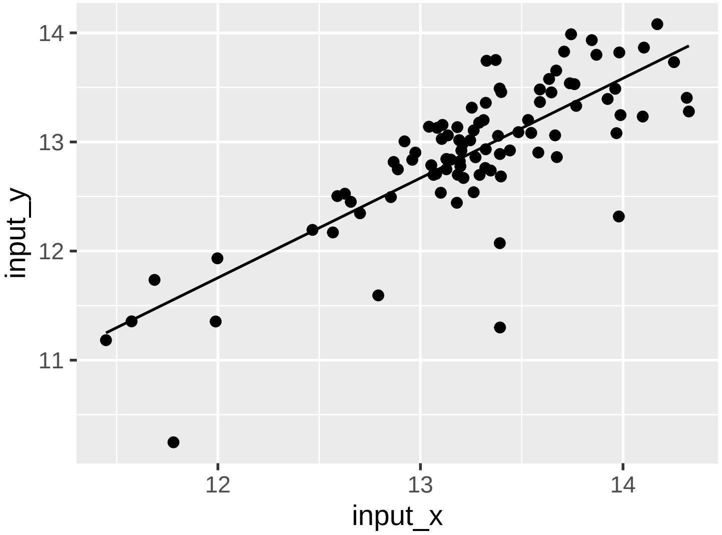
```
model$metrics
##                 variable   value        type p_value
## 1            (Intercept)  0.7842 coefficient  0.4375
## 2                    AMP  0.9142 coefficient       0
## 3        median_residual  0.0415   statistic    <NA>
## 4      total_sum_squares 40.3224   statistic    <NA>
## 5 residual_sum_squares 15.3901   statistic    <NA>
## 6              r_squared  0.6183   statistic    <NA>
```

It shows us the intercept (b), the variable for AMP (i.e. the slope, m), as well some other things (we will talk about them in a second). The other thing the model contains is the data (below). This includes the input_x and y values. The raw values for ADP and AMP, the residuals (see below for details), and the x and y values generated by the model.

```
head(model$data)
##    input_x  input_y      ADP      AMP   residuals  model_y
## 1 13.15029 12.83791 12.83791 13.15029  0.03119000 12.80672
## 2 13.48362 13.08980 13.08980 13.48362 -0.02165141 13.11146
## 3 14.32515 13.27943 13.27943 14.32515 -0.60138528 13.88082
## 4 13.31191 13.20029 13.20029 13.31191  0.24581244 12.95448
## 5 11.99764 11.93350 11.93350 11.99764  0.18057517 11.75293
## 6 12.95966 12.83649 12.83649 12.95966  0.20405638 12.63243
##    model_x
## 1 13.15029
## 2 13.48362
## 3 14.32515
## 4 13.31191
## 5 11.99764
## 6 12.95966
```

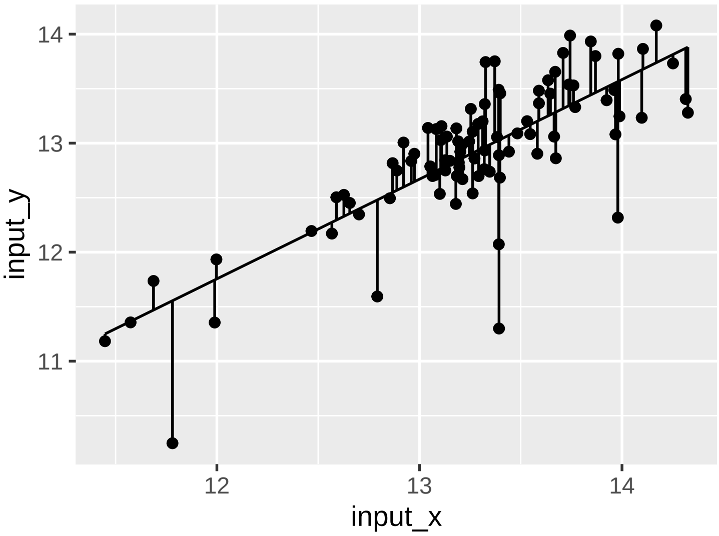
Let's plot the model!

```
ggplot(model$data) +
  geom_point(aes(x = input_x, y = input_y)) +
  geom_line(aes(x = model_x, y = model_y))
```



Very good. Now let's talk about evaluating the quality of our model. For this we need some means of assessing how well our line fits our data. We will use residuals - the distance between each of our points and our line.

```
ggplot(model$data) +
  geom_point(aes(x = input_x, y = input_y)) +
  geom_line(aes(x = model_x, y = model_y)) +
  geom_segment(aes(x = input_x, y = input_y, xend = input_x, yend = model_y))
```

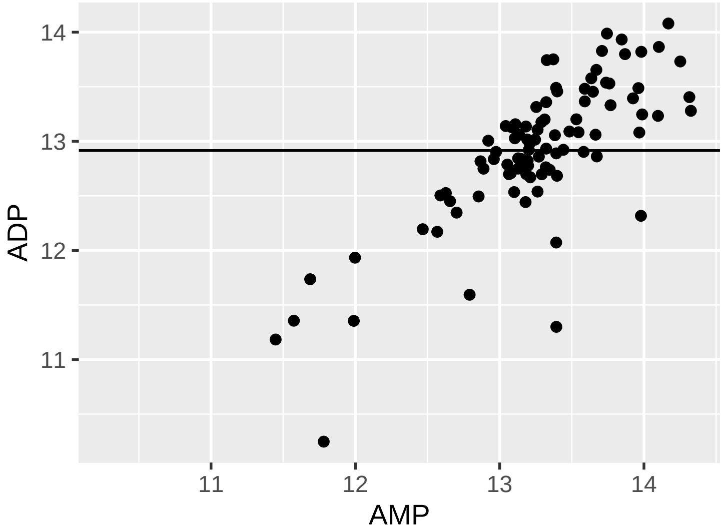


We can calculate the sum of the squared residuals:

```
sum(
  (model$data$input_y - model$data$model_y)^2
, na.rm = TRUE)
## [1] 15.39014
```

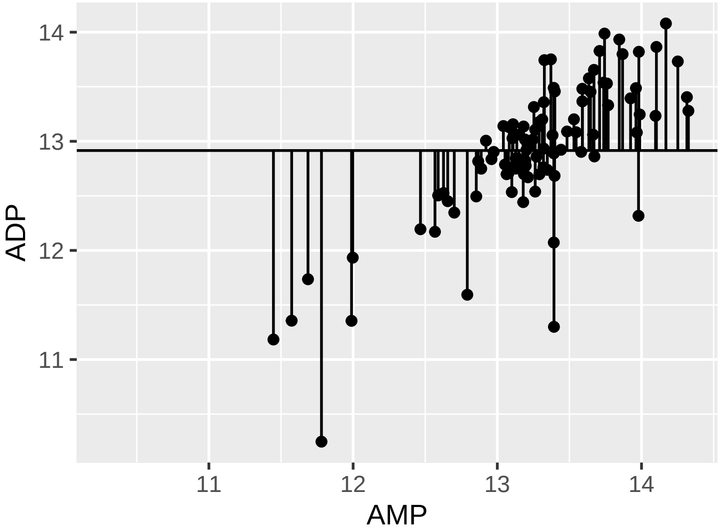
15.39! Let's call that the "residual sum of the squares". So. 15.39.. does that mean our model is good? I don't know. We have to compare that number to something. Let's compare it to a super simple model that is just defined by the mean y value of the input data.

```
ggplot(metabolomics_data) +
  geom_point(aes(x = AMP, y = ADP)) +
  geom_hline(aes(yintercept = mean(ADP, na.rm = TRUE)))
```



A pretty bad model, I agree. How much better is our linear model that the flat line model? Let's create a measure of the distance between each point and the point predicted for that same x value on the model:

```
sum(
  (metabolomics_data$ADP - mean(metabolomics_data$ADP, na.rm = TRUE))^2
, na.rm = TRUE)
## [1] 40.32239

ggplot(metabolomics_data) +
  geom_point(aes(x = AMP, y = ADP)) +
  geom_hline(aes(yintercept = mean(ADP, na.rm = TRUE))) +
  geom_segment(aes(x = AMP, y = ADP, xend = AMP, yend = mean(ADP, na.rm = TRUE)))
```

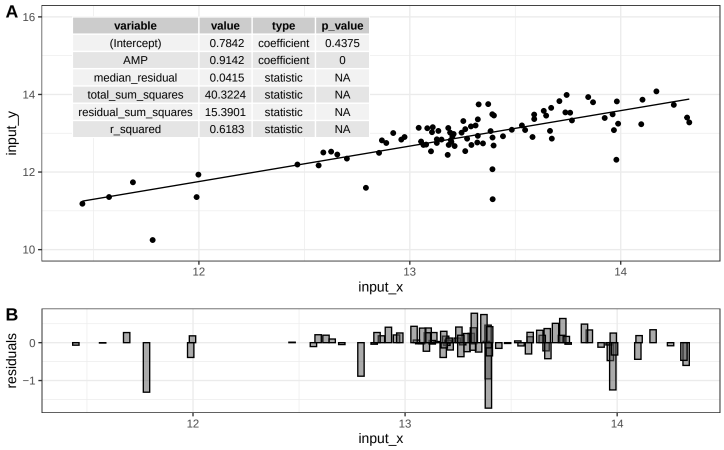


40.32! Wow. Let's call that the "total sum of the squares", and now we can compare that to our "residual sum of the squares":

```
1-(15.39/40.32)
## [1] 0.6183036
```

0.68! Alright. That is our R squared value. It is equal to 1 minus the ratio of the "residual sum of the squares" to the "total sum of the squares". Now, let's put it all together and make it pretty:

```
top <- ggplot(model$data) +
  geom_point(aes(x = input_x, y = input_y)) +
  geom_line(aes(x = model_x, y = model_y)) +
  annotate(geom = "table",
    x = 11.4,
    y = 16,
    label = list(model$metrics)
  ) +
  coord_cartesian(ylim = c(10,16)) +
  theme_bw()

bottom <- ggplot(model$data) +
  geom_col(
    aes(x = input_x, y = residuals),
    width = 0.03, color = "black", position = "dodge", alpha = 0.5
  ) +
  theme_bw()

cowplot::plot_grid(top, bottom, ncol = 1, labels = "AUTO", rel_heights = c(2,1))
```



variable	value	type	p_value
(Intercept)	0.7842	coefficient	0.4375
AMP	0.9142	coefficient	0
median_residual	0.0415	statistic	NA
total_sum_squares	40.3224	statistic	NA
residual_sum_squares	15.3901	statistic	NA
r_squared	0.6183	statistic	NA

0.9.2 exercises

To practice creating linear models, try the following:

1. Choose one of the datasets we have used so far, and run a principal components analysis on it. Note that the output of the analysis when you run "pca_ord" contains the Dimension 1 coordinate "Dim.1" for each sample, as well as the abundance of each analyte in that sample.
2. Using the information from the ordination plot, identify two analytes: one that has a variance that is strongly and positively correlated with the first principal component (i.e. dimension 1), and one that has a variance that is slightly less strongly, but still positively correlated with the first principal component. Using `buildLinearModel`, create and plot two linear models, one that regresses each of those analytes against dimension 1. Which has the greater r-squared value? Based on what you know about PCA, does that make sense?
3. Choose two analytes: one should be one of the analytes from question 2 above, the other should be an analyte that, according to your PCA ordination analysis, is negatively correlated with the first principal component. Using `buildLinearModel` create plots showing how those two analytes are correlated with dimension 1. One should be positively correlated, and the other negatively correlated. Enhance the plots by including in them a visual represetation of the residuals.

0.10 comparing means

shapiroTest leveneTest tTest wilcoxTest anovaTest tukeyTest kruskalTest dunnTest

“Are these two things the same?”

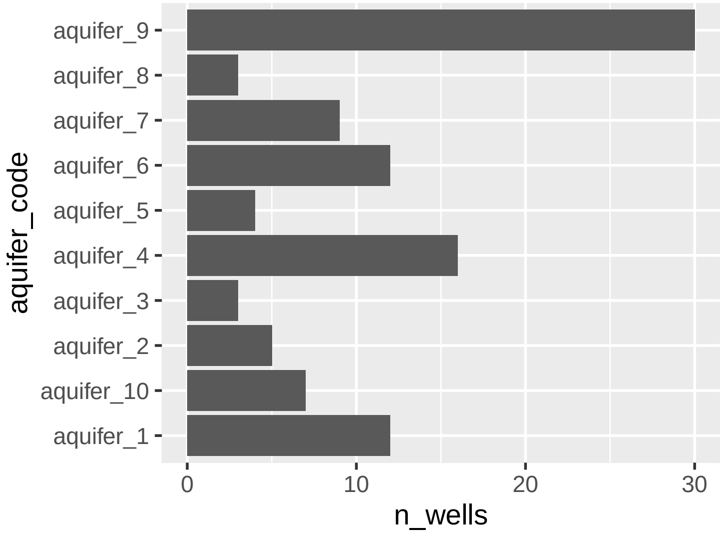
Often, we want to know if our study subjects contain different amounts of certain analytes. For example, “Does this lake over here contain more potassium than that lake over there?” For this, we need statistical tests. Here, we will have a look at comparing mean values for analyte abundance in situations with two samples and in situations with more than two samples.

I find many of the concepts discussed in this chapter easier to think about with an example in mind. For that, suppose that you are an analytical chemist on Hawaii that is studying the chemistry of the island’s aquifers. you have the data set `hawaii_aquifers`. You can see in the output below the structure of the data set - we have 990 measurements of a 9 different analytes in multiple wells that draw on a set of 10 aquifers.

```
hawaii_aquifers
## # A tibble: 990 x 6
##    aquifer_code well_name      longitude latitude analyte
##    <chr>        <chr>              <dbl>    <dbl> <chr>
##  1 aquifer_1    Alewa_Heights~        NA       NA SiO2
##  2 aquifer_1    Alewa_Heights~        NA       NA Cl
##  3 aquifer_1    Alewa_Heights~        NA       NA Mg
##  4 aquifer_1    Alewa_Heights~        NA       NA Na
##  5 aquifer_1    Alewa_Heights~        NA       NA K
##  6 aquifer_1    Alewa_Heights~        NA       NA SO4
##  7 aquifer_1    Alewa_Heights~        NA       NA HCO3
##  8 aquifer_1    Alewa_Heights~        NA       NA dissolved~
##  9 aquifer_1    Alewa_Heights~        NA       NA Ca
## 10 aquifer_1    Beretania_Hig~        NA       NA SiO2
## # ... with 980 more rows, and 1 more variable:
## #   abundance <dbl>
unique(hawaii_aquifers$aquifer_code)
##  [1] "aquifer_1"  "aquifer_2"  "aquifer_3"  "aquifer_4"
##  [5] "aquifer_5"  "aquifer_6"  "aquifer_7"  "aquifer_8"
##  [9] "aquifer_9"  "aquifer_10"
```

Importantly, there are many wells that draw on each aquifer, as shown in the graph below.

```
hawaii_aquifers %>%
  select(aquifer_code, well_name) %>%
  group_by(aquifer_code) %>%
  summarize(n_wells = length(unique(well_name))) -> aquifers_summarized

aquifers_summarized
## # A tibble: 10 x 2
##    aquifer_code n_wells
##    <chr>          <int>
##  1 aquifer_1         12
##  2 aquifer_10         7
##  3 aquifer_2          5
##  4 aquifer_3          3
##  5 aquifer_4         16
##  6 aquifer_5          4
##  7 aquifer_6         12
##  8 aquifer_7          9
##  9 aquifer_8          3
## 10 aquifer_9         30

ggplot(aquifers_summarized) + geom_col(aes(x = n_wells, y = aquifer_code))
```



0.10.1 definitions

1. **populations and independent measurements**: When we are comparing means, we are comparing two *sets* of values. It is important to consider where these values came from in the first place. In particular, it is usually useful to think of these values as representatives of larger populations. In the example of our aquifer data set,

we can think of the measurements from different wells drawing on the same aquifer as independent measurements of the "population" (i.e. the aquifer).

2. **the null hypothesis**: When we conduct a statistical test, we are testing the null hypothesis. The null (think "default") hypothesis is that there is no difference bewteen the means (hence the name "null"). In the example of our aquifers, let's say that we're interested in whether two aquifers have different abundances of potassium - in this case the null hypothesis is that they do not differ, in other words, that they have the same amount of potassium.
3. **the *p* value**: The *p* value represents the probability of getting data as extreme as our results if the null hypothesis is true. In other words - the *p* value is the probability that we would observe the differences we did, if in fact there were no differences in the means at all. To continue with our example: suppose we measure potassium levels in 10% of the wells that access each aquifer and find that aquifer_1 has potassium levels of 14 +/- 2 and aquifer_2 has potassium levels of 12 +/- 1. Suppose that we then conduct a statistical test and get a p value of 0.04. This means that, assuming the aquifers have the same magneisum levels (i.e. assuming the null hypothesis is true), there is a 4% chance that we would get the measured values that we did. In other words, IF the aquifers have the same potassium abundance, it is pretty unlikely that we would have obtained the measurements that we did.

Please note that the the *p* value is not the probability of a detected difference being a false positive. The probability of a false positive requires additional information in order to be calculated. For further discussion please see the end of this chapter.

0.10.2 test selection

There are many different types of statistical tests. Below is a flow chart illustrating how it is recommended that statistical tests be used in this course. You can see that there are three regimes of tests: variance and normality tests (blue), parametric tests (green), and non-parametric tests (orange):

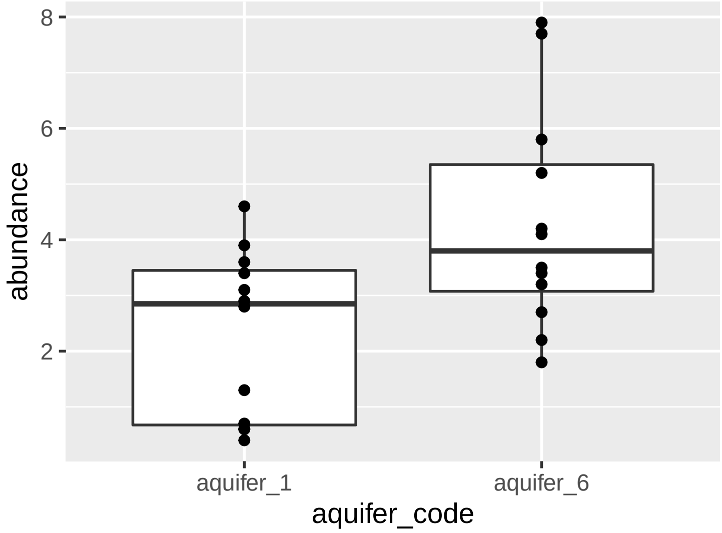
When we are comparing means, we need to first determine what kind of statistical tests we can use with our data. If (i) our data can be reasonably modelled by a normal distribution and (ii) the variances about the two means are similar, then we can use the more powerful "parametric" tests (i.e. tests that will be more likely to detect a difference in means, assuming one exists). If one of these criteria are not met, then we need to use less powerful "non-parametric" tests.

We can check our data for normality and similar variances using the Shapiro test and the Levene test. Let’s use the `hawaii_aquifers` data as an example, and let’s consider only the element potassium:

```
K_data <- hawaii_aquifers %>%
  filter(analyte == "K")
  K_data
## # A tibble: 110 x 6
##    aquifer_code well_name         longitude latitude analyte
##    <chr>        <chr>                 <dbl>    <dbl> <chr>
##  1 aquifer_1    Alewa_Heights_Sp~       NA       NA   K
##  2 aquifer_1    Beretania_High_S~       NA       NA   K
##  3 aquifer_1    Beretania_Low_Se~       NA       NA   K
##  4 aquifer_1    Kuliouou_Well         -158.     21.3 K
##  5 aquifer_1    Manoa_Well_II         -158.     21.3 K
##  6 aquifer_1    Moanalua_Wells_P~     -158.     21.4 K
##  7 aquifer_1    Moanalua_Wells_P~     -158.     21.4 K
##  8 aquifer_1    Moanalua_Wells_P~     -158.     21.4 K
##  9 aquifer_1    Nuuanu_Aerator_W~     -158.     21.4 K
## 10 aquifer_1    Palolo_Tunnel         -158.     21.3 K
## # ... with 100 more rows, and 1 more variable:
## #   abundance <dbl>
```

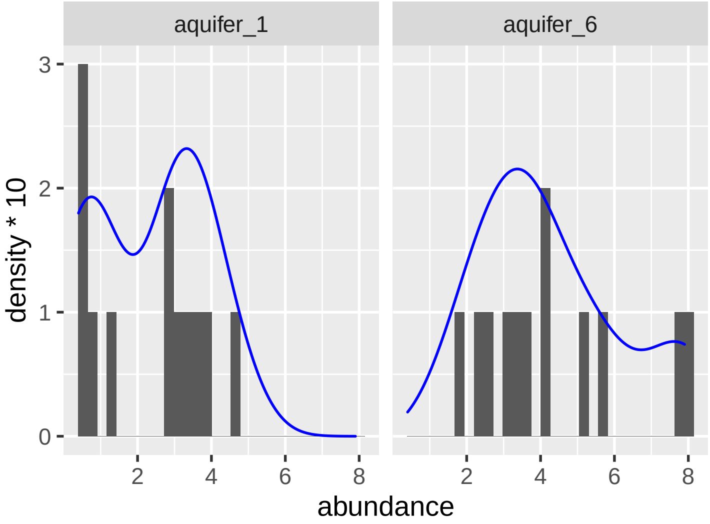
To work with two means, let’s just look at aquifers 1 and 6:

```
K_data_1_2 <- K_data %>%
    filter(aquifer_code %in% c("aquifer_1", "aquifer_6"))

K_data_1_2
## # A tibble: 24 x 6
##    aquifer_code well_name         longitude latitude analyte
##    <chr>        <chr>                 <dbl>    <dbl> <chr>
##  1 aquifer_1    Alewa_Heights_Sp~       NA       NA   K
##  2 aquifer_1    Beretania_High_S~       NA       NA   K
##  3 aquifer_1    Beretania_Low_Se~       NA       NA   K
##  4 aquifer_1    Kuliouou_Well         -158.     21.3 K
##  5 aquifer_1    Manoa_Well_II         -158.     21.3 K
##  6 aquifer_1    Moanalua_Wells_P~     -158.     21.4 K
##  7 aquifer_1    Moanalua_Wells_P~     -158.     21.4 K
##  8 aquifer_1    Moanalua_Wells_P~     -158.     21.4 K
##  9 aquifer_1    Nuuanu_Aerator_W~     -158.     21.4 K
## 10 aquifer_1    Palolo_Tunnel         -158.     21.3 K
## # ... with 14 more rows, and 1 more variable:
## #   abundance <dbl>

ggplot(K_data_1_2, aes(x = aquifer_code, y = abundance)) +
    geom_boxplot() +
    geom_point()
```



Are these data normally distributed? Do they have similar variance? Let's get a first approximation by looking at a plot:

```
K_data_1_2 %>%
  ggplot(aes(x = abundance)) +
    geom_histogram(bins = 30) +
    facet_wrap(~aquifer_code) +
    geom_density(aes(y = ..density..*10), color = "blue")
```



Based on this graphic, it's hard to say! Let's use a statistical test to help. When we want to run the Shaprio test, we are looking to see if each group has normally distributed here (here group is "aquifer_code", i.e. aquifer_1 and aquifer_6). This means we need to `group_by(aquifer_code)` before we run the test:

```
K_data_1_2 %>%
  group_by(aquifer_code) %>%
  shapiro_test(abundance)
## # A tibble: 2 x 4
##   aquifer_code variable  statistic     p
##   <chr>        <chr>         <dbl> <dbl>
## 1 aquifer_1    abundance     0.885 0.102
## 2 aquifer_6    abundance     0.914 0.239
```

Both p-values are above 0.05! This means that the distributions are not significantly different from a normal distribution. What about the variances about the two means? Are they similar? For this we need a Levene test. With that test, we are not looking within each group, but rather across groups - this means we do NOT need to `group_by(aquifer_code)` and should specify a `y ~ x` formula instead:

```
K_data_1_2 %>%
  levene_test(abundance ~ aquifer_code)
## # A tibble: 1 x 4
##     df1   df2 statistic     p
##   <int> <int>     <dbl> <dbl>
## 1     1    22     0.289 0.596
```

The p-value from this test is 0.596! This means that their variances are not significantly different.

0.10.3 two means

Now, since our data passed both test, this means we can use a normal t-test. A t-test is a parametric test. This means that it relies on modelling the data using a normal distribution in order to make comparisons. It is also a powerful test. This means that it is likely to detect a difference in means, assuming one is present. Let's try it out:

```
K_data_1_2 %>%
  t_test(abundance ~ aquifer_code)
## # A tibble: 1 x 8
##   .y.       group1 group2    n1    n2 statistic    df      p
## * <chr>     <chr>  <chr>  <int> <int>     <dbl> <dbl>  <dbl>
## 1 abundance aquif~ aquif~    12    12     -2.75  20.5 0.0121
```

A p-value of 0.012! This is below 0.05, meaning that there is a 95% chance that the two means are different. Suppose that our data had not passed the Shapiro and/or Levene tests. We would then need to use a Wilcox test. The Wilcox test is a non-parametric test, which means that it does not use a normal distribution to model the data in order to make comparisons. This means that is a less powerful test than the t-test, which means that it is less likely to detect a difference in the means, assuming there is one. For fun, let's try that one out and compare the p-values from the two methods:

```
K_data_1_2 %>%
  wilcox_test(abundance ~ aquifer_code)
## # A tibble: 1 x 7
##   .y.       group1    group2       n1    n2 statistic      p
## * <chr>     <chr>     <chr>     <int> <int>     <dbl>  <dbl>
## 1 abundance aquifer_1 aquifer_6    12    12      33.5 0.0282
```

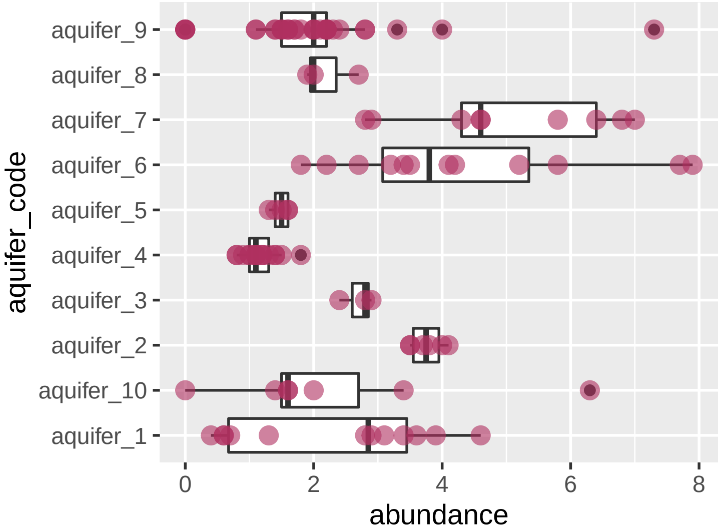
A p-value of 0.028! This is higher than the value given by the t-test (0.012). That is because the Wilcox test is a less powerful test: it is less likely to detect differences in means, assuming they exist.

0.10.4 more than two means

In the previous section we compared two means. What if we want to compare means from more than two study subjects? The first step is again to determine which tests to use. Let's consider our hawaii aquifer data again, though this time let's use all the aquifers, not just two:

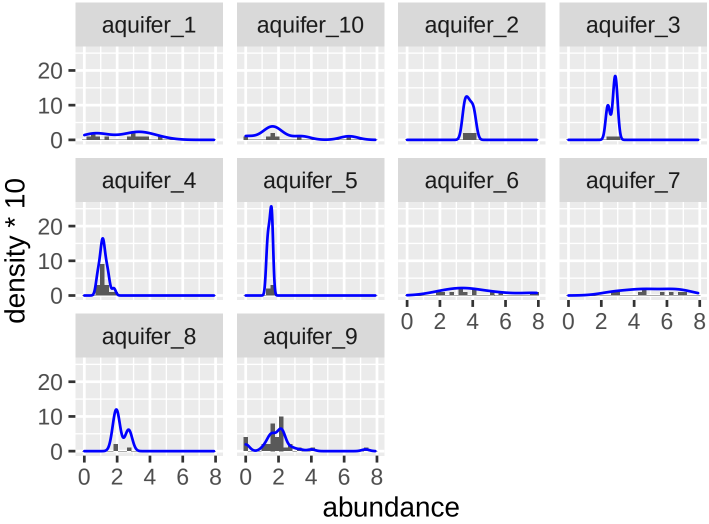
```
K_data <- hawaii_aquifers %>%
  filter(analyte == "K")

K_data
## # A tibble: 110 x 6
##    aquifer_code well_name          longitude latitude analyte
##    <chr>        <chr>                  <dbl>    <dbl> <chr>
##  1 aquifer_1    Alewa_Heights_Sp~       NA       NA   K
##  2 aquifer_1    Beretania_High_S~       NA       NA   K
##  3 aquifer_1    Beretania_Low_Se~       NA       NA   K
##  4 aquifer_1    Kuliouou_Well         -158.     21.3 K
##  5 aquifer_1    Manoa_Well_II         -158.     21.3 K
##  6 aquifer_1    Moanalua_Wells_P~     -158.     21.4 K
##  7 aquifer_1    Moanalua_Wells_P~     -158.     21.4 K
##  8 aquifer_1    Moanalua_Wells_P~     -158.     21.4 K
##  9 aquifer_1    Nuuanu_Aerator_W~     -158.     21.4 K
## 10 aquifer_1    Palolo_Tunnel         -158.     21.3 K
## # ... with 100 more rows, and 1 more variable:
## #   abundance <dbl>
```

```
ggplot(data = K_data, aes(y = aquifer_code, x = abundance)) +
  geom_boxplot() +
  geom_point(color = "maroon", alpha = 0.6, size = 3)
```



Let's check visually to see if each group is normally distributed and to see if they have roughly equal variance:

```
K_data %>%
  group_by(aquifer_code) %>%
  ggplot(aes(x = abundance)) +
    geom_histogram(bins = 30) +
    facet_wrap(~aquifer_code) +
    geom_density(aes(y = ..density..*10), colour = "blue")
```



Again, it is somewhat hard to tell visually if these data are normally distributed. It seems pretty likely that they have different variances about the means, but let's check using the Shapiro and Levene tests. Don't forget: with the Shaprio test, we are looking within each group and so need to `group_by()`, with the Levene test, we are looking across groups, and so need to provide a `y~x` formula:

```
K_data %>%
  group_by(aquifer_code) %>%
  shapiro_test(abundance)
## # A tibble: 10 x 4
##    aquifer_code variable  statistic          p
##    <chr>        <chr>         <dbl>      <dbl>
##  1 aquifer_1    abundance     0.885 0.102
##  2 aquifer_10   abundance     0.864 0.163
##  3 aquifer_2    abundance     0.913 0.459
##  4 aquifer_3    abundance     0.893 0.363
##  5 aquifer_4    abundance     0.948 0.421
##  6 aquifer_5    abundance     0.902 0.421
##  7 aquifer_6    abundance     0.914 0.239
##  8 aquifer_7    abundance     0.915 0.355
##  9 aquifer_8    abundance     0.842 0.220
## 10 aquifer_9    abundance     0.786 0.00000866
```

```
K_data %>%
  levene_test(abundance ~ aquifer_code)
## # A tibble: 1 x 4
##     df1   df2 statistic       p
##   <int> <int>     <dbl>   <dbl>
## 1     9   100      3.12 0.00239
```

Based on these tests, it looks like the data for aquifer 9 is significantly different from a normal distribution (Shaprio test p = 0.000008), and the variances are certainly different from one another (Levene test p = 0.002).

Let's assume for a second that our data passed these tests. This means that we could reasonably model our data with normal distributions and use a parametric test to compare means. This means that we can use an ANOVA to test for differences in means.

0.10.4.1 ANOVA, Tukey tests

We will use the `anova_test` function from the package `rstatix`. It will tell us if any of the means in the data are statistically different from one another. However, if there are differences between the means, it will not tell us which of them are different.

```
K_data %>%
  anova_test(abundance ~ aquifer_code)
## Coefficient covariances computed by hccm()
## ANOVA Table (type II tests)
##
##         Effect DFn DFd      F        p p<.05   ges
## 1 aquifer_code   9 100 10.021 7.72e-11     * 0.474
```

A p-value of 7.7e-11! There are definitely some significant differences among this group. But, WHICH are different from one another though? For this, we need to run Tukey's Honest Significant Difference test (implemented using `tukey_hsd`). This will essentially run t-test on all the pairs of study subjects that we can derive from our data set (in this example, aquifer_1 vs. aquifer_2, aquifer_1 vs. aquifer_3, etc.). After that, it will correct the p-values according to the number of comparisons that it performed. This controls the rate of type I error that we can expect from the test. These corrected values are provided to us in the `p.adj` column.

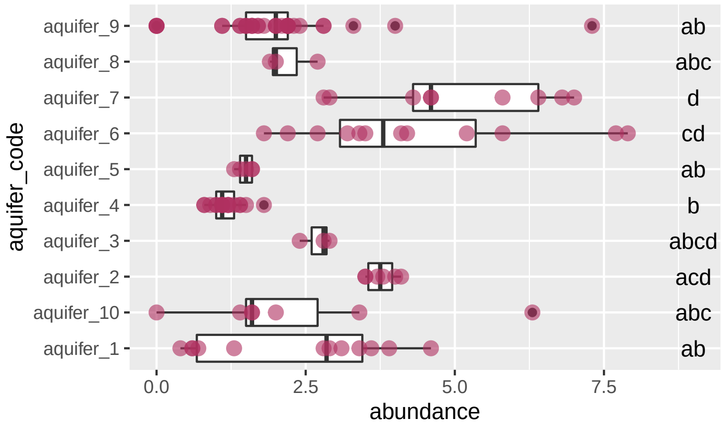
```
K_data %>%
  tukey_hsd(abundance ~ aquifer_code)
## # A tibble: 45 x 9
##    term         group1     group2 null.value estimate conf.low
##  * <chr>        <chr>      <chr>       <dbl>    <dbl>    <dbl>
##  1 aquifer_code aquifer_1  aquif~          0  0.00357   -2.00
##  2 aquifer_code aquifer_1  aquif~          0  1.44      -0.668
##  3 aquifer_code aquifer_1  aquif~          0  0.375     -2.35
##  4 aquifer_code aquifer_1  aquif~          0 -1.15      -2.75
##  5 aquifer_code aquifer_1  aquif~          0 -0.845     -3.09
##  6 aquifer_code aquifer_1  aquif~          0  1.98       0.261
##  7 aquifer_code aquifer_1  aquif~          0  2.70       0.837
##  8 aquifer_code aquifer_1  aquif~          0 -0.125     -2.85
##  9 aquifer_code aquifer_1  aquif~          0 -0.378     -1.78
## 10 aquifer_code aquifer_10 aquif~          0  1.44      -0.910
## # ... with 35 more rows, and 3 more variables:
## #     conf.high <dbl>, p.adj <dbl>, p.adj.signif <chr>
```

Using the output from our tukey test, we can determine which means are similar. We can do this using the p_groups function:

```
groups_based_on_tukey <- K_data %>%
  tukey_hsd(abundance ~ aquifer_code) %>%
  p_groups()
groups_based_on_tukey
##              treatment group spaced_group
## aquifer_1    aquifer_1    ab          ab
## aquifer_10 aquifer_10   abc          abc
## aquifer_2    aquifer_2   acd          a cd
## aquifer_3    aquifer_3  abcd          abcd
## aquifer_4    aquifer_4     b           b
## aquifer_5    aquifer_5    ab          ab
## aquifer_6    aquifer_6    cd            cd
## aquifer_7    aquifer_7     d             d
## aquifer_8    aquifer_8   abc          abc
## aquifer_9    aquifer_9    ab          ab
```

We can use the output from `p_groups` to annotate our plot:

```
ggplot(data = K_data, aes(y = aquifer_code, x = abundance)) +
  geom_boxplot() +
  geom_point(color = "maroon", alpha = 0.6, size = 3) +
  geom_text(data = groups_based_on_tukey, aes(y = treatment, x = 9, label = group))
```



Excellent! This plot shows us, using the letters on the same line with each aquifer, which means are the same and which are different. If a letter is shared among the labels in line with two aquifers, it means that their means do not differ significantly. For example, aquifer 2 and aquifer 6 both have "b" in their labels, so their means are not different - and are the same as those of aquifers 3 and 10.

0.10.4.2 Kruskal, Dunn tests

The above ANOVA example is great, but remember - our data did not pass the Shapiro or Levene tests. This means not all our data can be modelled by a normal distribution and taht we need to use a non-parametric test. The non-parametric alternative to the ANOVA is called the Kruskal test. Like the Wilcox test, it is less powerful that its parametric relative, meaning that it is less likely to detected differences, should they exist. However, since our data do not pass the Shapiro/Levene tests, we have to resort to the Kruskal test. Let's try it out:

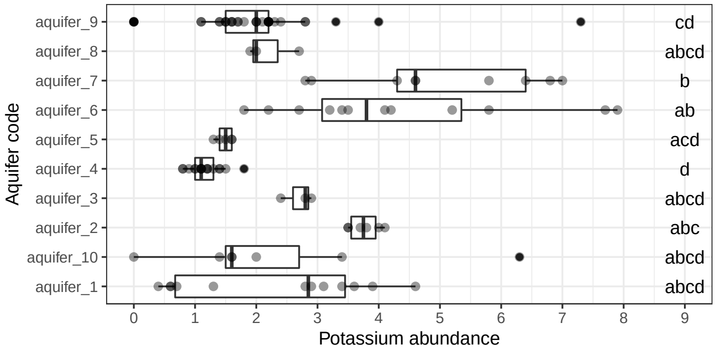
```
K_data %>%
  kruskal_test(abundance ~ aquifer_code)
## # A tibble: 1 x 6
##   .y.           n statistic    df            p method
## * <chr>     <int>     <dbl> <int>        <dbl> <chr>
## 1 abundance   110      57.7     9 0.0000000037 Kruskal-Wall~
```

A p-value of 3.9e-9! This is higher than the p-value from running ANOVA on the same data (remember, the Kruskal test is less powerful). Never the less, the value is still well below 0.05, meaning that some of the means are different. So, how do we determine WHICH are different from one another? When we ran ANOVA the follow-up test (the post hoc test) was Tukey's HSD. After the Kruskal test, the post hoc test we use is the Dunn test. Let's try:

```
K_data %>%
  dunn_test(abundance ~ aquifer_code)
## # A tibble: 45 x 9
##    .y.    group1 group2    n1    n2 statistic       p  p.adj
##  * <chr>  <chr>  <chr>  <int> <int>     <dbl>   <dbl>  <dbl>
##  1 abund~ aquif~ aquif~    12     7    -0.205 0.838   1
##  2 abund~ aquif~ aquif~    12     6     2.25  0.0242  0.702
##  3 abund~ aquif~ aquif~    12     3     0.911 0.362   1
##  4 abund~ aquif~ aquif~    12    17    -2.70  0.00702 0.232
##  5 abund~ aquif~ aquif~    12     5    -1.15  0.252   1
##  6 abund~ aquif~ aquif~    12    12     2.53  0.0113  0.351
##  7 abund~ aquif~ aquif~    12     9     3.02  0.00254 0.0967
##  8 abund~ aquif~ aquif~    12     3     0.182 0.855   1
##  9 abund~ aquif~ aquif~    12    36    -0.518 0.605   1
## 10 abund~ aquif~ aquif~     7     6     2.20  0.0278  0.777
## # ... with 35 more rows, and 1 more variable:
## #   p.adj.signif <chr>
```

This gives us adjusted p-values for all pairwise comparisons. Once again, we can use `p_groups()` to give us a compact letter display for each group, which can then be used to annotate the plot:

```
groups_based_on_dunn <- K_data %>%
  dunn_test(abundance ~ aquifer_code) %>%
  p_groups()
groups_based_on_dunn
##             treatment group spaced_group
## aquifer_1   aquifer_1  abcd         abcd
## aquifer_10 aquifer_10  abcd         abcd
## aquifer_2   aquifer_2   abc         abc
## aquifer_3   aquifer_3  abcd         abcd
## aquifer_4   aquifer_4     d            d
## aquifer_5   aquifer_5   acd         a cd
## aquifer_6   aquifer_6    ab         ab
## aquifer_7   aquifer_7     b          b
## aquifer_8   aquifer_8  abcd         abcd
## aquifer_9   aquifer_9    cd           cd

ggplot(data = K_data, aes(y = aquifer_code, x = abundance)) +
  geom_boxplot() +
  geom_point(color = "black", alpha = 0.4, size = 2) +
  scale_x_continuous(name = "Potassium abundance", breaks = seq(0,10,1)) +
  scale_y_discrete(name = "Aquifer code") +
  geom_text(data = groups_based_on_dunn, aes(y = treatment, x = 9, label = group)) +
  theme_bw()
```

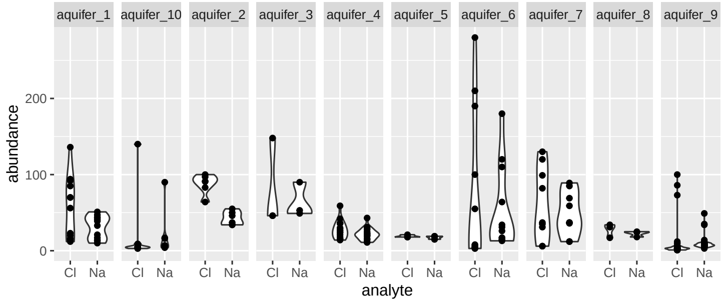


Note that these groupings are different from those generated by ANOVA/Tukey.

0.10.5 pairs of means

Oftentimes we have more than two means to compare, but rather than wanting to compare all means at once, we want to compare them in a pairwise fashion. For example, suppose we want to know if any of the aquifers contain different amounts of Na and Cl. We are not interested in testing for differences among *all* values of Na and Cl, rather, we want to test all *pairs* of Na and Cl values arising from each aquifer. That is to say, we want to compare the means in each facet of the plot below:

```
hawaii_aquifers %>%
  filter(analyte %in% c("Na", "Cl")) %>%
  ggplot(aes(x = analyte, y = abundance)) + geom_violin() + geom_point() + facet_grid(.~aq
```



Fortunately, we can use an approach that is very similar to the what we've learned in the earlier portions of this chapter, just with minor modifications. Let's have a look! We start with the Shapiro and Levene tests, as usual (note that we group using two variables when using the Shapiro test so that each analyte within each aquifer is considered as an individual distribution):

```
hawaii_aquifers %>%
  filter(analyte %in% c("Na", "Cl")) %>%
  group_by(analyte, aquifer_code) %>%
  shapiro_test(abundance)
## # A tibble: 20 x 5
##    aquifer_code analyte variable  statistic        p
##    <chr>        <chr>   <chr>         <dbl>    <dbl>
##  1 aquifer_1    Cl      abundance     0.900 1.59e- 1
##  2 aquifer_10   Cl      abundance     0.486 1.09e- 5
##  3 aquifer_2    Cl      abundance     0.869 2.24e- 1
##  4 aquifer_3    Cl      abundance     0.75  0
##  5 aquifer_4    Cl      abundance     0.903 7.49e- 2
##  6 aquifer_5    Cl      abundance     0.767 4.22e- 2
##  7 aquifer_6    Cl      abundance     0.741 2.15e- 3
##  8 aquifer_7    Cl      abundance     0.893 2.12e- 1
##  9 aquifer_8    Cl      abundance     0.878 3.17e- 1
## 10 aquifer_9    Cl      abundance     0.414 7.58e-11
## 11 aquifer_1    Na      abundance     0.886 1.06e- 1
## 12 aquifer_10   Na      abundance     0.593 2.26e- 4
## 13 aquifer_2    Na      abundance     0.884 2.88e- 1
## 14 aquifer_3    Na      abundance     0.822 1.69e- 1
## 15 aquifer_4    Na      abundance     0.933 2.41e- 1
## 16 aquifer_5    Na      abundance     0.782 5.71e- 2
## 17 aquifer_6    Na      abundance     0.764 3.80e- 3
## 18 aquifer_7    Na      abundance     0.915 3.51e- 1
## 19 aquifer_8    Na      abundance     0.855 2.53e- 1
## 20 aquifer_9    Na      abundance     0.544 2.09e- 9
```

Looks like some of those distributions are significantly different from normal! Let's run the levene test anyway. Note that for this particular case of the Levene test, we are interested in testing whether each pair of distributions has similar variances. For that we need to feed the Levene test data that is grouped by aquifer_code (so that it tests each pair as a group), then we need to specify the y ~ x formula (which in this case is abundance ~ analyte):

```
hawaii_aquifers %>%
  filter(analyte %in% c("Na", "Cl")) %>%
  group_by(aquifer_code) %>%
  levene_test(abundance ~ analyte)
## # A tibble: 10 x 5
##    aquifer_code   df1   df2 statistic       p
##    <chr>        <int> <int>     <dbl>   <dbl>
##  1 aquifer_1        1    22   10.5    0.00375
##  2 aquifer_10       1    12    0.0535 0.821
##  3 aquifer_2        1    10    0.0243 0.879
##  4 aquifer_3        1     4    0.320  0.602
##  5 aquifer_4        1    32    1.57   0.219
##  6 aquifer_5        1     8    0.474  0.511
##  7 aquifer_6        1    22    1.03   0.322
##  8 aquifer_7        1    16    1.54   0.232
##  9 aquifer_8        1     4    0.515  0.512
## 10 aquifer_9        1    70    1.07   0.304
```

It looks like the variances of the pair in aquifer 1 have significantly different variances. So - we for sure need to be using non-parametric testing. If this were a simple case of two means we would use the `wilcox_test`, but we have may pairs, so we will use `pairwise_wilcox_test` (note that with this test there are options for various styles of controlling for multiple comparisons, see: `?pairwise_wilcox_test`):

```
hawaii_aquifers %>%
  filter(analyte %in% c("Na", "Cl")) %>%
  group_by(aquifer_code) %>%
  pairwise_wilcox_test(abundance~analyte)
## # A tibble: 10 x 10
##    aquifer_code .y.       group1 group2    n1    n2 statistic
##  * <chr>        <chr>     <chr>  <chr>  <int> <int>     <dbl>
##  1 aquifer_1    abundance Cl     Na        12    12      99.5
##  2 aquifer_10   abundance Cl     Na         7     7      14
##  3 aquifer_2    abundance Cl     Na         6     6      36
##  4 aquifer_3    abundance Cl     Na         3     3       3
##  5 aquifer_4    abundance Cl     Na        17    17     189
##  6 aquifer_5    abundance Cl     Na         5     5      17.5
##  7 aquifer_6    abundance Cl     Na        12    12      53
##  8 aquifer_7    abundance Cl     Na         9     9      42
##  9 aquifer_8    abundance Cl     Na         3     3       6
## 10 aquifer_9    abundance Cl     Na        36    36     248.
## # ... with 3 more variables: p <dbl>, p.adj <dbl>,
## #   p.adj.signif <chr>
```

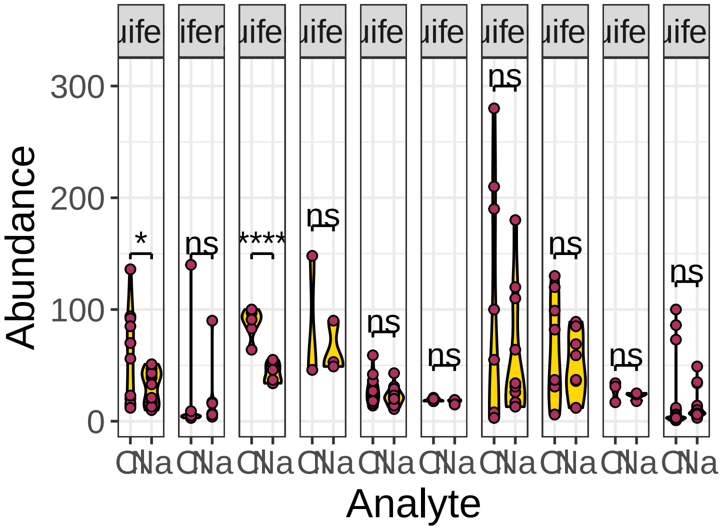
Excellent! It looks like there is a statistically significant difference between the means of the abundances of Cl and Na in aquifer_2 and (surprisingly?) in aquifer_9 (perhaps due to the large number of observations?).

What would we have done if our Shaprio and Levene tests had revealed no significant differences? Well, a `pairwise_t_test` of course!

```
hawaii_aquifers %>%
  filter(analyte %in% c("Na", "Cl")) %>%
  group_by(aquifer_code) %>%
  pairwise_t_test(abundance~analyte) -> test_output
  test_output
## # A tibble: 10 x 10
##    aquifer_code .y.       group1 group2    n1    n2         p
##  * <chr>        <chr>     <chr>  <chr>  <int> <int>     <dbl>
##  1 aquifer_1    abundance Cl     Na        12    12  4.69e-2
##  2 aquifer_10   abundance Cl     Na         7     7  8.82e-1
##  3 aquifer_2    abundance Cl     Na         6     6  3.75e-5
##  4 aquifer_3    abundance Cl     Na         3     3  6.83e-1
##  5 aquifer_4    abundance Cl     Na        17    17  1.03e-1
##  6 aquifer_5    abundance Cl     Na         5     5  1.45e-1
##  7 aquifer_6    abundance Cl     Na        12    12  5.66e-1
##  8 aquifer_7    abundance Cl     Na         9     9  5.21e-1
##  9 aquifer_8    abundance Cl     Na         3     3  4.28e-1
## 10 aquifer_9    abundance Cl     Na        36    36  9.48e-1
## # ... with 3 more variables: p.signif <chr>, p.adj <dbl>,
## #   p.adj.signif <chr>
```

Excellent, now we see how to run parametric and non-parametric pairwise comparisons. How do we annotate plots with the output of these tests? Here is an example:

```
anno <- data.frame(
  xmin = test_output$group1,
  xmax = test_output$group2,
  y_position = c(150, 150, 150, 175, 80, 50, 300, 150, 50, 125),
  text = test_output$p.signif,
  text_size = 10,
  text_vert_offset = 10,
  text_horiz_offset = 1.5,
  tip_length_xmin = 5,
  tip_length_xmax = 5,
  aquifer_code = test_output$aquifer_code
)

hawaii_aquifers %>%
  filter(analyte %in% c("Na", "Cl")) %>%
  ggplot(aes(x = analyte, y = abundance)) +
  geom_violin(fill = "gold", color = "black") +
  geom_point(shape = 21, fill = "maroon", color = "black") +
  facet_grid(.~aquifer_code) +
  geomSignif(data = anno) +
  scale_x_discrete(name = "Analyte") +
  scale_y_continuous(name = "Abundance") +
  theme_bw() +
  theme(
    text = element_text(size = 16)
    )
```



0.10.6 further reading

For more on comparing multiple means in R: www.datanovia.com

For more on parametric versus non-parametric tests: Statistics by Jim

For more on interpreting *p* values: [The p value wars (again) by Ulrich Dirnagl]

0.10.7 exercises

Using the `hawaii_aquifers` data set, please complete the following:

1. Choose one analyte and filter the data so only the rows for that analyte are shown.
2. Choose two of the aquifers. Are the mean abundances for your chosen analyte different in these two aquifers? Don't forget to test your data for normality and homogeneity of variance before selecting a statistical test. Use a plot to illustrate whether the means are similar or different.
3. Choose a second analyte, different from the first one you chose. Considering all the aquifers in the dataset, do any of them have the same abundance of this analyte? Again, don't forget about normality and homogeneity of variance tests. Use a plot to illustrate your answer.
4. Repeat #3 above, but switch the type of test used (i.e. use non-

parametric if you used parametric for #3 and vice-versa). Compare the *p* values and *p* groups obtained by the two methods. Use a graphic to illustrate this. Why are they different?

____

____

____

0.11 (PART) instrumental analysis

0.12 chromatography and separations

The content in this part is based heavily on Harvey's Analytical Chemistry 2.1 Chapter 12.

0.12.1 Chromatographic and Electrophoretic Methods

Drawing from an arsenal of analytical techniques, analytical chemists design methods that detect increasingly smaller concentrations of analyte in increasingly more complex matrices. Despite the power of these analytical techniques, they often suffer from a lack of selectivity. For this reason, many analytical procedures include a step to separate the analyte from potential interferents. Although effective, each additional step in an analytical procedure increases the analysis time and the cost of the analysis, and introduces uncertainty. In this chapter we consider two analytical techniques that avoid these limitations by combining the separation and analysis: chromatography and electrophoresis.

0.12.1.1 Chromatographic Separations

In chromatography we pass a sample-free phase, which we call the mobile phase, over a second sample-free stationary phase that remains fixed in space. We inject or place the sample into the mobile phase. As the sample moves with the mobile phase, its components partition between the mobile phase and the stationary phase. A component whose distribution ratio favors the stationary phase requires more time to pass through the system. Given sufficient time and sufficient stationary and mobile phase, we can separate solutes even if they have similar distribution ratios.

There are many ways in which we can identify a chromatographic separation: by describing the physical state of the mobile phase and the stationary phase;

by describing how we bring the stationary phase and the mobile phase into contact with each other; or by describing the chemical or physical interactions between the solute and the stationary phase. Let’s briefly consider how we might use each of these classifications.

0.12.1.1.1 Types of Mobile Phases and Stationary Phases

The mobile phase is a liquid or a gas, and the stationary phase is a solid or a liquid film coated on a solid substrate. We often name chromatographic techniques by listing the type of mobile phase followed by the type of stationary phase. In gas–liquid chromatography, for example, the mobile phase is a gas and the stationary phase is a liquid film coated on a solid substrate. If a technique’s name includes only one phase, as in gas chromatography, it is the mobile phase.

0.12.1.1.2 Contact Between the Mobile Phase and the Stationary Phase

There are two common methods for bringing the mobile phase and the stationary phase into contact. In column chromatography we pack the stationary phase into a narrow column and pass the mobile phase through the column using gravity or by applying pressure. The stationary phase is a solid particle or a thin liquid film coated on either a solid particulate packing material or on the column’s walls. In planar chromatography the stationary phase is coated on a flat surface—typically, a glass, metal, or plastic plate. One end of the plate is placed in a reservoir that contains the mobile phase, which moves through the stationary phase by capillary action. In paper chromatography, for example, paper is the stationary phase.

```
knitr::include_app("https://dthdpu.shinyapps.io/IntroChromatography/")
## PhantomJS not found. You can install it with webshot::install_phantomjs(). If it is ins
```

0.13 mass spectrometry

0.13.1 Things

0.14 spectroscopy

0.14.1 Things

0.15 electrochemistry

0.15.1 Things

____

____

____

0.15.2 subsetting

So far, we have always been passing whole data sets to ggplot to do our plotting. However, suppose we wanted to get at just certain portions of our dataset, say, specific columns, or specific rows? Here are a few ways to do that:

```
# To look at a single column (the third column)
alaska_lake_data[,3]
## # A tibble: 220 x 1
##    water_temp
##         <dbl>
##  1       6.46
##  2       6.46
##  3       6.46
##  4       6.46
##  5       6.46
##  6       6.46
##  7       6.46
##  8       6.46
##  9       6.46
## 10       6.46
## # ... with 210 more rows

# To look at select columns:
alaska_lake_data[,2:5]
## # A tibble: 220 x 4
##    park  water_temp    pH element
##    <chr>      <dbl> <dbl> <chr>
##  1 BELA        6.46  7.69 C
##  2 BELA        6.46  7.69 N
##  3 BELA        6.46  7.69 P
##  4 BELA        6.46  7.69 Cl
##  5 BELA        6.46  7.69 S
##  6 BELA        6.46  7.69 F
##  7 BELA        6.46  7.69 Br
##  8 BELA        6.46  7.69 Na
##  9 BELA        6.46  7.69 K
## 10 BELA        6.46  7.69 Ca
## # ... with 210 more rows
```

```
# To look at a single row (the second row)
alaska_lake_data[2,]
## # A tibble: 1 x 7
##   lake  park  water_temp    pH element mg_per_L element_type
##   <chr> <chr>      <dbl> <dbl> <chr>      <dbl> <chr>
## 1 Devi~ BELA        6.46  7.69 N          0.028 bound

# To look at select rows:
alaska_lake_data[2:5,]
## # A tibble: 4 x 7
##   lake  park  water_temp    pH element mg_per_L element_type
##   <chr> <chr>      <dbl> <dbl> <chr>      <dbl> <chr>
## 1 Devi~ BELA        6.46  7.69 N          0.028 bound
## 2 Devi~ BELA        6.46  7.69 P          0     bound
## 3 Devi~ BELA        6.46  7.69 Cl        10.4   free
## 4 Devi~ BELA        6.46  7.69 S          0.62  free

# To look at just a single column, by name
alaska_lake_data$pH
##   [1] 7.69 7.69 7.69 7.69 7.69 7.69 7.69 7.69 7.69 7.69 7.69
##  [12] 6.44 6.44 6.44 6.44 6.44 6.44 6.44 6.44 6.44 6.44 6.44
##  [23] 7.45 7.45 7.45 7.45 7.45 7.45 7.45 7.45 7.45 7.45 7.45
##  [34] 7.42 7.42 7.42 7.42 7.42 7.42 7.42 7.42 7.42 7.42 7.42
##  [45] 8.04 8.04 8.04 8.04 8.04 8.04 8.04 8.04 8.04 8.04 8.04
##  [56] 7.82 7.82 7.82 7.82 7.82 7.82 7.82 7.82 7.82 7.82 7.82
##  [67] 7.01 7.01 7.01 7.01 7.01 7.01 7.01 7.01 7.01 7.01 7.01
##  [78] 7.03 7.03 7.03 7.03 7.03 7.03 7.03 7.03 7.03 7.03 7.03
##  [89] 6.95 6.95 6.95 6.95 6.95 6.95 6.95 6.95 6.95 6.95 6.95
## [100] 7.15 7.15 7.15 7.15 7.15 7.15 7.15 7.15 7.15 7.15 7.15
## [111] 6.88 6.88 6.88 6.88 6.88 6.88 6.88 6.88 6.88 6.88 6.88
## [122] 6.45 6.45 6.45 6.45 6.45 6.45 6.45 6.45 6.45 6.45 6.45
## [133] 6.88 6.88 6.88 6.88 6.88 6.88 6.88 6.88 6.88 6.88 6.88
## [144] 7.22 7.22 7.22 7.22 7.22 7.22 7.22 7.22 7.22 7.22 7.22
## [155] 6.98 6.98 6.98 6.98 6.98 6.98 6.98 6.98 6.98 6.98 6.98
## [166] 6.34 6.34 6.34 6.34 6.34 6.34 6.34 6.34 6.34 6.34 6.34
## [177] 7.24 7.24 7.24 7.24 7.24 7.24 7.24 7.24 7.24 7.24 7.24
## [188] 6.56 6.56 6.56 6.56 6.56 6.56 6.56 6.56 6.56 6.56 6.56
## [199] 7.31 7.31 7.31 7.31 7.31 7.31 7.31 7.31 7.31 7.31 7.31
## [210] 6.87 6.87 6.87 6.87 6.87 6.87 6.87 6.87 6.87 6.87 6.87

# To look at select columns by name
select(alaska_lake_data, park, water_temp)
## # A tibble: 220 x 2
##    park  water_temp
##    <chr>      <dbl>
##  1 BELA        6.46
##  2 BELA        6.46
##  3 BELA        6.46
##  4 BELA        6.46
##  5 BELA        6.46
##  6 BELA        6.46
##  7 BELA        6.46
##  8 BELA        6.46
##  9 BELA        6.46
## 10 BELA        6.46
## # ... with 210 more rows
```

Part III

writing

a mini manuscript

For your final project in this course you will use the techniques we have learned in class to analyze a large dataset, prepare high quality figures, and write a miniature manuscript describing the results.

The manuscript will be comprised of a title, abstract, introduction, results and discussion section, figures and captions, conclusions section, and at least five references. Please note the following when preparing your manuscript:

The orders of presentation and preparation do not have to be the same! While in some instances a scientist may choose to write the components of a manuscript in the same order in which they appear on the page, this is not always the case. The order of preparation suggsted above is designed to minimize the amount of revision / re-writing that needs to be performed during the manuscript preparation process. Note that the suggested order of composition is in line with the class schedule for the rest of the semester.

0.16 figures & captions

A high quality figure is one in which, for example, axes tick labels do not overlap but also fill the space available to them, colors are used, raw data is plotted (if possible), axes labels are customized, an appropriate theme is chosen, and geoms are chosen carefully. The plots should be visually attractive and professional.

Components of a caption:

1. Title - an overall description of the what is shown
2. For each subplot:

- The type of plot (line plot, bar chart, etc.)
- Describe what is plotted as y vs x in words.
- Describe where the data are from.
- Describe what each bar, point, or error bar represents.
- If applicable, describe the number of independent samples or measurements (sometimes called "replicates") that underlie a given geometric feature or summary statistic.

3. Avoid abbreviations, but if you do use any, specify what they mean.

An example:

```
ggplot(
  data = filter(alaska_lake_data, element_type == "bound"),
  aes(y = lake, x = mg_per_L)
) +
  geom_col(
    aes(fill = element),
    alpha = 0.5, size = 0.5, position = "dodge",
    color = "black"
  ) +
  facet_grid(park~., scales = "free", space = "free") +
  theme_bw() +
  scale_fill_brewer(palette = "Set1") +
  scale_y_discrete(name = "Lake Name") +
  scale_x_continuous(name = "Abundance mg/L)") +
  theme(
    text = element_text(size = 14)
  )
```

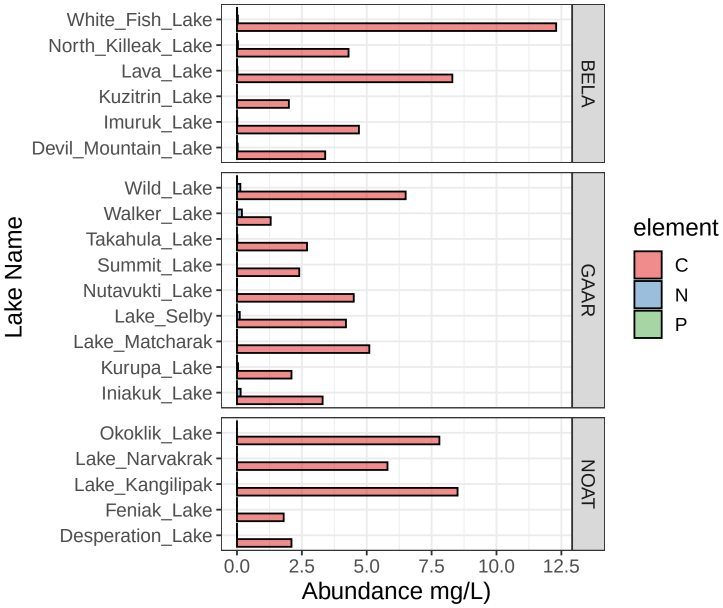


Fig. 1: Carbon, nitrogen, and phosphorous in Alaskan lakes. A bar chart showing the abundance (in mg per L, x-axis) of C, N, and P in various Alaskan lakes (lake names on y-axis) that are located in one of three parks in Alaska (park names on right y groupings). The data are from a public chemistry data repository. Each bar represents the result of a single measurement of a single analyte, the identity of which is coded using color as shown in the color legend. Abbreviations: BELA - Bering Land Bridge National Preserve, GAAR - Gates

Of The Arctic National Park & Preserve, NOAT - Noatak National Preserve. –>

0.17 results and discussion

- Objective: Walk your reader through your results, drawing conclusions and making interpretations as you go.
- As you go: make notes of what should go into the introduction.

0.17.1 structure

- Key: **number of suggested sentences**: *purpose*: "example"
- Introductory paragraph:
  - **1**: *Review the aim of the paper*: "In order to understand…"
  - **3-4**: *Combine a methods summary (with generalized summary of results?) to call out subsections*: "We used method X to quantify property Y of our study subject (section 2.1)"
- Each subsection paragraph:
  - **1**: *Purpose of the work described in this paragraph*: "In order to determine…"
  - **1**: *Review methods or experimental design specific to this subsection (if necessary)*
  - **4-5**: *Results of that method or experiment (i.e. data features)*
  - **1-2**: *Comparison of new results against those in literature (if possible)*
  - **1-2**: *Conclusion from the combined results or some other concluding remark* "Thus, analysis X revealed that…"
  - **1**: *Interpretation of the conclusion in a larger context (if possible / reasonable)*
- Let's look at an example:

  2. Results and Discussion.

  In order to better understand pollution in the state of Minnesota, this study focused on a detailed analyses of chemical measurements from soil samples from 300 sites around the state. The analyses consisted of a principal components analysis to determine which sites were similar to one another (Section 2.1) followed by statistical tests to see whether differences could be detected in the sites' chemistry (Section 2.2).

2.1 Principal Components Analysis

To understand relationships between the sites from which soil chemistry was sampled, a principal components analysis was used. Each of the 20 different analytes, all of which contained halogen atoms, were included in the analysis. A scatter plot showing the position of each of the 300 samples in a space defined by dimesions 1 and 2 (which explain 54% and 35% of the total variance of the dataset, respectively), revealed that two major clusters are present, with a small number of outliers (Fig. 1). By color coding these two clusters according to whether the samples were collected from rural versus urban areas, it was possible to see that the first cluster was made out of almost exclusively samples from urban areas, while the second cluster was made up of almost entirely samples from rural areas. This suggested that variance in pollutant chemistry among the samples collected was assocaited with urban versus rural environments.

2.2 Statistical Analyses

Using the groupings that were identified via principal components analysis, statistical tests were conducted to determine if chemical abundances differed between groups. Tests for normality and homogeneity of variance (Shapiro and Levene tests) revealed that the data could not be assessed using ANOVA but instead required the use of a non-parametric test. Accordingly, the Kruskall-Wallis test followed by post-hoc Dunn tests were applied, which showed that the abundances of halogenated pollutants is significantly higher in urban versus rural areas ($p = 0.0035$, Fig. 2A). These direct observations are consistent with conclusions drawn by others in recent literature reviews focused on hydrocarbon compounds (Petrucci et al., 2018; Hendrix et al., 2019). *Thus, the new chemical analyses presented here demonstrate that the discrepancy in urban versus rural pollution is true not only for hydrocarbon compounds (as had been found previously), but also for halogenated compounds.* Together, these findings strongly suggest that either cities are a source of more pollution or that there is some other mechanism that concentrates pollution in cities.

0.17.2 suggestions

1. **Create paragraph outlines:**

- Identify “data features” in your figures, then possible conclusions those could lead to. Example:
  - “The GC-MS data presented here indicates that cities have higher levels of pollution than rural areas (Fig. 1),” (a data feature)

  - “suggesting that either cities are a source of more pollution or that there is some other mechanism that concentrates pollution in cities.” (a conclusion)
- Expand your “data feature” -> “conclusion” combinations with “supplementary information” or “literature information”. Example:
  - “The GC-MS data presented here indicates that cities have higher levels of pollution than rural areas.” (data feature)
  - “These direct observations are consistent with conclusions drawn by others in recent literature reviews (Petrucci., 2018; Hendrix et al., 2019)”
  - “Overall, this suggests that either cities are a source of more pollution or that there is some other mechanism that concentrates pollution in cities.”

2. **Write drafts of your paragraphs:**

- Combine each of your “data feature” -> “supp/lit info” -> “conclusion” combinations into a single paragraph.
- Consider editing each paragraph so that it highlights what new contribution your data makes to the situation. Example (note the sentence in italics that highlights the new findings):
  - ”The GC-MS data presented here indicates that cities have higher levels of pollution than rural areas (Fig. 1). These direct observations are consistent with meta-analyses of previously published observations (Supplemental Figure 1), as well as with conclusions drawn by others in recent literature reviews (So and so et al., 2018; The other person et al., 2019). *The new chemical analyses presented here demonstrate that this is true not only for hydrocarbon compounds (as had been found previously), but also for halogenated compounds in the atmosphere.* Together these findings strongly suggest that either cities are a source of more pollution or that there is some other mechanism that concentrates pollution in cities.

3. **Order your paragraphs:**

- Identify characteristics of your paragraphs that can help determine what order they should go in:
  - Whether any of your paragraphs are prerequisites for others.
  - Whether any paragraphs can be grouped according to topic.
- Group paragraphs according to topic and prerequisite dependencies (putting prereq dependencies as close to eachother as possible.)
- Rearrange groups for what seems like the most natural flow. Consider:
  - Starting with group of paragraphs most relevant to the overall pitch/goal of the paper

  - Ending on the group of paragraphs that has the most future perspective
  - Ending in a strong suit (i.e. not something too speculative)

- After this, if you have any orphaned paragraphs, consider putting them (or a shortened version of them) into the conclusion section.

- Throughout this process, read lots of literature and incorporate it into your discussion section. Place your research into the context of what has been done previously.

4. **Edit your results and discussion section as a whole:**

- Edit each paragraph, particularly its first and last sentences, to connect the paragraphs into a flowing document. Specifically, this means several things:
  - There should be no implicit cross-paragraph references (i.e. a new paragraph should not begin “The compound described above exhibited other interesting properties”, rather, “3-hydroxycinnamic acid exhibited other interesting properties.”).
  - There should be no abrupt jumps in subject between paragraphs, if there are consider breaking the discussion into subsections to help the reader identify logical resting points.
  - The discussion should not require the reader to go back and read its first half in order to understand its second half.

0.18 conclusion and introduction

- Objective (conclusion): to convey a short statement of the take-home messages of your study. What are the most important things that you want the reader to remember from your study?

- Objective (introduciton): to prepare the reader by giving the reader sufficient background to understand the study as a whole. It therefore should only contain information pertinent to understanding the study and its broader significance.

- Make sure that the scope of your introduction is in-line with the scope of the conclusion. That way, the reader will not be underwhelmed, nor will your work be undersold.

0.18.1 structure

Conclusion:

- *One paragraph*

  - **2-3**: Summarize over-arching conclusions from each section of the paper (omit the details described in results or discussion)
  - **2-3**: Based on a general description of findings, use pros and cons to argue for, if possible, alternative hypotheses.
  - **1-2**: Suggest experiments to test these hypotheses.
  - **1-2**: Describe future directions.

Introduction:

- *Paragraph 1: Introduce the topic*
  - **1**: Introduce a topic and, ideally, an application of the research you will describe. Grab reader's attention.
  - **1**: State why the topic is important.
  - **1**: Describe what is known about the topic (at least, as pertains to the work at hand).
  - **1**: Identify a gap in knowledge: "despite research in this area, here is what we don't know about the topic."
  - **1**: List the negative things that will happen if we don't fill this gap in knowledge.
- *Paragraph 2: Provide background information*
  - **3-5**: Describe, in moderate detail, the background information (concepts, literature) relevant to the study.
  - **1**: End by saying how the details you just described relate to the application/topic described in the first paragraph.
- *Paragraph 3: Objectives of this study*
  - **1**: State the objective of this study.
  - **1**: Briefly describe what was done and the techniques or instruments used.
  - **1-2**: For this project, briefly describe where you got the data, how you cleaned it up, if you merged multiple datasets, etc.
  - **1**: (optional) State the major conclusion from the work and what it means for the application described in paragraph 1.

0.18.2 suggestions

- If something is well-established, say so.
- Be clear about what is speculation.
- Last paragraph can mention objectives in list form.
- Last sentence can briefly mention methods (specific techniques or instruments) that were used.

0.19 abstract and title

0.19.1 abstract

- **Structure** *One paragraph* Use about 200 - 500 words (ideally no more than 400 words)
    - **1-2 sentences**: Introduction: Describe the topic, the motivation, and overall purpose of the research (Why is this research interesting and important? What gap in our knowledge does it fill?)
    - **1-2 sentences**: Objective: Specific research objective, and potentially hypotheses/predictions, if any.
    - **1-2 sentences**: Methods: Very concise overview of the methods used to address the research questions.
    - **2-3 sentences**: Results/Discussion: Describe the major results (what you found) and interpretation of the results (what the results mean).
    - **1-2 sentences**: Conclusions: Synthesizes the major contributions of the study into the context of the larger field to which the study belongs. What did we learn about the bigger picture of this field in general from doing this study?
- **Function: an abstract proves a short summary of the entire study.** The abstract should include the motivation or reason for conducting the study, what the research question or hypothesis was, how the experiments were conducted, what the results were, how the results are interpreted in light of the research question or hypothesis, and a concluding sentence about the general contribution or importance of the study. A good abstract should:
    - Inform readers about the article's content
    - Summarize complex information in a clear, concise manner
    - Help readers decide whether or not to read the article
    - Used in conferences to summarize what the speaker will say during his/her presentation

0.19.2 title

- **Structure** *One sentence* Use about 75-140 characters (ideally no more than 125 characters). There are essentially two types of titles: descriptive titles and mechanistic titles.
    - If your manuscript is exploratory research, consider using a descriptive title. For example:
        - “Comparative analysis of carbon, sulfur, and phoshorous chemistry in six Alaskan lakes.”
    - If your manuscript is hypothesis-driven research, consider using a mechanistic title. For example:
        - “Dissolved organic carbon in Alaskan lakes is heavily influenced by water pH and temperature.”
- **Function: a title captures attention and highlight the research question(s).** A good title should:
    - Be indicative of the content of the paper

  - Attract the interest of potential readers
  - Reflect whether the article is deascriptive or mechanistic
  - Include important keywords

0.19.3 further reading

- Titles Guide

IMAGE ANALYSIS

0.20 image color analysis

For analyze color images we use an interactive app called by `analyzeImage()`. It takes two arguments: `share_link`, and `monolist_out_path`. `share_link` should be the Google Drive share link for the photo that you wish to analyze. `share_link` can also be the share link for a Google Drive folder, in which case the app will allow you to cycle through the photos in that folder one-by-one. `monolist_out_path` should be a path to a new or existing .csv file on your local sytem where the results are to be saved as you work. Below is an example. Remember, if you are on Mac you should use a path that has single slashes, for example: `/Users/bust0037/Desktop/output.csv`. If you are on PC you should use a path that has double slashes, for example: `C://Users//Busta_Lab//Desktop//output.csv`.

```
analyzeImage(
  share_link = "https://drive.google.com/file/d/1rvfh9_DqEWlpaegGwfLZLdjjYEDlMOZL/view?usp
  monolist_out_path = "/Users/bust0037/Desktop/output.csv"
)
```

0.21 images of mass spectra

```
analyzeMassSpectralImages()
```

0.22 phylochemistry

phylochemistry is a set of functions for chemical, transcriptomic, and genomic analysis. These tools are provided though a combination of new computational functions and wrapped features of previously developed packages. A number of new organizational and data handling functions to streamline analyses in this interdisciplinary space are also provided. This page provides access to the latest version of phylochemistry.

0.22.1 requirements

- To run phylochemistry, you need to have R and RStudio installed. For instructions on how to install those, please see this page.

0.22.2 load phylochemistry

Phylochemistry is not an R package, but rather a set of components that you can add to your R environment by running an R script hosted on this site. phylochemistry requires a number of existing R packages in order to run, but don't worry,phylochemistry will help you install these packages if they are not installed already.

1. Load phylochemistry directly into your R session by running the following command in RStudio:

```
source("http://thebustalab.github.io/phylochemistry/phylochemistry.R")
```

Sometimes running the command above generates the message "You need to install the following packages before proceeding […] Run: installPhylochemistry() to automatically install the required packages.". This means that some of the prerequisite packages that phylochemistry needs are not installed. If this happens, run the following:

```
installPhylochemistry()
```

Once that is complete, then try the source() command again:

```
source("http://thebustalab.github.io/phylochemistry/phylochemistry.R")
```

0.22.3 R scripts on Google Drive

Sometimes we want to save our R scripts on Google Drive. If you have an R script on Google Drive and want to open it in RStudio, get the share link for the file and use the following command:

```
openRGD("file_share_link_here")
```

When you do this, ”IN_USE____” will appear in front of the file name in Google Drive, so that others will know that you are using it. When you are done using the file, you can save and close it using:

```
closeRGD("file_share_link_here")
```

0.22.4 new features

1. A Shiny app for GC-FID and GC-MS data analysis, including a large MS library.
2. Open reading frame extraction from multiple fasta files.
3. BLAST searches that export .fasta files of hits and store results in a .csv file.
4. Minor ticks for ggplot2 axes.
5. Phylogenetic signal for discrete traits.
6. Analyze multiple sequence alignments for sites associated with user-defined function
7. Multiple column name, multiple row name data structures (aka “polylists”).
8. Draw annotated multiple sequence alignments.
9. Use image analysis to automatically get the csv of a mass spectrum from a published image.
10. Draw chemical structures in R from a csv of molecular coordinates.

0.22.5 wrapped features

1. BLAST transcriptomes, via NCBI BLAST+.
2. Multiple sequence alignments and codon alignments of amino acid and nucleotide sequences, via msa and orthologr.
3. Phylogenetic tree construction (including g-blocks trimming, pruning, ancestral states reconstruction), via phangorn.
4. Systematic read/write functions (csv, newick, wide tables, fasta, summary statistic tables, GFFs, chromatograms, mass spectra).
5. Phylogenetic signal for continuous traits, via picante.

0.23 mass spectrometric analysis

0.23.1 integrationAppLite

phylochemistry provides a simple application for integrating and analyzing GC-MS data. With it, you can analyze .CDF files, which contain essentially all the data from a GC-MS run, and can be exported from most GC-MS systems using the software provided by the manufacturer. Instructions for this are provided at the end of this chapter. To run the lite version of the integration app, use the following guidelines:

1. Create a new folder on your hard drive and place your CDF file into that folder. It doesn't matter what the name of that folder is, but it must not contain special characters (including a space in the name). For example, if my CDF file is called "sorghum_bicolor.CDF", then I might create a folder called gc_data on my hard drive, and place the "sorghum_bicolor.CDF" file in that folder.
2. In RStudio, run the source command to load phylochemistry:

```
source("http://thebustalab.github.io/phylochemistry/phylochemistry.R")
```

3. In RStudio, run the integrationAppLite command on the *folder* that contains your CDF file.

If you are on a Mac, *use single forward slashes*. For example:

```
integrationAppLite("/Volumes/My_Drive/gc_data")
```

If you are on a PC, *use double back slashes*. For example:

```
integrationAppLite("C:\\Users\\My Profile\\gc_data")
```

The first time you open your datafile, it may take a while to load. This is normal - the program is analyzing all the data points in your data file. For a typical exploratory GC-MS run of around 60 minutes, this is more than 2.5

million data points! So please be patient. After you open your data file once, subsequent openings will not take so long.

Please watch this overview video for a demonstration of how to use the integration app.

As a reference, below are the key commands used to operate the integration app. This is the information that is covered in the overview video.

To control the chromatogram window:

- shift + q = update
- shift + a = add selected peak
- shift + r = remove selected peak
- shift + f = forward
- shift + d = backward
- shift + c = zoom in
- shift + v = zoom out
- shift + z = save table

To control the mass spectrum window:

- shift+1 = extract mass spectra from highlighted chromatogram region, plot average mass spectrum in panel 1.
- shift+2 = refresh mass spectrum in panel 1. This is used for zooming in on a region of the mass spectrum that you have highlighted. A spectrum needs to first be extracted for this to be possible.
- shift+3 = extract mass spectra from highlighted chromatogram region, subtract their average from the mass spectrum in panel 1.
- shift+4 = search current spectrum in panel 1 against library of mass spectra.

0.23.2 CDF export

1. On the GC-MS computer, open Enhanced Data Analysis
2. File > Export Data To .AIA Format, Create New Directory (“OK”) > Desktop (create a folder with a name you will remember)
3. Select all the datafiles you wish to analyze and process them, saving the output into the folder you just created
4. Copy the .D files for the samples you wish to analyze to the same folder
5. Move this folder to your personal computer
6. Create one folder for each sample, and put the corresponding .CDF file into that folder.

0.24 transcriptomic analyses

0.24.1 BLAST

0.25 genomic analyses

0.25.1 loading GFF files

0.26 evolutionary analyses

0.26.1 buildTree

0.26.1.1 Simple template

```
buildTree(
  scaffold_type = "newick",
  scaffold_in_path = NULL,
  members = NULL
)
```

0.26.1.2 Full template

```
buildTree(
  scaffold_type = c("amin_alignment", "nucl_alignment", "newick"),
  scaffold_in_path = NULL,
  members = NULL,
  gblocks = FALSE,
  gblocks_path = NULL,
  ml = FALSE,
  model_test = FALSE,
  bootstrap = FALSE,
  rois = FALSE,
  rois_data = NULL,
  ancestral_states = FALSE,
  root = NULL
)
```

0.26.2 collapseTree

APPENDIX

0.27 links

0.27.1 geoms

geoms and ggplot2 cheatsheet

0.27.2 colors

ColorBrewer2

0.28 faq

0.28.1 filtering

`filter(<data>, <variable> < 18)` ## less than 18

`filter(<data>, <variable> <= 18)` ## less than or equal to 18

`filter(<data>, <variable> > 18)` ## greater than 18

`filter(<data>, <variable> >= 18)` ## greater than or equal to 18

`filter(<data>, <variable> == 18)` ## equals than 18

`filter(<data>, <variable> != 18)` ## not equal to 18

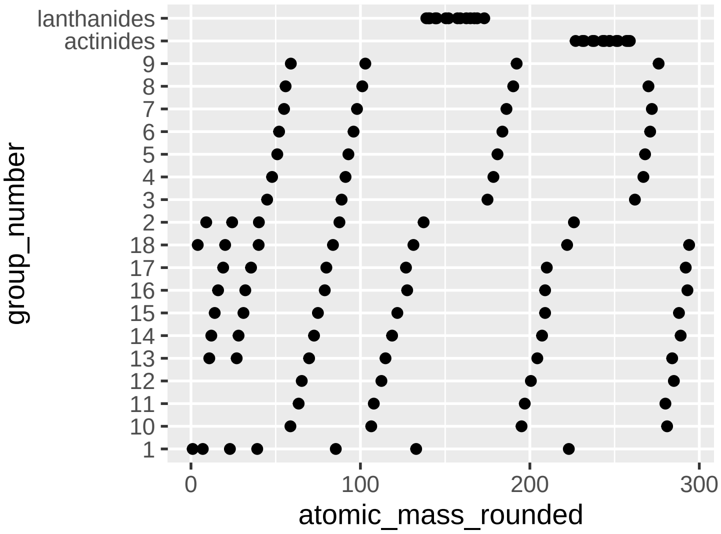
`filter(<data>, <variable> == 18 | <variable> == 19)` ## equal to 18 or 19

0.28.2 ordering

A list of numeric element has an inherent order to it: -inf -> +inf. A list of character element also has an inherent order to it: A -> Z, or if it's a mixed number and letter list (which is interpreted by R as a character list): 0 -> 9 -> A -> Z.

However, there are cases where we will want a list of character elements to have some order other than A -> Z. In these cases, we want to convert the list of character elements into a list of factor elements. Factors are lists of character elements that have an inherent order that is not A -> Z. For example, in the plot below, the y axis is not, perhaps, in the “correct” order:

```
ggplot(periodic_table) +
  geom_point(aes(y = group_number, x = atomic_mass_rounded))
```

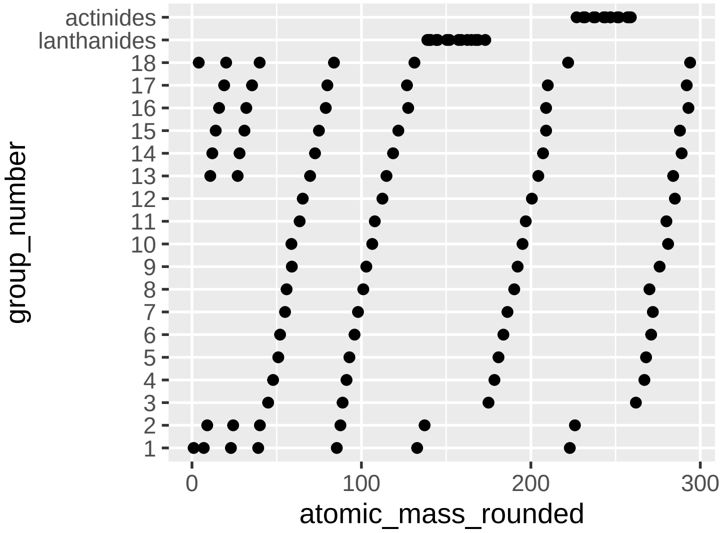


How do we fix this? We need to convert the column `group_number` into a list of factors that have the correct order (see below). For this, we will use the command `factor`, which will accept an argument called `levels` in which we can define the order the the characters should be in:

```
periodic_table$group_number <- factor(
  periodic_table$group_number,
  levels = c("1", "2", "3", "4", "5", "6", "7", "8", "9", "10", "11", "12", "13", "14", "1
)

periodic_table
## # A tibble: 118 x 41
##    atomic_number element_name atomic_symbol group_number
##            <dbl> <chr>        <chr>         <fct>
##  1             1 hydrogen     H             1
##  2             2 helium       He            18
##  3             3 lithium      Li            1
##  4             4 beryllium    Be            2
##  5             5 boron        B             13
##  6             6 carbon       C             14
##  7             7 nitrogen     N             15
##  8             8 oxygen       O             16
##  9             9 fluorine     F             17
## 10            10 neon         Ne            18
## # ... with 108 more rows, and 37 more variables:
## #   period <dbl>, atomic_mass_rounded <dbl>,
## #   melting_point_C <dbl>, boiling_point_C <dbl>,
## #   state_at_RT <chr>, density_g_per_mL <dbl>,
## #   electronegativity_pauling <dbl>,
## #   first_ionization_poten_eV <dbl>,
## #   second_ionization_poten_eV <dbl>, ...
```

Notice that now when we look at the type of data that is contained in the column `group_number` it says “”. This is great! It means we have converted that column into a list of factors, instead of characters. Now what happens when we make our plot?

```
ggplot(periodic_table) +
  geom_point(aes(y = group_number, x = atomic_mass_rounded))
```



VICTORY!

0.28.3 column manipulation

How to select specific columns:

```
alaska_lake_data %>%
  select(water_temp, pH)
## # A tibble: 220 x 2
##    water_temp    pH
##         <dbl> <dbl>
##  1       6.46  7.69
##  2       6.46  7.69
##  3       6.46  7.69
##  4       6.46  7.69
##  5       6.46  7.69
##  6       6.46  7.69
##  7       6.46  7.69
##  8       6.46  7.69
##  9       6.46  7.69
## 10       6.46  7.69
## # ... with 210 more rows
```

How to remove certain columns:

```
alaska_lake_data %>%
  select(!water_temp)
## # A tibble: 220 x 6
##    lake                park     pH element mg_per_L element_type
##    <chr>               <chr> <dbl> <chr>      <dbl> <chr>
##  1 Devil_Mountain_Lake BELA   7.69 C          3.4   bound
##  2 Devil_Mountain_Lake BELA   7.69 N          0.028 bound
##  3 Devil_Mountain_Lake BELA   7.69 P          0     bound
##  4 Devil_Mountain_Lake BELA   7.69 Cl        10.4   free
##  5 Devil_Mountain_Lake BELA   7.69 S          0.62  free
##  6 Devil_Mountain_Lake BELA   7.69 F          0.04  free
##  7 Devil_Mountain_Lake BELA   7.69 Br         0.02  free
##  8 Devil_Mountain_Lake BELA   7.69 Na         8.92  free
##  9 Devil_Mountain_Lake BELA   7.69 K          1.2   free
## 10 Devil_Mountain_Lake BELA   7.69 Ca         5.73  free
## # ... with 210 more rows
```

0.29 templates

0.29.1 matrix analyses

0.29.1.1 basic `runMatrixAnalysis()` template

```

runMatrixAnalysis(

  data = NULL,
  analysis = c("hclust", "pca", "pca_ord", "pca_dim"),

  column_w_names_of_multiple_analytes = NULL,
  column_w_values_for_multiple_analytes = NULL,

  columns_w_values_for_single_analyte = NULL,

  columns_w_sample_ID_info = NULL

)
```

0.29.1.2 advanced `runMatrixAnalysis()` template

```
runMatrixAnalysis(
  data = NULL, # the data set to work on
  analysis = c("hclust", "pca", "pca_ord", "pca_dim"), # the analysis to conduct
  column_w_names_of_multiple_analytes = NULL, # a column with names of multiple analytes
  column_w_values_for_multiple_analytes = NULL, # a column with quantities measured for mu
  columns_w_values_for_single_analyte = NULL, # a column with quantities measured for a si
  columns_w_additional_analyte_info = NULL, # a column with character or numeric informati
  columns_w_sample_ID_info = NULL, # a column with information about the sample (i.e. cont
  transpose = FALSE,
  kmeans = c("none", "auto", "elbow", "1", "2", "3", "etc."),
  na_replacement = c("none", "mean", "zero", "drop")
)
```