Week 4 - Plotting & Visualization

Cole & Emma

4/5/2020

Outline:

1. Data Visualization Rules
2. Speed overview of plotting in base R
3. ggplot2 – Basic Plotting
4. Aesthetics and Annotations
5. Themes

Data Visualization Guidelines (COLE)

Data visualization can be seen as equal parts art and science. Balancing a graphic that is beautiful, as well as informative, takes a lot of practice and most importantly a very strong understanding of the data you are visualizing, and how to let the data communicate their message effectively. It is incredibly important when doing data visualizations to be cognizant of three major guiding questions:

1. What is the message you were trying to communicate with your visualization?
2. Who is the audience you are trying to communicate your message to?
3. How can your data be best displayed to achieve this goal, while remaining true to the nuances inherent to the data themselves?

There is a whole field of study dedicated to the treatment of how to produce excellent and broadly applicable figures and plots. We do not hope to cover all or even some of the theory that is involved with data viz, though fascinating sub-topics of this include a) how the human eye tracks across aspects of a figure, b) how incorrect usage of different plotting components can be used to deceive the viewer, c) how storytelling and visuals go hand in hand, as well as the many other important components of data visualization. However, we think it's important to have a general understanding of how one can go about making good figures generally, as well as specifically in R.

There is a plethora of excellent (free) resources online for those of you interested in the more nuanced aspects of data visualization. Here, however, it might be useful to go over what one could consider seven important aspects of data visualization. There is an excellent article by *flowingdata.com* (https://flowingdata.com/2010/07/22/7-basic-rules-for-making-charts-and-graphs/) that describes these aspects in more detail. I encourage you to check out the full article, linked here, as well as some of the other excellent pieces by *flowingdata.* Here, we will briefly describe these seven key traits of visualization.

1. **Check the data.** This is, without doubt, the most important of all. It is all too easy to purposefully or by accident mislead viewers with your formulation and dissemination of data in the form of plots and figures. While there is nothing we can do about purposeful deception of data, avoiding accidental mistakes leading from the poor interpretation of your data is a problem we CAN fix - namely by ensuring great care and thought in the data cleaning aspect, as well as the careful consideration of how your message being told by the data will be percieved by others.
2. **Always explain your encodings.** This can also be translated as "include a figure legend".
3. **Label your axes.** Figure axes are the basis of understanding on what scale the data are being visualized. Ensure that you always have a descriptive label along with the values on each of your X and Y axes respectively.

4. **Include units.** That's it. Include yor units. Always.
5. **Keep your geometry in check.** This is less relevant to the types of visualizations we will be making in this course, but when including objects such as circles or other shapes, that are meant to represent

some sort of relative magnitude, ensuring that the actual size of those shapes is proportional to the relative difference in magnitude of the data is is important.

6. **Include your sources.** Periodt.
7. **Consider your audience.** This feeds into the guiding question we talked about above in terms of determining who you were trying to convey your message to. It is always continually ask yourself 'Who is this for, and what do they need to know?'. Ask this at the beginning stage when deciding what type of visualization you want to use, in the construction stage when figuring out all of the important details, and then in the final stage when you are linking your figure to your main message in the text. It is all too easy to forget that when we are writing about a particular piece of data, we are often experts on that data by the time we are visualizing it for a publication or paper. It is important to step back and consider who will be reading the work/viewing the figures, and what they will get out of it when in reality it is unlikely that any reader will spend more than a few seconds initially looking at your figure.

Quick Plotting With BaseR Graphics (EMMA)

Most of this session is going to focus on plotting using a package called `ggplot2` but you can also make figures using built-in functions like `plot()`. It is totally personal preference regarding which you prefer. I (Emma) mostly plot in base R and Cole mostly plots with ggplot. Whatever gets the job done. For now, I'll just do a very cursory overview of plotting in base R.

First, let's make up some fake data.

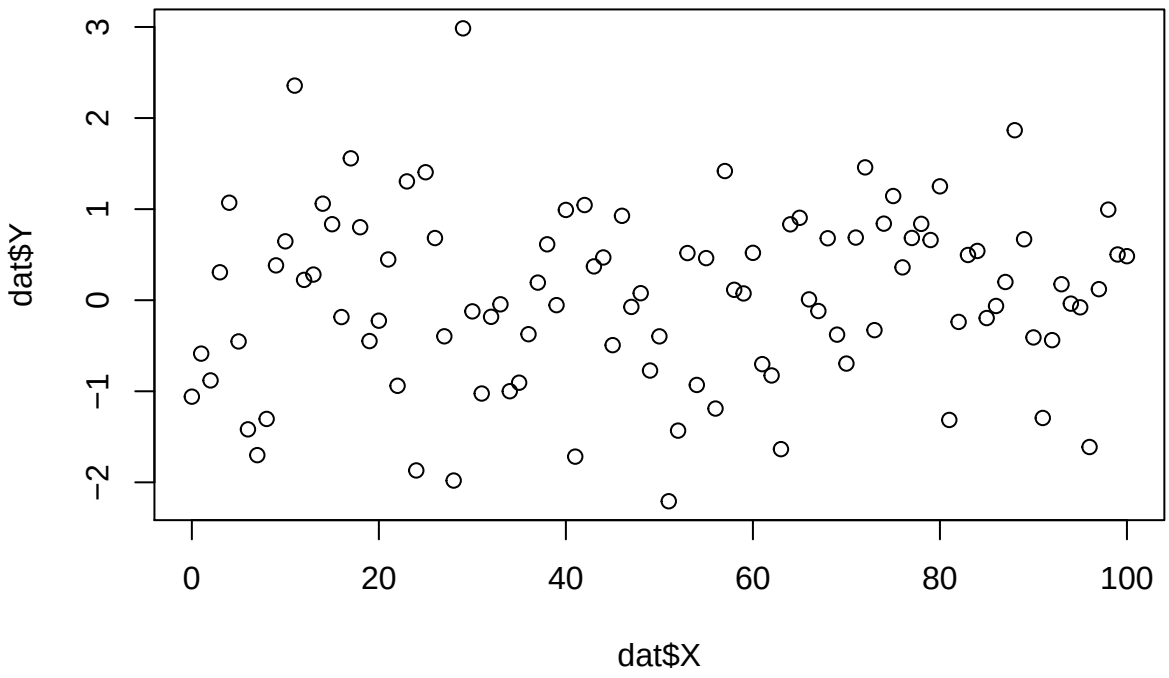
```
# setting indep. variable as a sequence from 0 to 100, increasing by increments of 1
x <- seq(0,100,1)
# drawing 101 random numbers from a normal distribution for our independant variable
y <- rnorm(101)

dat <- data.frame(X=x, Y=y) # put it in a dataframe
head(dat) # take a look!
```

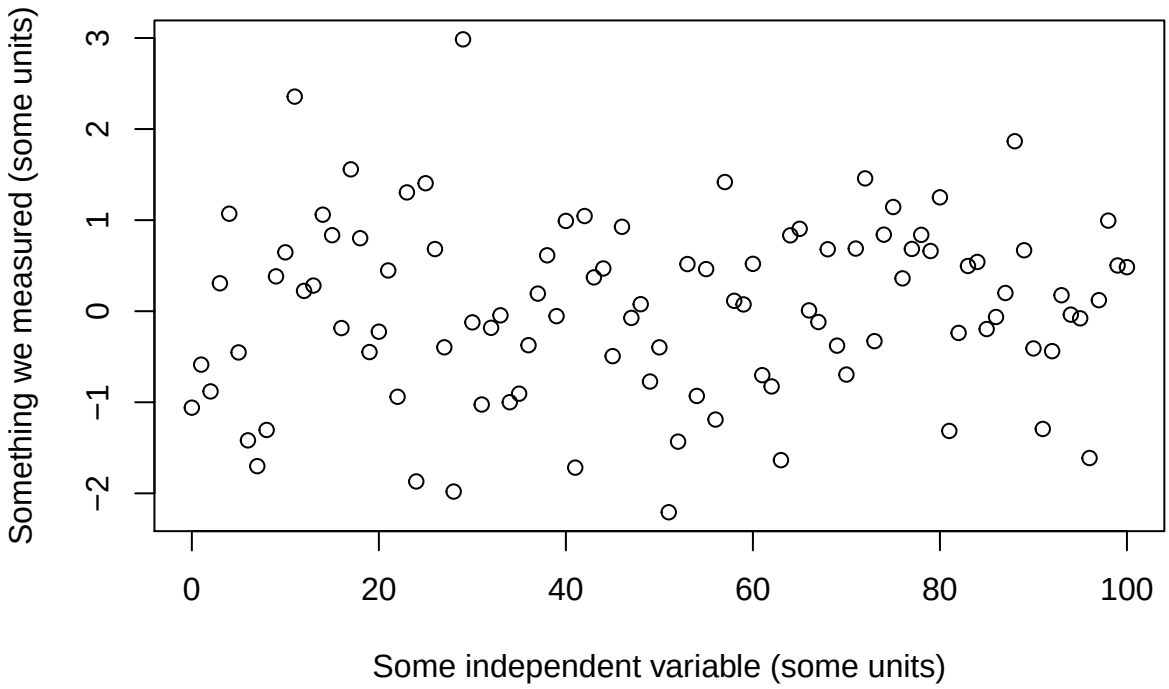
```
##   X          Y
## 1 0 -1.0590199
## 2 1 -0.5866098
## 3 2 -0.8807595
## 4 3  0.3065140
## 5 4  1.0704335
## 6 5 -0.4527518
```

OK, so we've got data to plot. It can be overwhelming to look at someone's finished script and try to interpret each component, especially with plotting code. I am going to start simple and build in components piecewise to give you a sense for the process you might follow if making a real figure. I tend to start with just a rough, stripped down version of the plot I think I want to make. And then build on it from there.

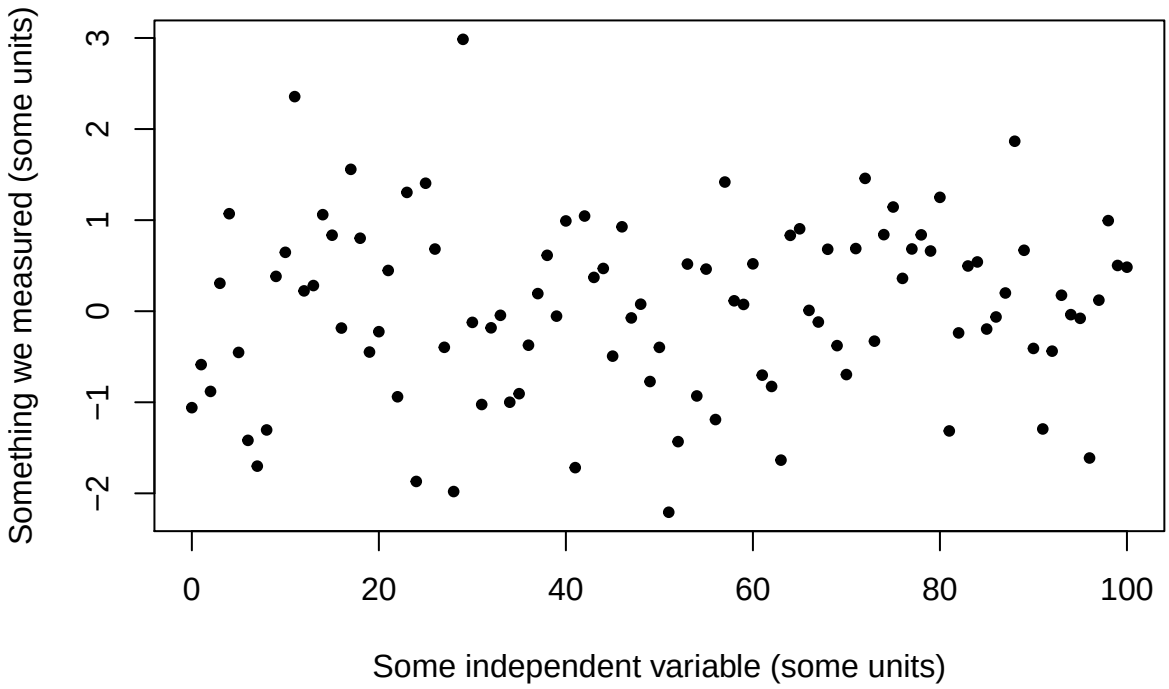
```
# First, let's make the simplest version
plot(dat$X, dat$Y)
```



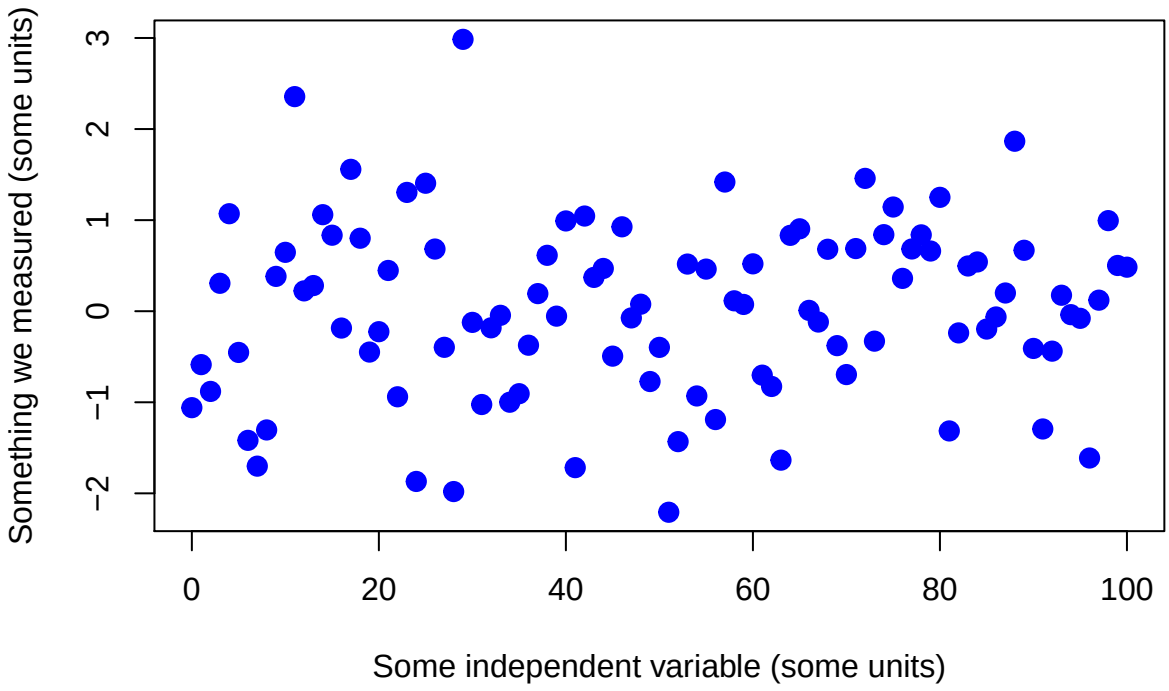

```
# Maybe I want to change the axis labels
plot(dat$X, dat$Y, xlab="Some independent variable (some units)",
     ylab="Something we measured (some units)")
```



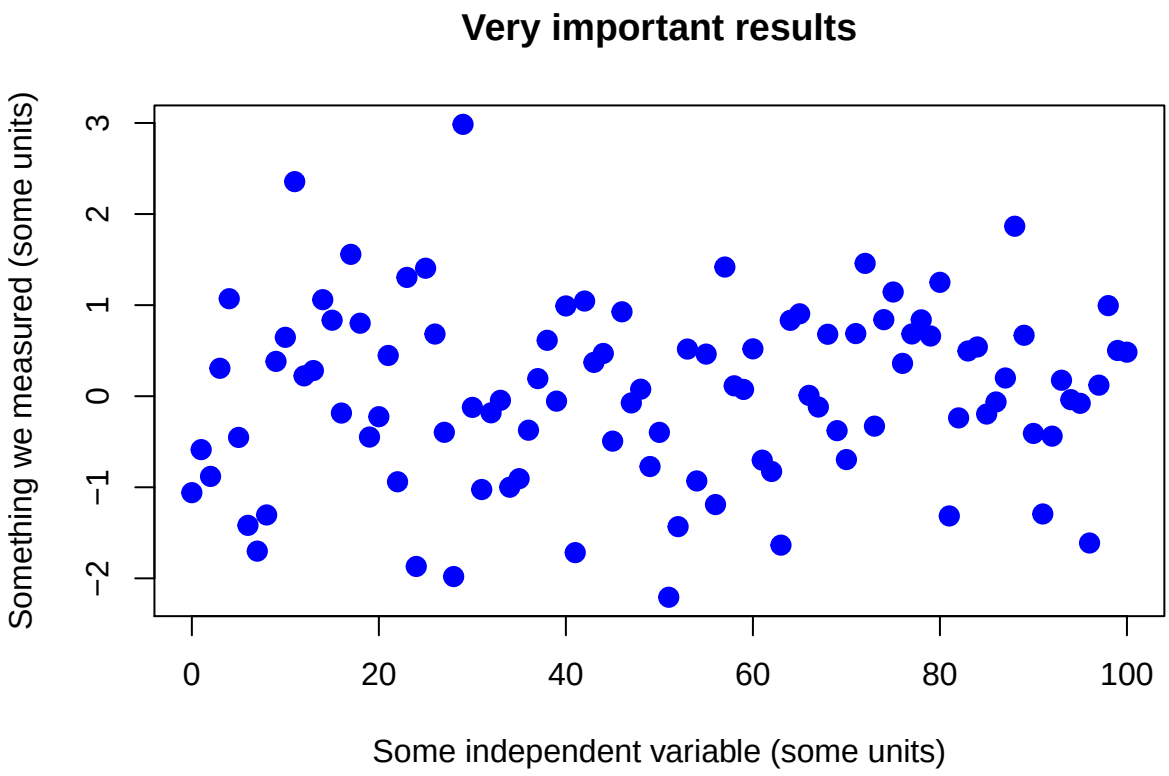

```
# I don't really like how those points look so I'll change the shape
plot(dat$X, dat$Y,
     xlab="Some independent variable (some units)",
     ylab="Something we measured (some units)",
     pch=20)
```



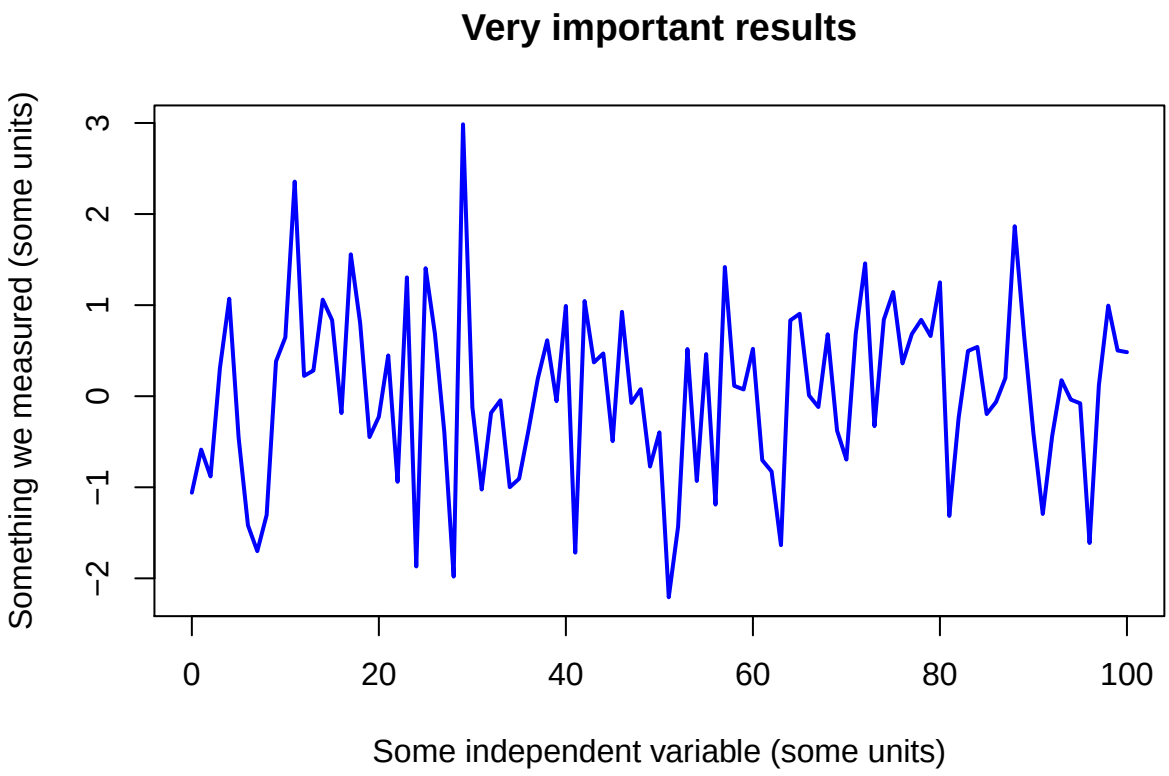

```
# Make them bigger and change their colour
plot(dat$X, dat$Y,
     xlab="Some independent variable (some units)",
     ylab="Something we measured (some units)",
     pch=20,
     cex=2,
     col="blue")
```




```
# Now I want a title
plot(dat$X, dat$Y,
     xlab="Some independent variable (some units)",
     ylab="Something we measured (some units)",
     pch=20,
     cex=2,
     col="blue",
     main="Very important results")
```

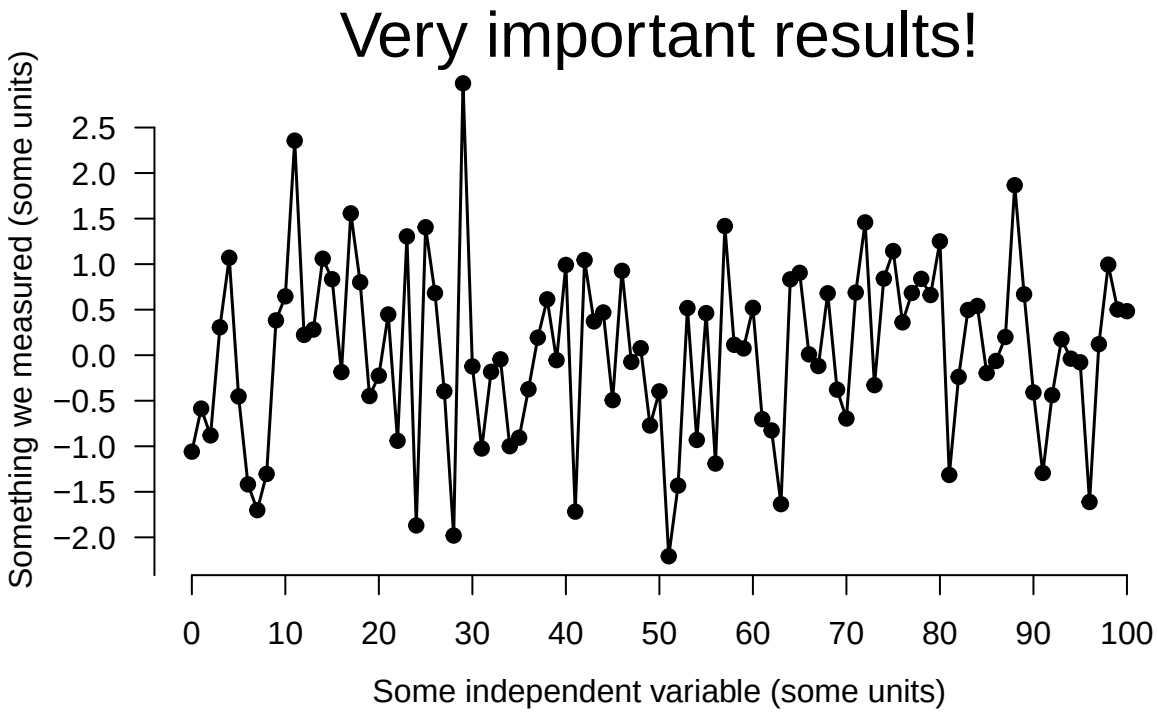



```
# If these data were consecutive measurements over time
#(like a time series of abundance data),
#I could plot them with a line instead
plot(dat$X, dat$Y,
     xlab="Some independent variable (some units)",
     ylab="Something we measured (some units)",
     type="l", # line type
     lwd=2, # line thickness
     col="blue",
     main="Very important results")
```



There are a whole suite of other arguments I can include in my plot() function call to further customize my plot. I still find myself Googling things like "how to change orientation of axis tick labels in R" all the time. Another valuable resource is the help page for plot() which can be accessed by typing `?plot` in your console. My favourite part about using base R for plotting is the ability to start from scratch and basically add each piece of the plot incrementally to get it exactly the way you want it.

```
plot(dat$X, dat$Y,
     col="white",
     bty="n",
     axes=FALSE,
     xlab="",
     ylab="")
lines(dat$X, dat$Y, col="black",lwd=1.5)
points(dat$X, dat$Y, col="black", cex=1.5, pch=20)
axis(side=1, at=seq(0,max(dat$X),10), labels=TRUE)
axis(side=2, at=seq(-2.5,2.5,0.5), labels=TRUE, las=2) #'las' rotates the axis tick labels
mtext("Some independent variable (some units)", side=1, line=2.5) #line specifies how far away
#from the axis to place the text
mtext("Something we measured (some units)", side=2, line=3)
mtext("Very important results!", side=3, cex=2)
```



ggplot2 – Basic Plotting (COLE)

The main tool we're going to teach you is `ggplot2()`, a package built by Hadley Wickam (of RStudio fame), that is one of the easiest ways to make good-looking plots in R. BaseR graphics are great too, but we think `ggplot2` is a bit easier to work with, so we'll use it for our purposes here. `ggplot2()` was originally built as a stand-alone package, but has since been integrated into the `Tidyverse` (which we discussed last week). See our notes from last week on a full discussion of the `Tidyverse`, but in terms of plotting, there are a lot of advantages to using `ggplot2`, which is why we're using it here.

Note: There are a TON of things you can do with `ggplot2`, and with BaseR graphics. Using both methods, you can visualize networks, make pie charts, make beautiful maps, make paneled figures based on multiple variables, and much much more. We're not going to into that, as we could run a whole course on JUST using the different componenets of `ggplot2`. However, luckily, there are a ton of resources online for you to look at if you're trying to make plots or componnets of plots we don't cover here. As usual, we recommend you look broadly online for other tutorials, and even ask a question on *Stack Overflow* if you can't find your answer. I would direct you here (https://ggplot2.tidyverse.org) for a good source of introductory and slightly more advanced materials on using this package.

Geoms and Main Plot Components

Okay, let's start plotting things! To get ourselves set up, we'll first read in some data. For ease, let's just use the data we used last week for our data cleaning exercises:

```
#you'll need to change to your own path here
dir = "C:/Users/coleb/Documents/GitHub/Introduction-to-Programming-for-Biology/data"
setwd(dir)
lice_data = read.csv('week_three_data.csv')
```

So, let's take a quick look and think about what we might want to plot.

```
head(lice_data)
```

```
##    location    region         date slice_conc louse_sex louse_stage temp_0_hrs
## 1 Burdwoods Broughton June 11 2012          0         f          p2          8
## 2 Burdwoods Broughton June 11 2012          0         m          p2          8
## 3 Burdwoods Broughton June 11 2012          0         m          p2          8
## 4 Burdwoods Broughton June 11 2012          0         m          p2          8
## 5 Burdwoods Broughton June 11 2012          0         m          p1          8
## 6 Burdwoods Broughton June 11 2012          0         m          p1          8
##   temp_6_hrs temp_12_hrs temp_18_hrs temp_24_hrs dead moribund live avg_temp
## 1         13           9           8          10    0        0    1      9.6
## 2         13           9           8          10    0        0    1      9.6
## 3         13           9           8          10    0        0    1      9.6
## 4         13           9           8          10    0        0    1      9.6
## 5         13           9           8          10    0        0    1      9.6
## 6         13           9           8          10    0        0    1      9.6
##   dead_and_moribund comments
## 1                 0
## 2                 0
## 3                 0
## 4                 0
## 5                 0
## 6                 0
```

Okay, maybe we want to see how our average temperature changes over time! This is going to require a bit of data prep however. To look at something over time, we need a date, to plot that against. There's a couple ways we could go about this, but in order to get a little more practice doing data manipulation, let's start by separating out the `date` column into `day`, `month` and `year` columns. To do this, we'll use some useful BaseR functions. In the nature of transparency, there are MANY ways we could do this more concisely using packages. However, we think it's useful to see how we do things without packages, as it gives us a better idea of what's going on 'under the hood'. The easiest way to do this is actually to make a new dataframe with the new columns, then bind the old dataframe with the new, so we can keep the `date` column as well as the subset columns.

```
#this is an important first step to let the column be manipulated
lice_data$date = as.character(lice_data$date)

#we're going to make the new columns first
d_m_y = read.table(text = lice_data$date, #specify text is we're reading as a table
            #sep is asking how the characters are separated,
            #and then the column names are listed in a character vector
           sep = ' ', col.names = c('month', 'day', 'year'))

#now bind them together by columns
lice_data = cbind(lice_data, d_m_y)
```

Okay, so our data is ready. But why did we do this? Well, plotting with a continous numeric variable on the x-axis is usually easier than working with factors or characters, especially since if we're plotting along a timescale, that's supposed to be a 'continuous' data vizualization. So we've separated out the day, now let's make it an integer (for easy plotting).

```
lice_data$day = as.integer(lice_data$day)
```

Now, we want to plot our average temperature across time, so let's do that. But first, we need to install and load the `ggplot2` package. **This requires an internet connections, your installation will likely**

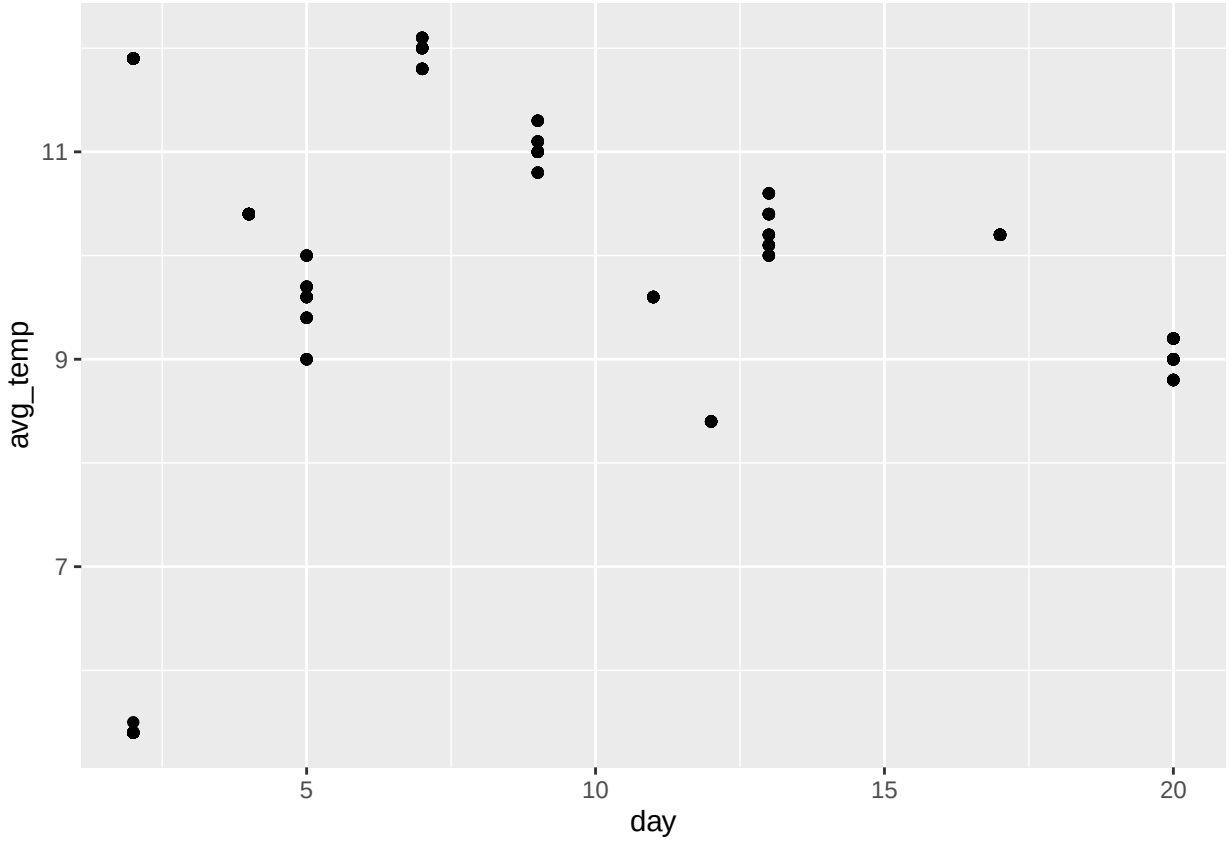
print out a message, that's supposed to happen.

```
#install.packages('ggplot2') #how we install a new package
#- YOU WILL NEED TO RUN THIS IF YOU DON'T HAVE GGPLOT INSTALLED
library(ggplot2) #this is how we call an installed packaged in a script to use it
```

```
## Warning: package 'ggplot2' was built under R version 3.5.3
```

Now we can make our first call to `ggplot2` works somewhat differently than other packages. We use `+` symbols to add arguments together. The first argument is a call to `ggplot()`, with a second argument to call a `geom`. There are many `geoms` you can use for different types of plots. For our purposes, we'll use the `geom_point` to make a scatterplot.

```
ggplot(data = lice_data) + #the first thing you write, it needs is what data to use
  #the only neccessary arguments are the aesthetics (aes()) which take the x and y axis
  geom_point(aes(x = day, y = avg_temp))
```



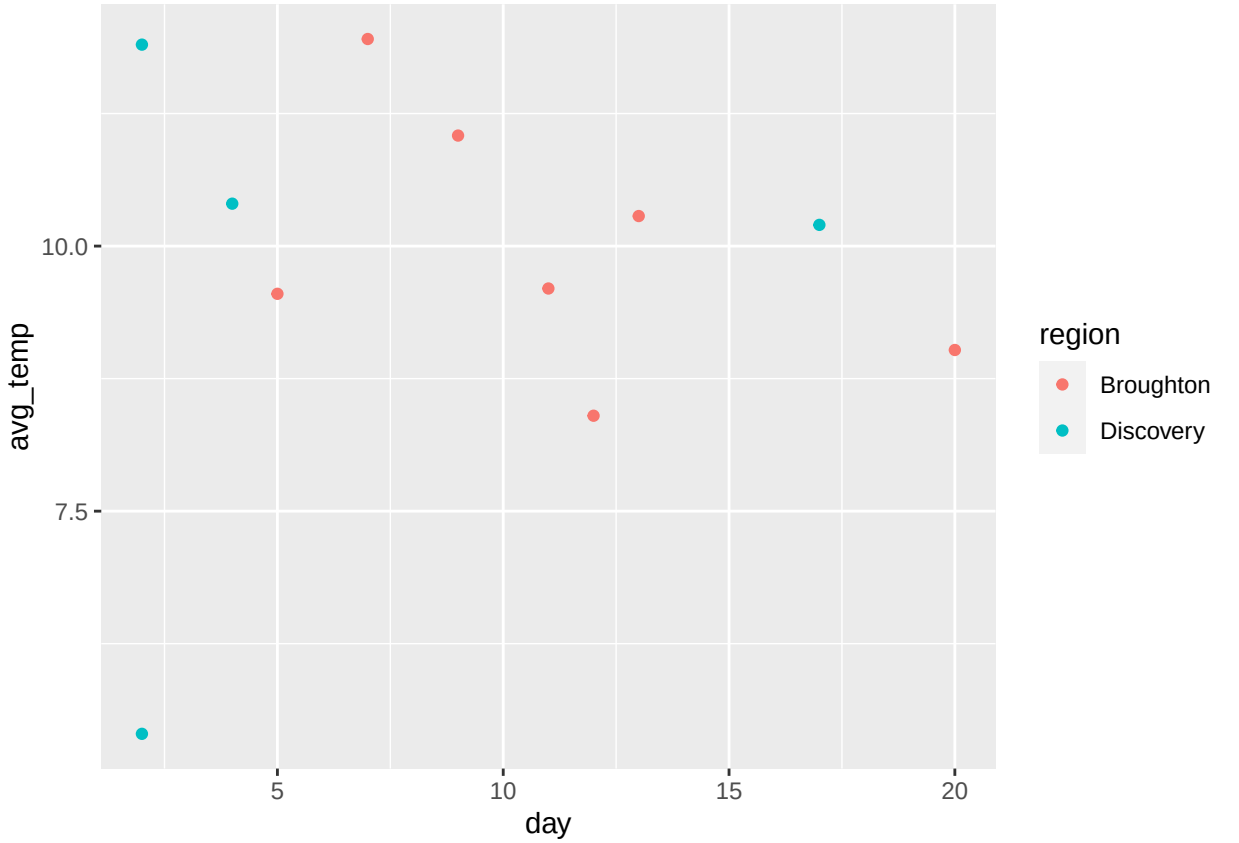
Cool, we've made a plot. It's quite ugly, but that's okay for now. Interesting though, it looks like the data are stacked on each other. Well, that makes sense, we've plotted all of the months and years of our data, so the measurements taken on the same day in each month/year will be stacked on top of each other. So let's do this: Let's create a summarized dataframe that gives us the average temperature as well as the 6th hour temperature for each day in each month in each year. Here, we're going to use `aggregate()` to calculate mean values of `avg_temp`, `temp_0_hrs`, and `temp_6_hours` while grouping by day, month, and year, as well as by sampling region. For a more complete description, check out the video on this topic.

```
plot_data = aggregate(cbind(avg_temp, temp_6_hrs, temp_0_hrs) ~ day +
                        month + year + region, data = lice_data,
                      mean, na.rm = TRUE)
plot_data
```

```
##    day month year    region avg_temp temp_6_hrs temp_0_hrs
## 1   11  June 2012 Broughton  9.60000  13.000000    8.00000
## 2   12  June 2012 Broughton  8.40000   8.000000   11.00000
## 3   20  June 2012 Broughton  9.02000   8.000000   13.83333
## 4    5  June 2015 Broughton  9.55000   7.500000   13.00000
## 5    7  June 2015 Broughton 11.95294   7.676471   17.00000
## 6    9  June 2015 Broughton 11.04242   9.393939   14.00000
## 7   13  June 2015 Broughton 10.28333   8.000000   13.00000
## 8    2  July 2012 Discovery 11.90000  11.500000   11.50000
## 9    2  June 2012 Discovery  5.40000   5.000000    5.00000
## 10   4  June 2012 Discovery 10.40000  10.000000   10.00000
## 11  17  June 2012 Discovery 10.20000  10.000000   10.00000
```

So now let's try to make this plot again and maybe make it look a little bit prettier. Maybe we can make the points coloured by regions. I'm going to use my new `plot_data` object now.

```
ggplot(data = plot_data) +
  geom_point(aes(x = day, y = avg_temp, colour = region))
```



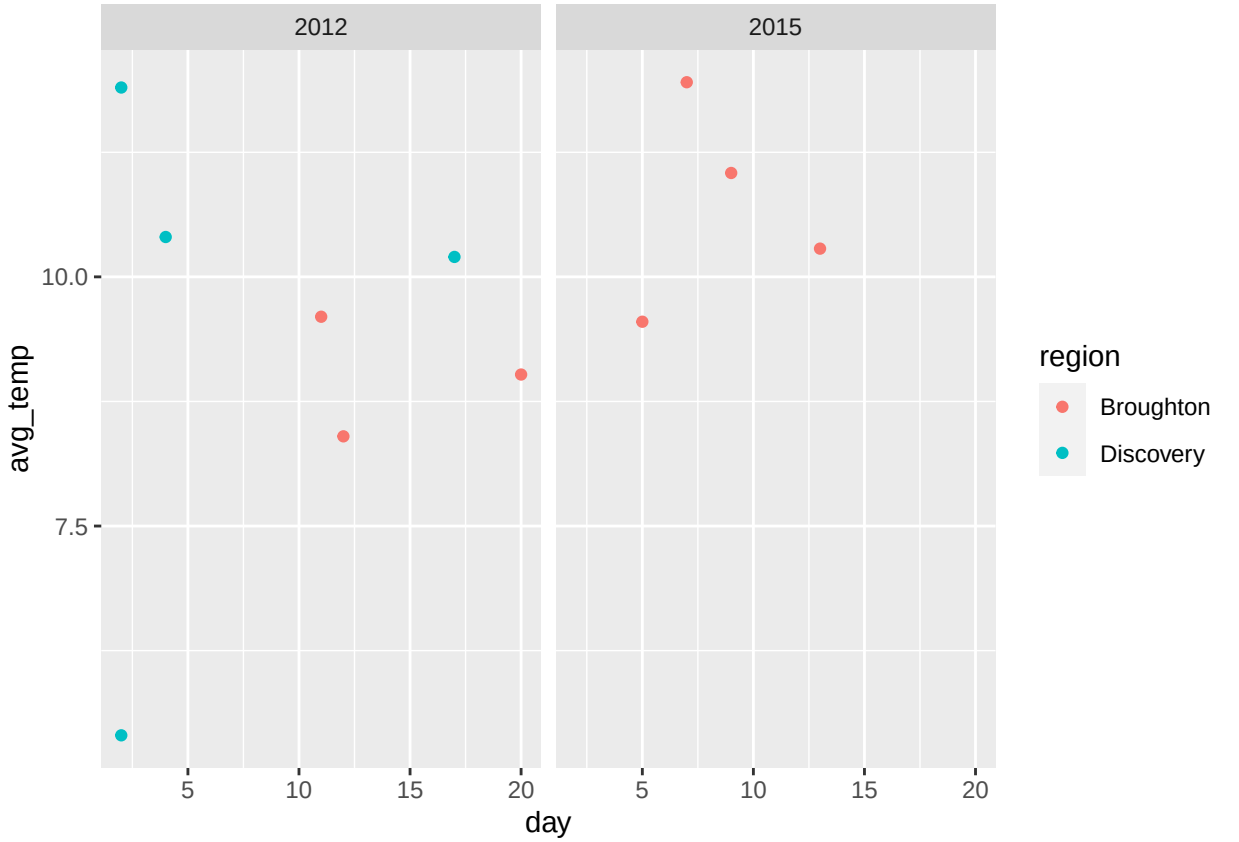

```
#by specifying colour in the aes() call it will assign it based on
#levels of 'region' - note that region has to be a factor for this to work properly in our case
```

Fun! This is a much more readable graph. What if we plotted the years in separate panels? Might be easier to look at?

Faceting

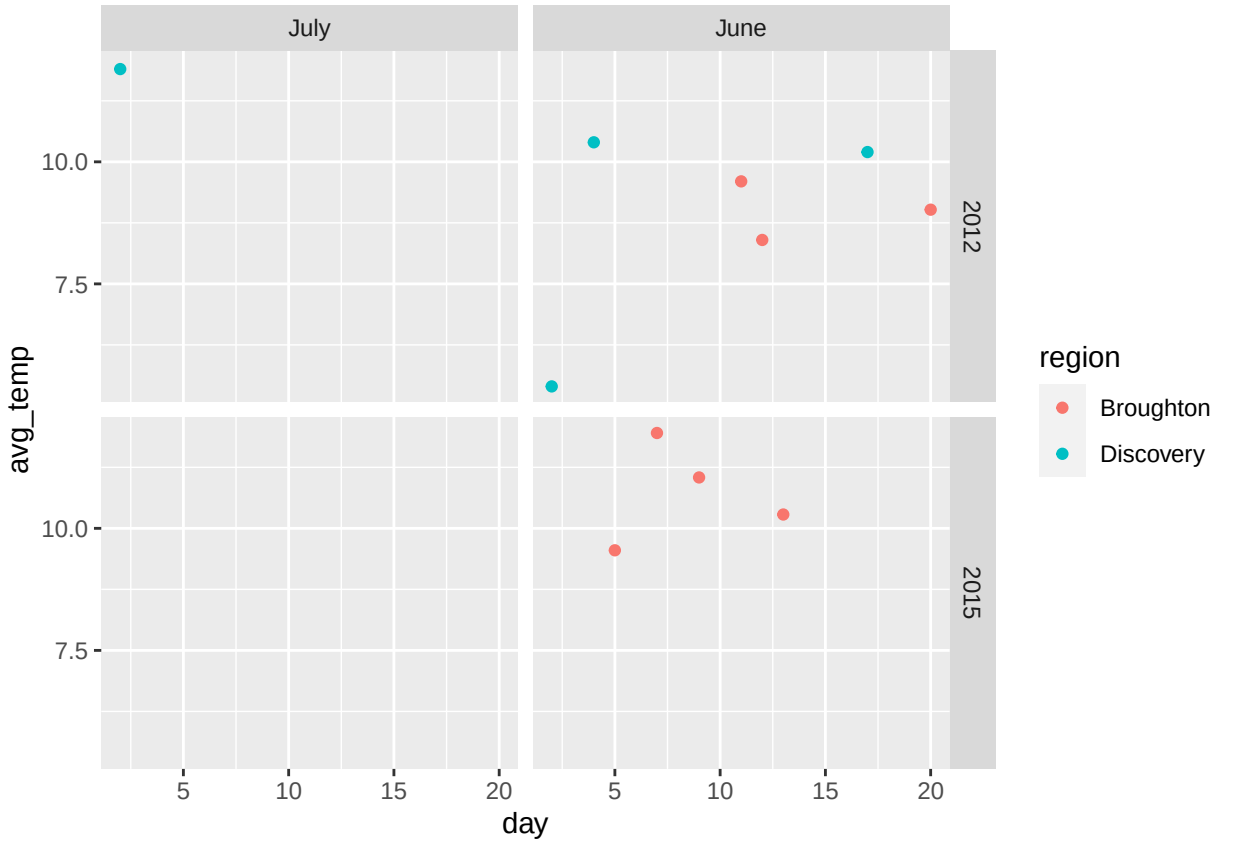
One thing you might want to do is split up your plot by some categorical variable. Here, we can do it by year, but we could have just as easily done it by region or some other variable.

```
ggplot(data = plot_data) +
  geom_point(aes(x = day, y = avg_temp, colour = region)) +
  facet_wrap(~year)
```



Cool, that did what I want, and I now see interestingly that there are no measurements in the Discovery Islands in 2015 here. But there's still multiple months here, we could separate those as well, similarly to above.

```
ggplot(data = plot_data) +
  geom_point(aes(x = day, y = avg_temp, colour = region)) +
  facet_grid(year~month)
```



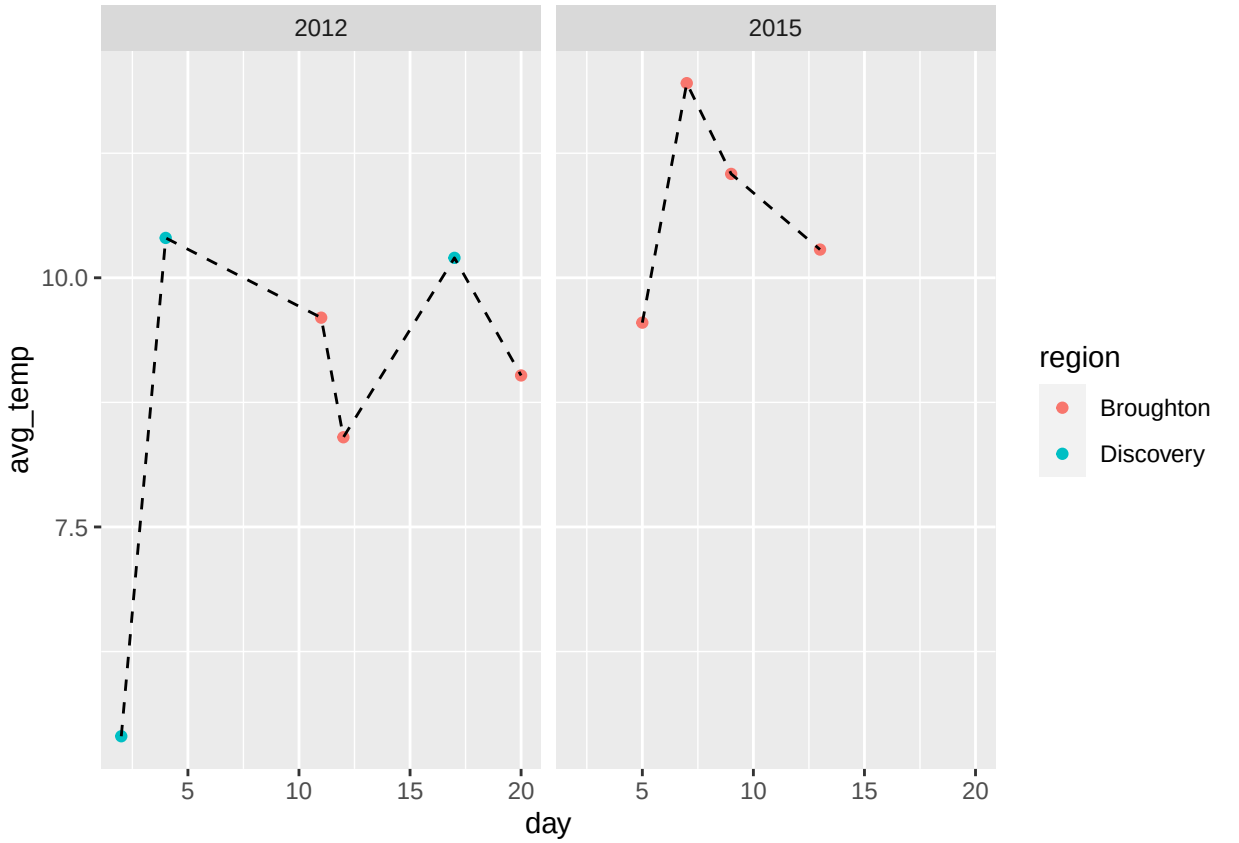
So this is a bit less dense since there's no mesaurements at all in the July/2015 panel, but that's okay for our purposes. For easier viewing, let's only plot June measurements. We can do this by filtering our data which we covered last week.

```
#we learned this method of subsetting last week
plot_data = plot_data[which(plot_data$month == 'June'),]
plot_data
```

```
##    day month year    region avg_temp temp_6_hrs temp_0_hrs
## 1   11  June 2012 Broughton  9.60000  13.000000    8.00000
## 2   12  June 2012 Broughton  8.40000   8.000000   11.00000
## 3   20  June 2012 Broughton  9.02000   8.000000   13.83333
## 4    5  June 2015 Broughton  9.55000   7.500000   13.00000
## 5    7  June 2015 Broughton 11.95294   7.676471   17.00000
## 6    9  June 2015 Broughton 11.04242   9.393939   14.00000
## 7   13  June 2015 Broughton 10.28333   8.000000   13.00000
## 9    2  June 2012 Discovery  5.40000   5.000000    5.00000
## 10   4  June 2012 Discovery 10.40000  10.000000   10.00000
## 11  17  June 2012 Discovery 10.20000  10.000000   10.00000
```

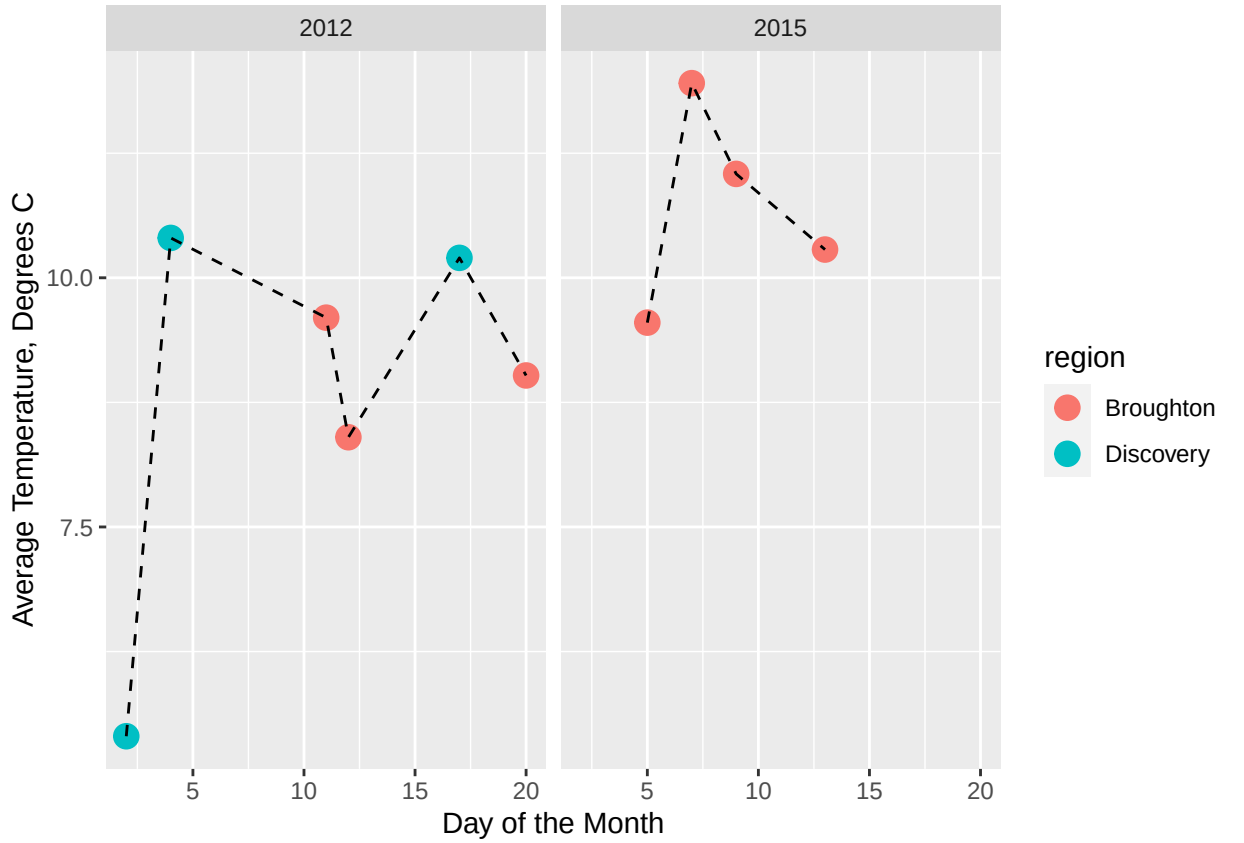
Let's just plot June, and let's also connect our points with a line

```
ggplot(data = plot_data) +
  geom_point(aes(x = day, y = avg_temp, colour = region)) +
  geom_line(aes(x = day, y = avg_temp), linetype = 'dashed') +
  facet_wrap(~year)
```



Great! However, the points are bit small to see. Let's make them bigger. Also, let's make our axes labels look a bit better.

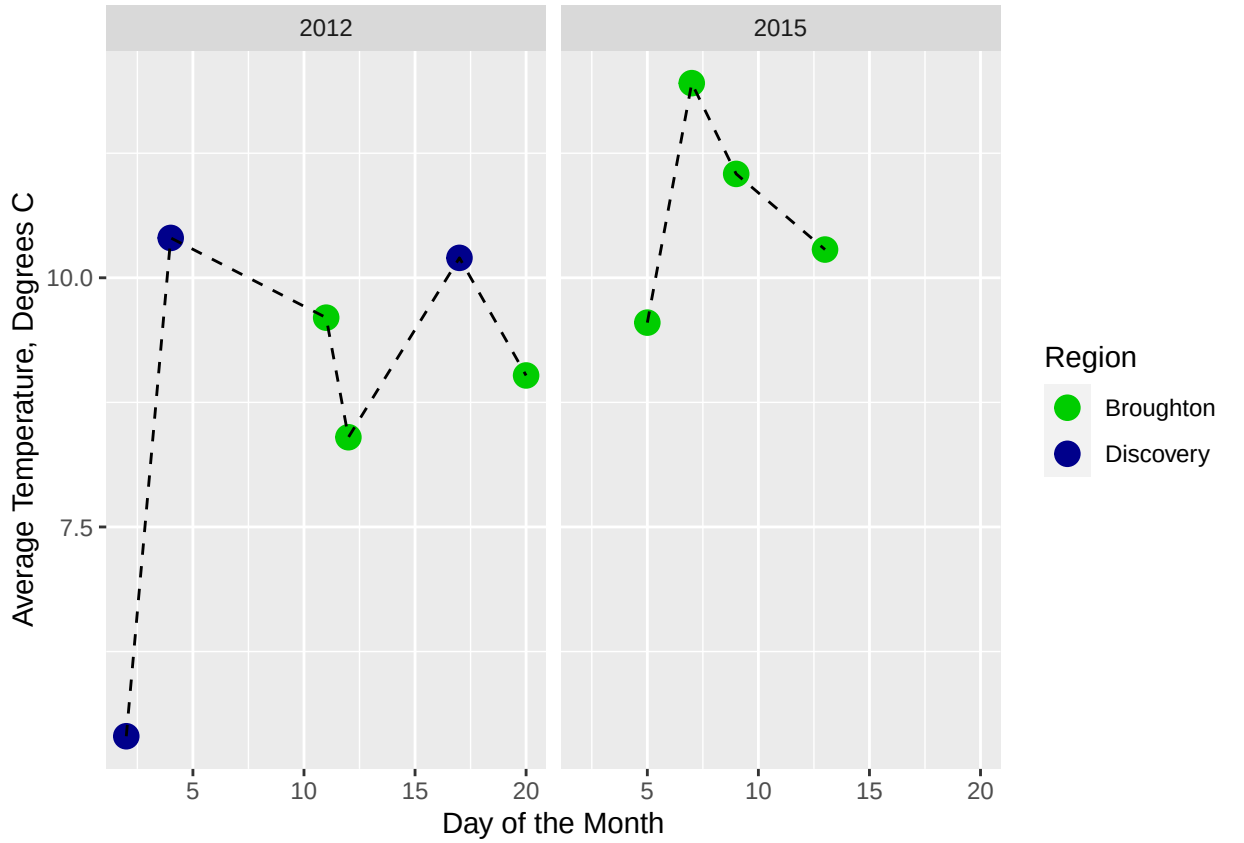
```
ggplot(data = plot_data) +
  geom_point(aes(x = day, y = avg_temp, colour = region), size = 4) +
  geom_line(aes(x = day, y = avg_temp), linetype = 'dashed') +
  facet_wrap(~year) +
  labs(x = 'Day of the Month',
       y = 'Average Temperature, Degrees C') #this labs() argument is for labels
```



Notice I've included the units on the y-axis, which is important to do. What else? Well, I don't particularly like the general look of the colours that R has picked for my points. Nor do I like that the legend title isn't capitalized. Let's change those two things.

```
leg_title = 'Region' #here I define what I want my name of the legend to mbe

ggplot(data = plot_data) +
  geom_point(aes(x = day, y = avg_temp, colour = region), size = 4) +
  geom_line(aes(x = day, y = avg_temp), linetype = 'dashed') +
  facet_wrap(~year) +
  labs(x = 'Day of the Month', y = 'Average Temperature, Degrees C') +
  scale_colour_manual(leg_title,
                      values = c('green3', 'blue4'))
```



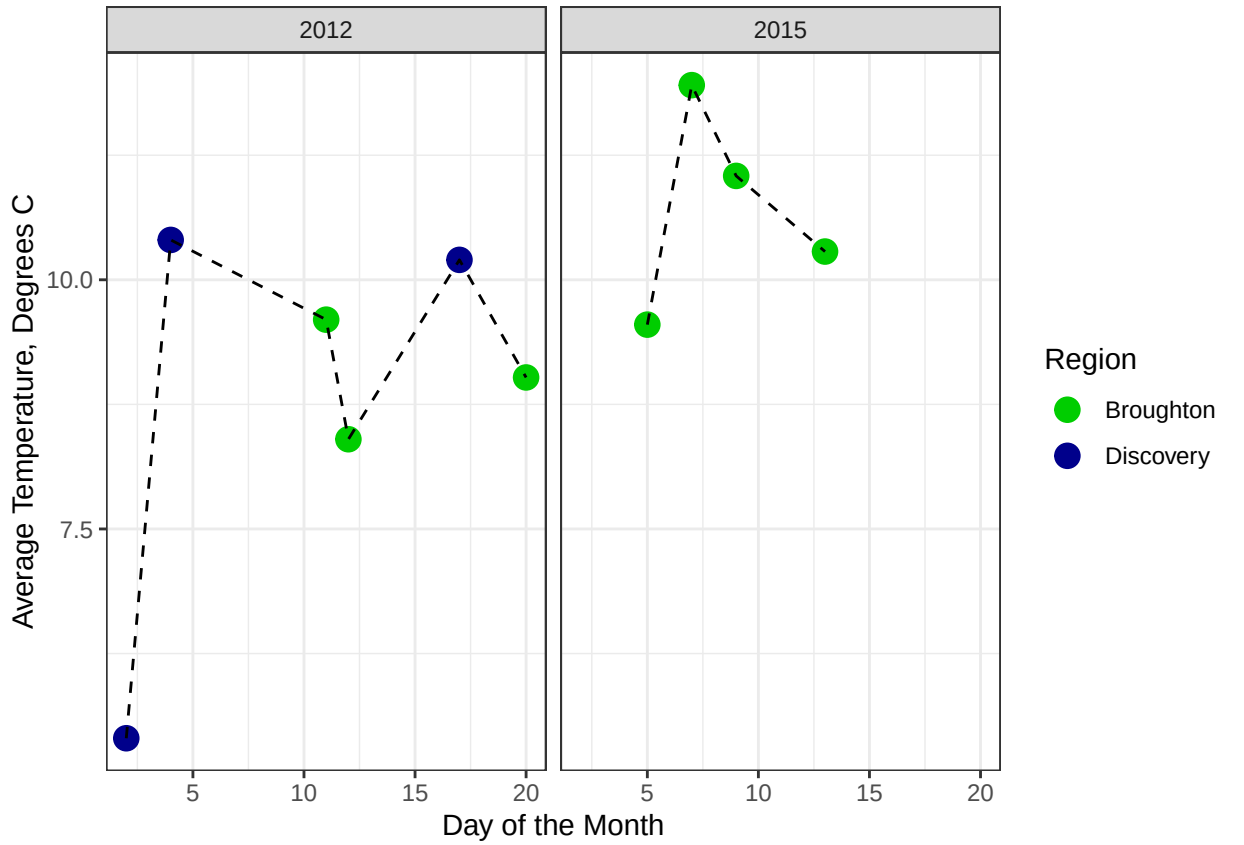

```
#the first argument is name of the legend,
#then the values of the colours I want my points to be
```

Themes

Much better. But I don't really like this grey background. Not to fear, we can change that too, with built in `ggplot2` themes. Now, if you're feeling fancy, you can make your own theme as a function. There are good tutorials online, and if you will be making a lot of plots in R, I highly recommend taking the time to make a theme function that you can add to your own plots. It can be a bit time-consuming, but since I made my first theme function, I've just copied and pasted it into different scripts since then and altered it depending on each individual use case. You'll find as you code more and more, that much of your coding is simply copying and pasting from old code you wrote previously.

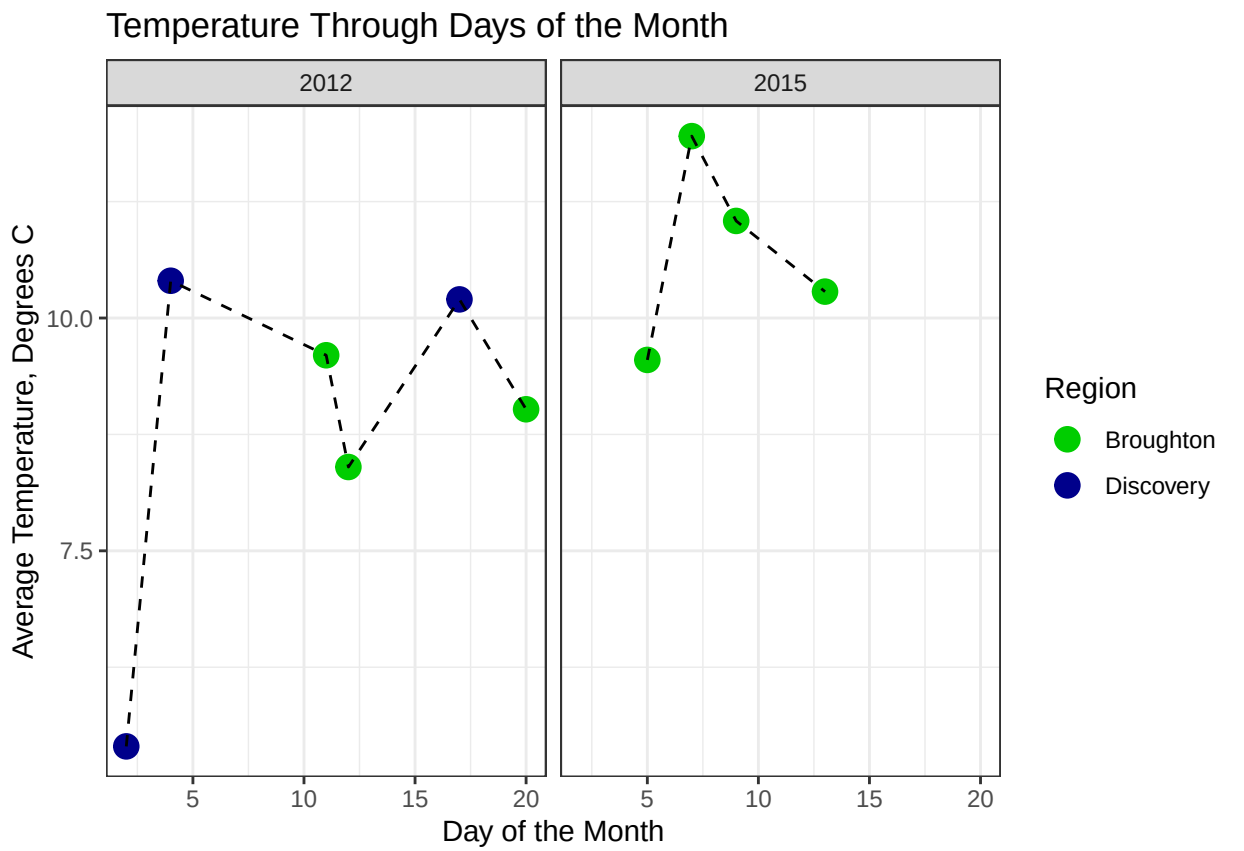
But I digress, let's use a built in theme.

```
#since we specified 'leg_title' above, we don't need to do it again
ggplot(data = plot_data) +
  geom_point(aes(x = day, y = avg_temp, colour = region), size = 4) +
  geom_line(aes(x = day, y = avg_temp), linetype = 'dashed') +
  facet_wrap(~year) +
  labs(x = 'Day of the Month', y = 'Average Temperature, Degrees C') +
  scale_colour_manual(leg_title, values = c('green3', 'blue4')) +
  theme_bw()
```



Okay, this looks pretty good! I might just add a title.

```
ggplot(data = plot_data) +
  geom_point(aes(x = day, y = avg_temp, colour = region), size = 4) +
  geom_line(aes(x = day, y = avg_temp), linetype = 'dashed') +
  facet_wrap(~year) +
  labs(x = 'Day of the Month', y = 'Average Temperature, Degrees C',
       title = 'Temperature Through Days of the Month') +
  scale_colour_manual(leg_title, values = c('green3', 'blue4')) +
  theme_bw()
```



Great! So, this is pretty much a complete plot. There are things we could keep changing if we wanted to, but for a basic visualization, this looks pretty good. I'm happy with it.

Annotations

The last thing we'll mention is how to make some small annotations. Often we want to superimpose additional compnents onto graphs such as other words or arrows. What if, for example, we wanted to point out that the first value in 2012 is really low compared to all the others? We might want to point at it with an arrow. Let's do that using an `annotation`.

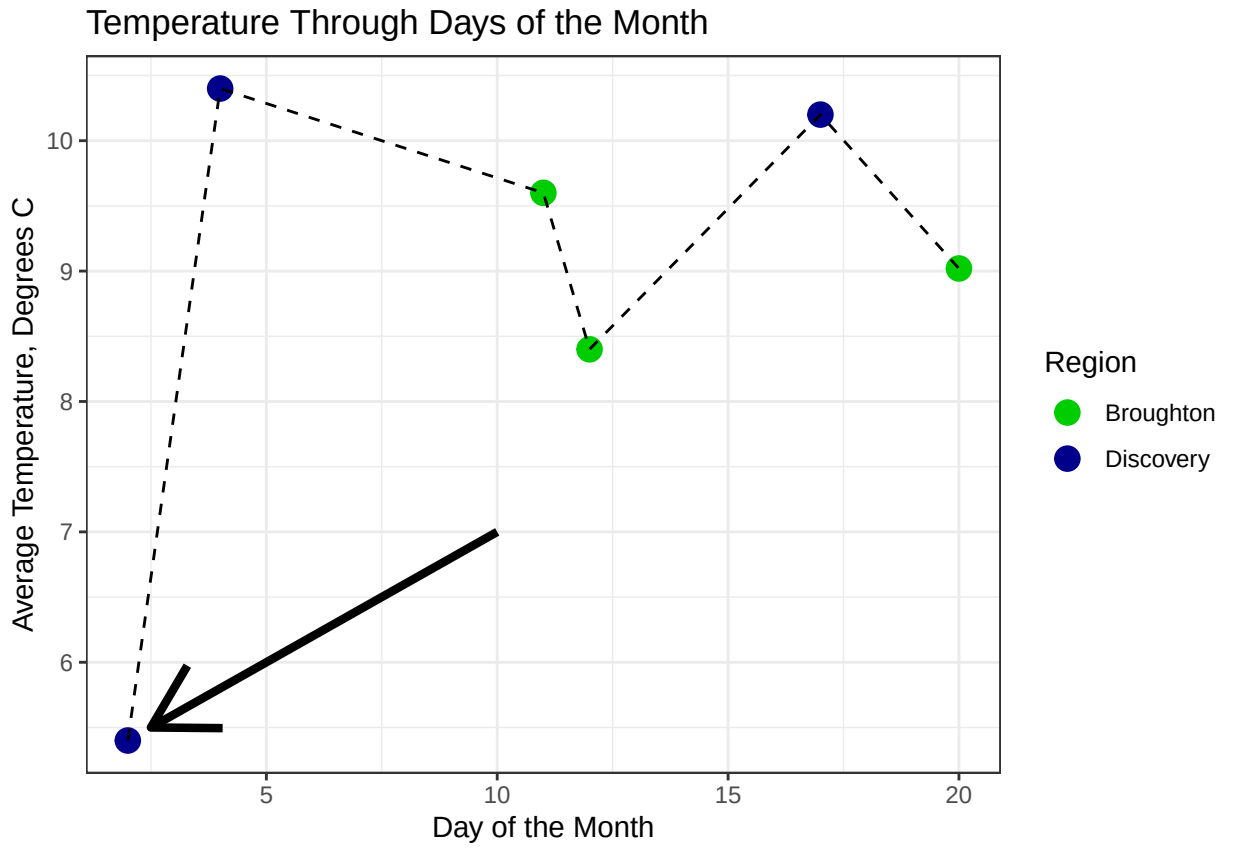
Annotating faceted plots is a bit more difficult, so simply in the interests of time, I'll only plot the 2012 data on one plot here, so we don't have to deal with the faceting. However, for those of you who are curious, a good way to plot a line segment or some other annotation on a single facet of a multi-facet plot, here's a link (https://stackoverflow.com/questions/14508277/adding-annotation-segment-arrow-in-only-certain-facet-ggplot).

So, I'll first subset my data to just 2012:

```
#we learned this method of subsetting last week
plot_data = plot_data[which(plot_data$year == '2012'),]
plot_data
```

```
##    day month year    region avg_temp temp_6_hrs temp_0_hrs
## 1   11  June 2012 Broughton     9.60         13    8.00000
## 2   12  June 2012 Broughton     8.40          8   11.00000
## 3   20  June 2012 Broughton     9.02          8   13.83333
## 9    2  June 2012 Discovery     5.40          5    5.00000
## 10   4  June 2012 Discovery    10.40         10   10.00000
## 11  17  June 2012 Discovery    10.20         10   10.00000
```

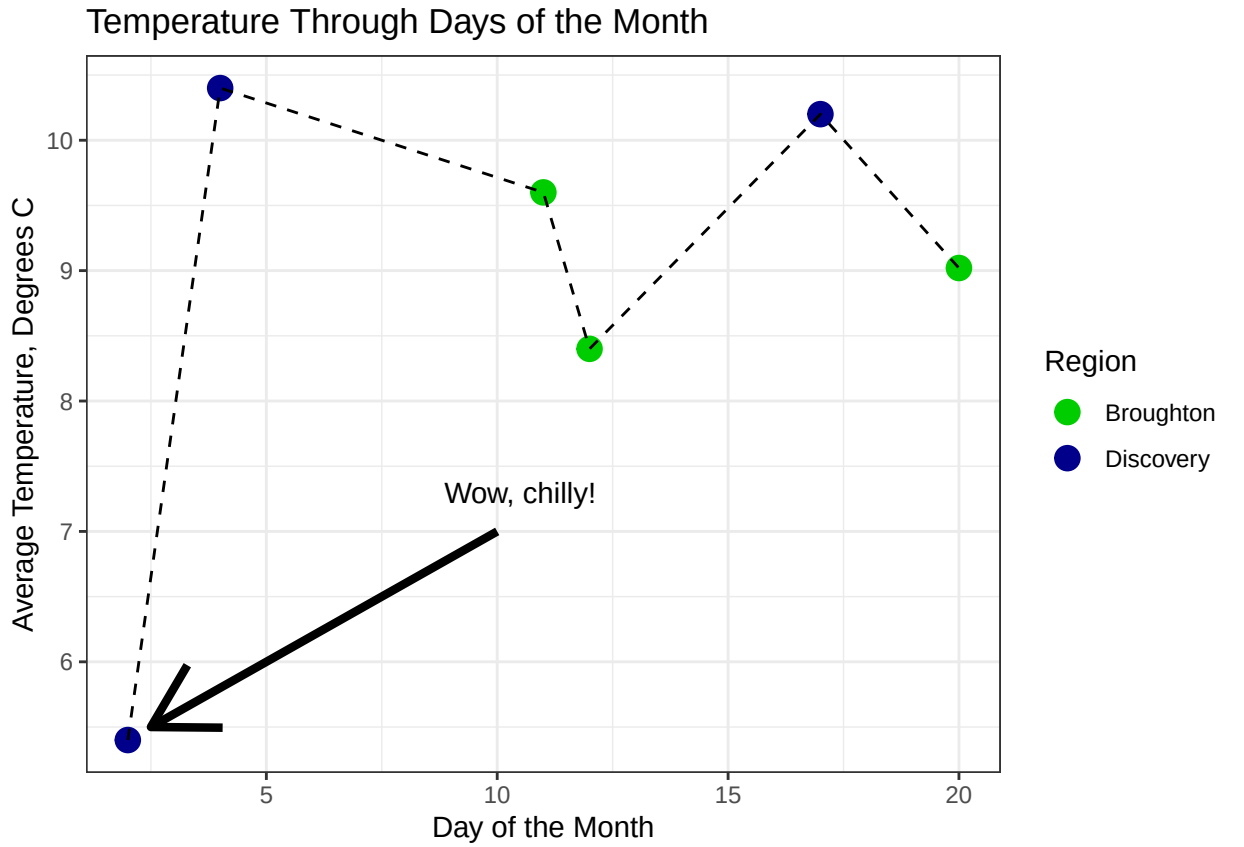
```
#note I've taken out the line for facet_wrap() here since we no longer need it
ggplot(data = plot_data) +
  geom_point(aes(x = day, y = avg_temp, colour = region), size = 4) +
  geom_line(aes(x = day, y = avg_temp), linetype = 'dashed') +
  labs(x = 'Day of the Month', y = 'Average Temperature, Degrees C',
       title = 'Temperature Through Days of the Month') +
  scale_colour_manual(leg_title, values = c('green3', 'blue4')) +
  theme_bw() +
  annotate('segment', #type of annotation
           x=10, y=7.0, xend=2.5, yend=5.5, #the beginning and end of the arrow
           arrow=arrow(length = unit(0.1, "npc")), #arrow ending and size of arrow tails
           size = 1.5) #thickness ofthe line
```



Okay cool, we have our arrow. Now we might want to add some text. As this is just an example, there's nothing specific we need to add, but let's add something anyways. As a treat.

```
ggplot(data = plot_data) +
  geom_point(aes(x = day, y = avg_temp, colour = region), size = 4) +
  geom_line(aes(x = day, y = avg_temp), linetype = 'dashed') +
  labs(x = 'Day of the Month', y = 'Average Temperature, Degrees C',
       title = 'Temperature Through Days of the Month') +
  scale_colour_manual(leg_title, values = c('green3', 'blue4')) +
  theme_bw() +
  annotate('segment',x=10, y=7.0, xend=2.5, yend=5.5,
           arrow=arrow(length = unit(0.1, "npc")),
           size = 1.5) +
  annotate('text', #type of annotation
```

```
          x = 10.5, y = 7.3, #position of the centre of the annotation
          label = 'Wow, chilly!', #what I actually want to say
          size = 4) #size of text
```



And there we go! So we've done it. We've made a plot, edited it substantially, and even threw an arrow and a label on top! Fun!

As a recap, there are LOTS of different ways to plot things in R, both with `ggplot2` and with BaseR. There are often even multiple ways to achieve the same result within each of these. There is not a 'right' or 'wrong' way, and as you continue on you'll find ways that you like to make plots. Don't be surprised when you find yourself doing a LOT of googling (or searching on your prefered search engine, we are not endorsed by Google) at first to figure out how to do things with plots. There are a lot of resources out there, and we encourage you to search widely in your quest for beautiful, informative plots.