A cheat sheet to ‘tidy’ data processing for ECOL 4000/6000

Katie Schroeder ([k.schroeder@uga.edu](mailto:k.schroeder@uga.edu))

2023-10-13

#### Topic 1: Basics - setting your working directory, install and calling packages

**Setting your working directory:**  
This step is really important and tells R where you’re grabbing files from on your computer. There are a couple options for setting your working directory:

* set it manually using the setwd() command
  + Ex: setwd(“~/User/Dawg1/Documents”)
* set it to the source file location
  + If your script is saved in the same location as the files you will be loading, you can go to the session menu, select “Set working directory” and then select “To source file location”
* set it to a folder you browse for
  + You can select your working directory location by going to the session menu, selecting “Set working directory” and then selecting “Choose directory”

**Installing a package:**  
We’ve used a few packages in this class so far (one example is tidyverse). As you work more with R, you may find more and more useful packages. If you don’t have a package on your computer, you’ll have to install it using install.packages(). Make sure you put the name of the package in quotes when you’re using install.packages()

**Calling a package from your library:**  
If you have a package installed, it’s in your library, but will not be automatically loaded when you start a new R session. To use the functions in that package, you’ll need to call it from your library.  
For example, if we want to use ggplot(), we’ll need to call tidyverse using the library() command.  
Example: library(tidyverse)

Side note: You could also use library(ggplot2) if you were just using ggplot functions. tidyverse is a bit of a super package that contains a lot of other packages (including ggplot2).

#### Topic 2: Reading in data and viewing it

Often times we’ll read things in as .csv files using read.csv

For the bears problem set, we used the following code:

bears<- read.csv("PS1\_Data\_Bears.csv")

**Viewing the data frame:**

To view the bears data frame, there are a couple options:

* You can type View(bears) into the console or into your script
* You can click on bears in the global environment

Some options for looking at your data:

* Overview of the entire data frame:
  + str() gives an overview of the length, column names, and the type of data (number, character, integer, etc.) in each column
  + head() gives the first six rows of your data frame
  + summary() gives a summary of each column in your data frame (quantiles, minimums, maximums)
* For specific columns - remember to use the $ operator to specify the column!
  + sum() the total in a column
  + min() mean() median() and max() for those summary stats

**Subsetting the data frame:**

The brackets method from the bears problem set:

#getting a single column, females  
bears[,2] #second column  
bears$females #equivalent   
  
#making two data frames for data before and after 1976  
bears1 <- bears[bears$year<1977,]  
bears2 <- bears[bears$year>1975,]

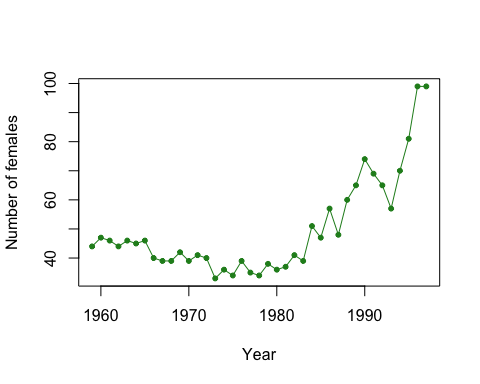
Tidyverse options:

#load tidyverse  
library(tidyverse)  
  
#getting a single column  
bears %>% select(females)  
  
#subsetting before and after 1976  
bears1 <- bears %>% filter(year<1977)  
bears2 <- bears %>% filter(year>1975)

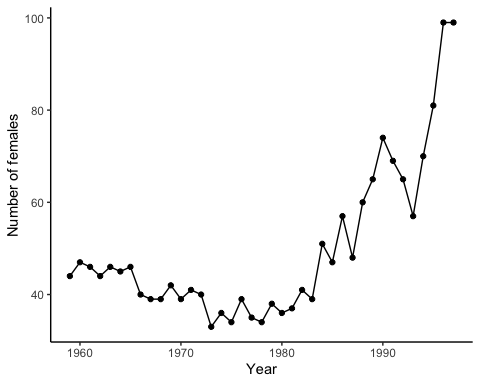
#### Topic 3: Plotting

Example from the bears problem set:

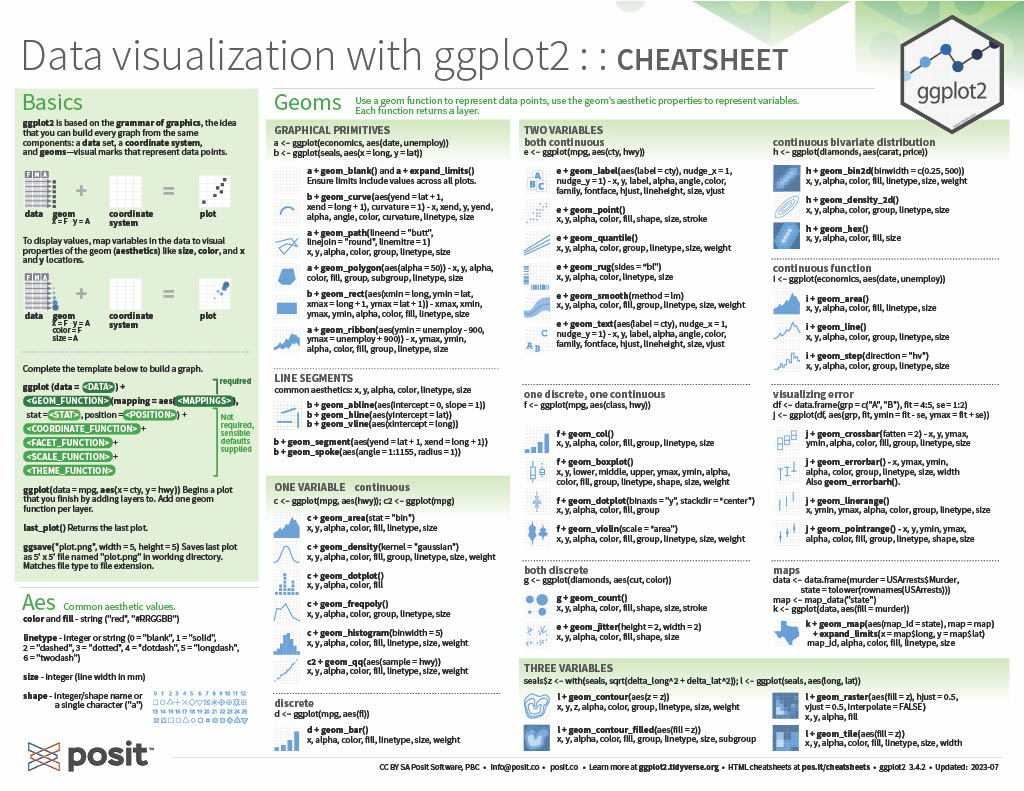
plot(bears$year,  
 bears$females,  
 xlab="Year",  
 ylab="Number of females",   
 col="forestgreen",   
 type="o",   
 pch=20)

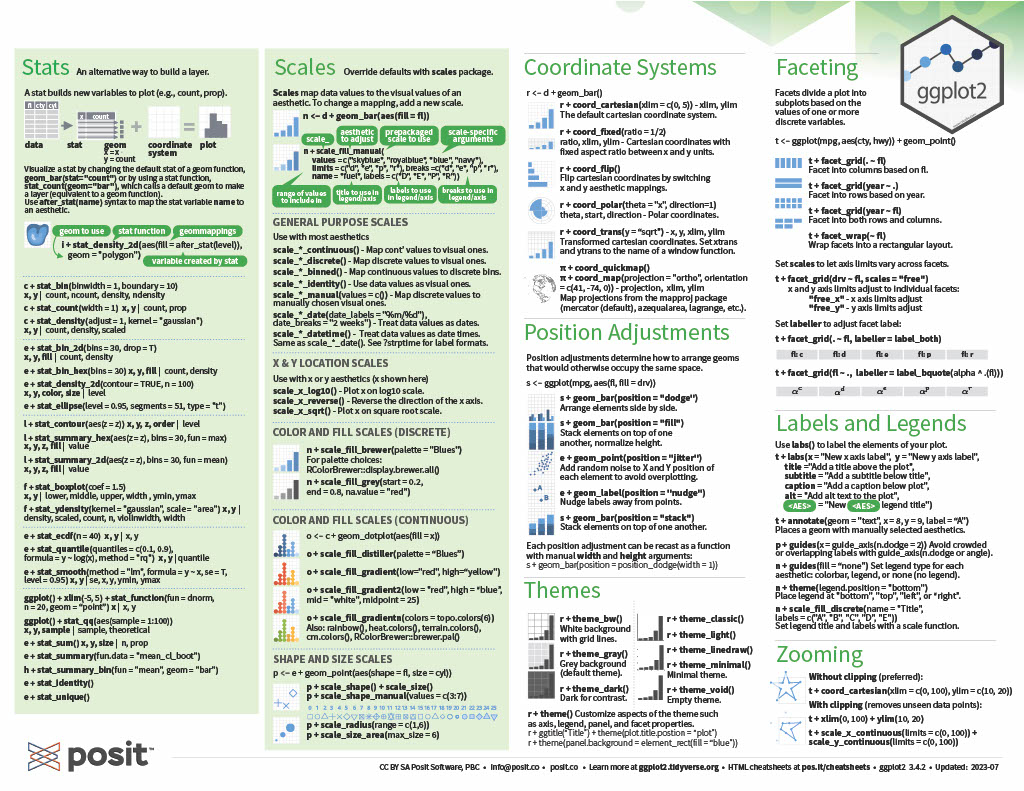


# a tidyverse option:   
ggplot(bears,aes(x=year,y=females)) +  
 geom\_point() +  
 geom\_line() +  
 xlab("Year") +  
 ylab("Number of females") +  
 theme\_classic()



**Commands for more advanced plotting in ggplot**

ggplot cheat sheet page 1:  


ggplot cheat sheet page 2:  


#### Topic 4: Calculating different response variables

In problem set 2, we calculated lambda for the terns on all of the islands using a for loop:

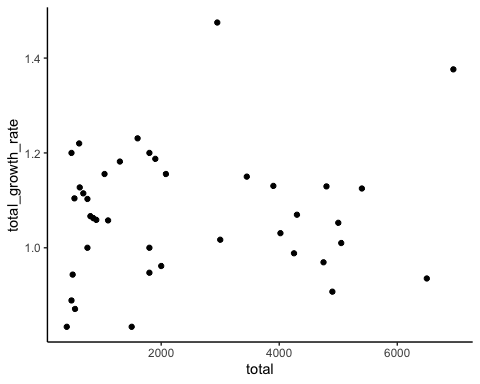
terns <- read.csv("PS2\_data\_terns.csv")  
  
#calculate the total terns by adding together the terns from each island  
terns$total <- terns$bird + terns$ram + terns$pikense  
  
#create an empty column for the growth rate to populate  
terns$total\_growth\_rate <- NA  
  
for(i in 2:length(terns$year)){ # initiate the loop with "for"  
 terns$total\_growth\_rate[i] <- terns$total[i]/terns$total[i-1]   
 # calculate the growth rate as the ratio between population size now and population size one year ago  
 } # close the loop

We could have also done this calculation using the lag() command in the package dplyr. dplyr is another package in tidyverse. lag() can be used to create column that is one row behind the column we input.

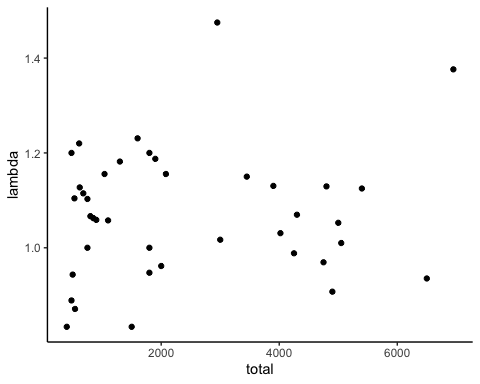
terns$total\_N0 <- lag(terns$total) #lag function makes a new column with total column shifted down a row (lag=1)  
  
#calculate lambda by dividing total at N+1 by total at N  
terns$lambda <- terns$total / terns$total\_N0

If we compare the two lambas we calculated, total\_growth rate using a loop and lambda using lag, we can see they lead to the same result:

ggplot(terns,aes(x=total,y=total\_growth\_rate)) +  
 geom\_point() + theme\_classic()

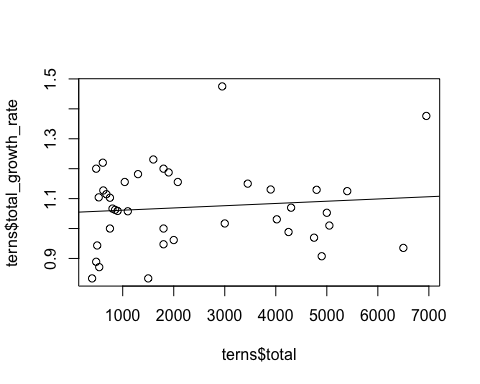


ggplot(terns,aes(x=total,y=lambda)) +  
 geom\_point() +theme\_classic()



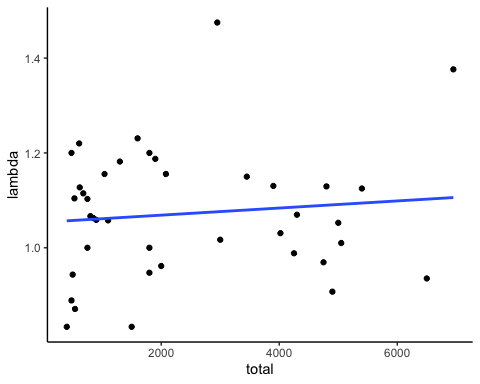
In the tern example, we also fit a line to these data. In base r, we used abline() and lm() (linear model). We can do the same in ggplot using geom\_smooth()

#base r  
plot(terns$total, terns$total\_growth\_rate)  
abline(lm(total\_growth\_rate~total, data=terns))



#tidyverse option  
ggplot(terns,aes(x=total,y=lambda)) +  
 geom\_point() +  
 geom\_smooth(method='lm',se=F) + #se=F here means it will just show the line and not an estimate of the error associated with the fit  
 theme\_classic()

## `geom\_smooth()` using formula = 'y ~ x'



#### Topic 5: Working with the SRS zooplankton data

We often want to know how many of a certain thing we have. For one of the problem sets, we calculated the number of unique taxa using length() and unique()

zoop <- read.csv("SRS\_zoop.csv")  
#number of different taxa   
length(unique(zoop$taxa))

## [1] 133

#number of bays  
length(unique(zoop$bay))

## [1] 14

From this, we see that there are 133 different taxa found in the 14 different bays.

We may want to ask about unique taxa in each bay. We could subset and create data frames for each of the bays like we did for the bears data or we could create a loop, but we can also use tidyverse syntax and the group\_by() and summarize() commands. Within summarize, we’re using n\_distinct() which counts the number of distinct observations in the specified column.

zoop %>% group\_by(bay) %>%  
 summarize(unique\_taxa = n\_distinct(taxa))

## # A tibble: 14 × 2  
## bay unique\_taxa  
## <int> <int>  
## 1 3 72  
## 2 4 54  
## 3 7 59  
## 4 9 59  
## 5 11 70  
## 6 25 52  
## 7 26 71  
## 8 40 71  
## 9 41 62  
## 10 44 54  
## 11 66 60  
## 12 78 93  
## 13 79 35  
## 14 80 37

**Other loop and tidyverse equivalent processes from working with these data in the problem sets:**

In the problem set, we calculated abundance of each of the taxa using this code:

taxa <- unique(zoop$taxa) # first step is to create a list of all taxa  
total.abundance <- data.frame(taxa=taxa) # then create an empty dataframe  
# that just includes the names of the taxa. then, we'll fill in a second  
# column of the dataframe with our loop:  
for(i in 1:length(taxa)){  
 zoop.subset <- zoop[zoop$taxa==taxa[i],] # subset the data to only include  
 # rows where taxa i is observed.   
 total.abundance$abundance[i] <- sum(zoop.subset$abund, na.rm=T)  
 # sum up the total abundance of that taxa, after removing NAs  
 # and input it as the i'ith observation in the summary dataframe  
}  
  
#we can use head to see the first six rows of our sorted abundance data frame  
head(total.abundance[order(total.abundance$abundance),])

## taxa abundance  
## 15 Alona sp. 1  
## 41 Curculionidae 1  
## 52 Diptera larva 1  
## 54 Dunhevedia crassa 1  
## 85 Lepidoptera 1  
## 101 Nepidae 1

#adding a minus sign in front of the column we want to sort using order sorts it the opposite direction (descending)  
head(total.abundance[order(-total.abundance$abundance),])

## taxa abundance  
## 42 Cyclopoida 113593  
## 21 Bosmina tubicen 59437  
## 107 Ostracoda 37760  
## 51 Diaphanosoma brachyurum 35520  
## 22 Calanoida juv. 31321  
## 43 Daphnia laevis 19662

There’s also an option to do the same thing in tidyverse. In this example, we group the data by taxa and then summarize each of those groups by summing abundance. We can then compare the first and last six values we get from this method.

total\_abundance<-zoop %>% group\_by(taxa) %>%  
 summarize(abundance=sum(abund,na.rm=T))  
  
# to sort using pipes, we use arrange and then head to get the first 6 rows of that arrange data  
total\_abundance %>% arrange(abundance) %>% head()

## # A tibble: 6 × 2  
## taxa abundance  
## <chr> <int>  
## 1 Alona sp. 1  
## 2 Curculionidae 1  
## 3 Diptera larva 1  
## 4 Dunhevedia crassa 1  
## 5 Lepidoptera 1  
## 6 Nepidae 1

# to sort so abundance is descending, we use arrange with desc (for descending)  
total\_abundance %>% arrange(desc(abundance)) %>% head()

## # A tibble: 6 × 2  
## taxa abundance  
## <chr> <int>  
## 1 Cyclopoida 113593  
## 2 Bosmina tubicen 59437  
## 3 Ostracoda 37760  
## 4 Diaphanosoma brachyurum 35520  
## 5 Calanoida juv. 31321  
## 6 Daphnia laevis 19662

We also calculated occupancy of each of the bays for *Bosmina tubicen*. We did this with a for loop:

bays <- unique(zoop$bay) # first we need to create a list of the bays  
bay.occupancy <- data.frame(bay=bays) # next we create an empty summary   
# dataframe  
  
#in this loop, we loop through each of the bays  
for(i in 1:length(bays)){  
 zoop.subset <- zoop[zoop$bay==bays[i],] # subset the dataset so we are  
 # only looking at the i'ith bay  
 bos.subset <- zoop.subset[zoop.subset$taxa=="Bosmina tubicen",]  
 # subset data from the i'ith bay to only include observations of bosmina  
 bay.occupancy$bos[i] <- ifelse(sum(bos.subset$abund, na.rm=T) > 0, 1,0)  
 # use the ifelse statement to say, if abundance is greater than 0, input  
 # a 1 into the summary spreadsheet. otherwise (else) input a 0.   
}  
  
# remember we define occupancy in metapopulation theory as the fraction of   
# sites that are occupied. so we need to divide the total number of sites  
# where bosmina was present by the number of bays:  
sum(bay.occupancy$bos)/length(bays)

## [1] 0.9285714

We can do this using tidyverse too.

bay\_occupancy <- zoop %>% group\_by(bay) %>% #we group by bay  
 mutate(bos = str\_detect(taxa, 'Bosmina tubicen')) %>% #add a column that is true or false depending on if bosmina is in the taxa list  
 summarize(bos\_present = ifelse(any(bos==TRUE),1,0)) #and then summarize using ifelse to get a list of the bays with 1 or 0 based on if the column we added has any bosmina tubicen observations in it  
  
#we can calculate occupancy just like we did with the base r method  
sum(bay\_occupancy$bos\_present/length(bays))

## [1] 0.9285714

#### Topic 6: Tidy versions of problem set 7:

In problem set 7, we calculated various diversity indices with the zoop dataset:

#look at just bay 80 by creating a subsetted data frame called bay80  
bay80 <- zoop[zoop$bay==80,]   
  
#now subset that data frame to one sampling date  
bay80s41410 <- bay80[bay80$sample\_date=="4/14/10",]  
  
# for richness, we just need to know the number of unique species.   
length(unique(bay80s41410$taxa))

## [1] 15

# for simpson's & shannon's diversity, we first need to convert all species abundances  
# into frequencies. the first step is to figure out the denominator; i.e., the total  
# number of 'things' counted. for that, we'll use the sum function.  
total.counted <- sum(bay80s41410$abund)  
  
# next, we'll create a new column in the subset dataframe where we calculate the   
# frequencies for each species  
bay80s41410$freq <- bay80s41410$abund / total.counted  
# now, for simpson's diversity, we would need to square each one...  
bay80s41410$freq\_sq <- bay80s41410$freq^2  
# and then sum them up, and divide from 1:  
simp\_bay80s41410 <- 1 / sum(bay80s41410$freq^2)  
simp\_bay80s41410

## [1] 1.624887

If we want to do that same thing using tidyverse, we can do the same steps, but pipe them through. One benefit of using pipes is that we can start with the main zoop data frame each time and not name new objects. This reduces ‘clutter’ in your global environment.

#first, calculating richness  
zoop %>%   
 filter(bay=="80"&sample\_date=="4/14/10") %>% #filter to the bay and sampling date  
 select(taxa) %>% #select just the taxa column  
 summarize(richness=n\_distinct(taxa)) #summarize the column and find the distinct entries (the number of different taxa)

## richness  
## 1 15

#now, calculating simpson's diversity  
zoop %>%   
 filter(bay=="80"&sample\_date=="4/14/10") %>%  
 mutate(total\_counted = sum(abund),  
 freq = abund/total\_counted,  
 frequency\_sq = freq^2) %>%  
 summarize(simpsons\_div = 1/sum(frequency\_sq))

## simpsons\_div  
## 1 1.624887

Now a bonus: the end of problem set 7 included a loop to find richness of all the sames in bay 80.

sampls <- unique(bay80$sample) # a list of the samples; this is what we'll loop through  
rich.summary <- data.frame(sample\_date = sampls) #create summary dataframe, to be filled in row by row  
  
for (i in 1:length(sampls)){ # for each one of the samples,  
 sub80 <- bay80[bay80$sample==sampls[i],] # create a subset of the data for that sample  
 rich.summary$rich[i] <- length(unique(na.omit(sub80$taxa))) # calculate richness and fill it it  
 # fill it in on the summary dataframe. notice the 'na.omit'... that's because I want a sample  
 # with only NA's to have a richness of 0, not 1.  
 }  
rich.summary # you can check the summary here.

## sample\_date rich  
## 1 3/17/10 13  
## 2 3/31/10 14  
## 3 2/18/09 12  
## 4 4/14/10 15  
## 5 3/4/10 9  
## 6 2/17/10 10  
## 7 1/22/09 9  
## 8 12/15/09 10  
## 9 4/16/09 14  
## 10 4/28/10 13  
## 11 5/28/09 12  
## 12 4/2/09 16  
## 13 3/12/09 11  
## 14 2/3/10 12  
## 15 12/29/09 10  
## 16 4/30/09 5  
## 17 3/19/09 14  
## 18 5/14/09 5  
## 19 1/20/10 8  
## 20 5/11/10 10  
## 21 1/6/10 12

I’m sure you’ve guessed this by now, but we can also do the same thing using tidyverse syntax.

bay80\_richness <- zoop %>%   
 filter(bay == "80") %>%  
 group\_by(sample\_date) %>%  
 summarize(richness = n\_distinct(taxa,na.rm=T))  
bay80\_richness

## # A tibble: 21 × 2  
## sample\_date richness  
## <chr> <int>  
## 1 1/20/10 8  
## 2 1/22/09 9  
## 3 1/6/10 12  
## 4 12/15/09 10  
## 5 12/29/09 10  
## 6 2/17/10 10  
## 7 2/18/09 12  
## 8 2/3/10 12  
## 9 3/12/09 11  
## 10 3/17/10 13  
## # ℹ 11 more rows

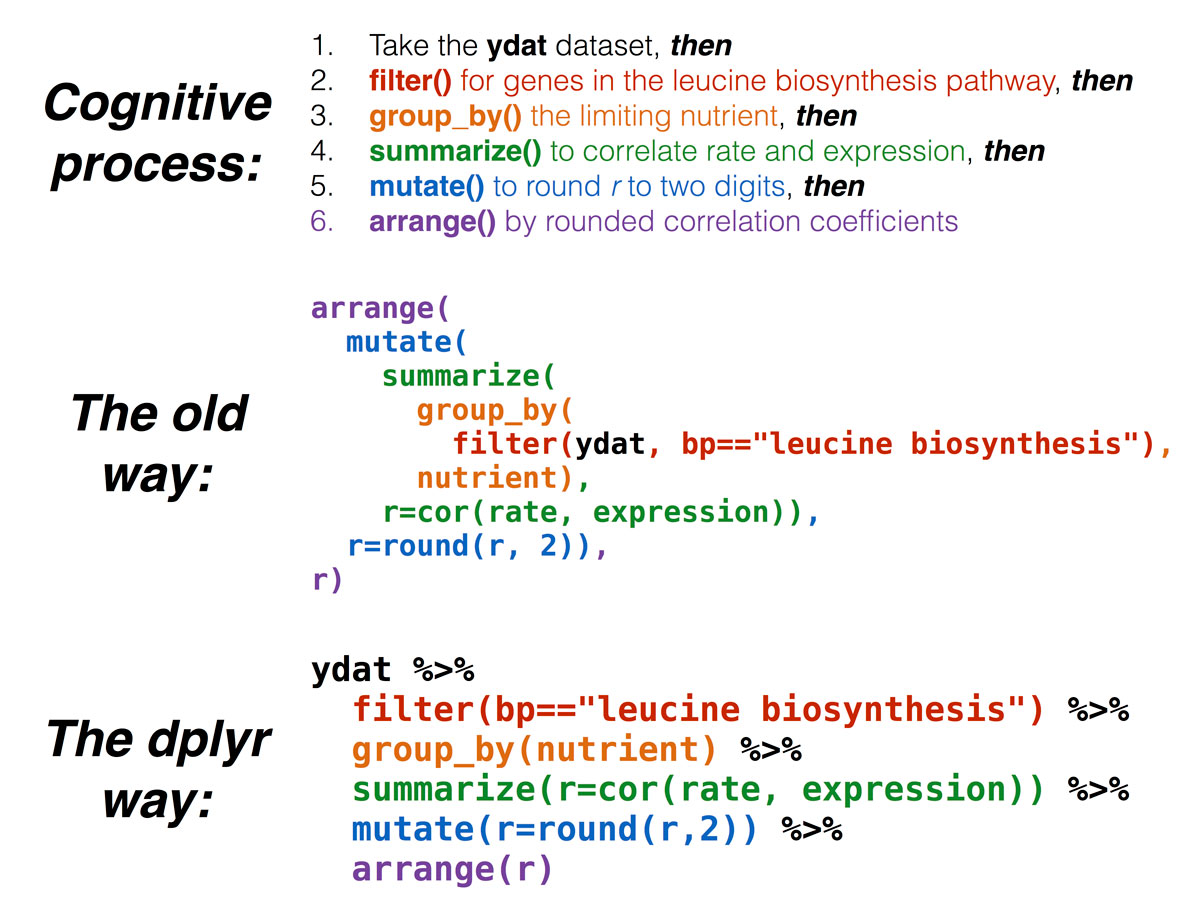
If you want to convince yourself that these two methods output the same results, we can merge the two outputs together and compare the richness we calculated for each sample date. Merging, using the merge() command, is a useful skill to know and is helpful if you’re bringing together multiple data sources.

#here's how we would merge the two outputs together  
#the arguements we need are the two data frames we're merging and what column they have in common  
merge(bay80\_richness,rich.summary,by="sample\_date")

## sample\_date richness rich  
## 1 1/20/10 8 8  
## 2 1/22/09 9 9  
## 3 1/6/10 12 12  
## 4 12/15/09 10 10  
## 5 12/29/09 10 10  
## 6 2/17/10 10 10  
## 7 2/18/09 12 12  
## 8 2/3/10 12 12  
## 9 3/12/09 11 11  
## 10 3/17/10 13 13  
## 11 3/19/09 14 14  
## 12 3/31/10 14 14  
## 13 3/4/10 9 9  
## 14 4/14/10 15 15  
## 15 4/16/09 14 14  
## 16 4/2/09 16 16  
## 17 4/28/10 13 13  
## 18 4/30/09 5 5  
## 19 5/11/10 10 10  
## 20 5/14/09 5 5  
## 21 5/28/09 12 12

**A last example of one of the benefits of using pipes in tidyverse:**

We haven’t run into this much in our problem sets, but one of the disadvantages of using some of the base functions in R is that you often need to nest multiple commands and you end up with a lot of nested parentheses. Using pipes in tidyverse, you can break up these nested parentheses with the %>% operator.

This graphic from [a data science workshop](https://4va.github.io/biodatasci/index.html) highlights what these differences might look like.  


### Switching topics

We’ll now move into a couple topics that were not in previous problem sets, but may be useful for a couple of the group projects.

#### Topic 7: Finding the distance between points (coordinates)

There is a package you can use to find the distance between points using their latitude and longitude.

If you want to learn more about this package, you can follow [this link to the package documentation](https://cran.r-project.org/web/packages/geosphere/index.html).

The reference manual and a vignette are linked under documentation and are good references for using the package.

#call the package from your library  
library(geosphere)  
#read in a file with latitude and longitude for your points of interest  
location <- read.csv("location.csv")  
  
#One method for determining the distance between two points is using the Haversine method. We'll use the distm function in the geosphere package and then specify the Haversine method  
  
#first we need to get a dataframe that is just the latitude and longitude of our sites  
#for this example, those are columns 4 and 5  
loc <- location[4:5]  
  
#if we want to do this with tidyverse, we would use:  
loc2 <- location %>% select(c(better\_lat,better\_long))  
  
distances<-as.data.frame(distm(loc,fun=distHaversine))  
  
distances

## V1 V2 V3 V4 V5 V6 V7  
## 1 0.000 660.877 8022.8087 7900.37420 7837.35339 8718.7374 8361.5898  
## 2 660.877 0.000 7380.4758 7257.84625 7194.56792 8085.0650 7732.1484  
## 3 8022.809 7380.476 0.0000 122.91365 187.02067 809.2943 664.6537  
## 4 7900.374 7257.846 122.9136 0.00000 64.39771 920.4051 739.8633  
## 5 7837.353 7194.568 187.0207 64.39771 0.00000 982.1989 789.1201  
## 6 8718.737 8085.065 809.2943 920.40513 982.19893 0.0000 397.4869  
## 7 8361.590 7732.148 664.6537 739.86327 789.12014 397.4869 0.0000  
## 8 8149.464 8810.321 16060.1671 15939.88189 15878.87452 16697.3855 16318.8813  
## 9 7291.742 7952.547 15201.8928 15081.53153 15020.46325 15841.0240 15463.1125  
## 10 8737.038 9397.777 16630.6414 16510.64050 16449.86301 17262.2034 16882.2936  
## 11 22834.952 23491.412 30821.7077 30700.55699 30638.80284 31473.3718 31097.4343  
## 12 22534.567 23191.140 30520.4745 30399.34074 30337.60171 31171.7885 30795.7697  
## 13 21559.856 22217.964 29529.7296 29408.93970 29347.49859 30173.6168 29795.7191  
## 14 21328.766 21986.988 29297.2879 29176.52326 29115.10371 29940.6724 29562.6591  
## 15 21273.830 21932.145 29241.0995 29120.36033 29058.96235 29883.9551 29505.8126  
## 16 7776.517 7164.361 1151.1077 1133.27302 1136.71848 1214.4204 827.8534  
## V8 V9 V10 V11 V12 V13 V14  
## 1 8149.4638 7291.7420 8737.0380 22834.952 22534.567 21559.8559 21328.76599  
## 2 8810.3209 7952.5467 9397.7774 23491.412 23191.140 22217.9642 21986.98757  
## 3 16060.1671 15201.8928 16630.6414 30821.708 30520.474 29529.7296 29297.28789  
## 4 15939.8819 15081.5315 16510.6405 30700.557 30399.341 29408.9397 29176.52326  
## 5 15878.8745 15020.4632 16449.8630 30638.803 30337.602 29347.4986 29115.10371  
## 6 16697.3855 15841.0240 17262.2034 31473.372 31171.789 30173.6168 29940.67241  
## 7 16318.8813 15463.1125 16882.2936 31097.434 30795.770 29795.7191 29562.65907  
## 8 0.0000 859.6085 597.8743 14784.663 14482.670 13477.0371 13243.88217  
## 9 859.6085 0.0000 1446.0147 15637.232 15335.340 14332.6579 14099.66611  
## 10 597.8743 1446.0147 0.0000 14232.782 13930.623 12918.0293 12684.47188  
## 11 14784.6632 15637.2324 14232.7823 0.000 302.263 1385.0338 1614.00022  
## 12 14482.6698 15335.3397 13930.6233 302.263 0.000 1097.5474 1322.69540  
## 13 13477.0371 14332.6579 12918.0293 1385.034 1097.547 0.0000 234.68694  
## 14 13243.8822 14099.6661 12684.4719 1614.000 1322.695 234.6869 0.00000  
## 15 13186.9641 14042.9311 12627.0568 1677.841 1387.056 295.8171 64.82476  
## 16 15642.3681 14789.5076 16199.7238 30426.664 30124.736 29117.5697 28884.09397  
## V15 V16  
## 1 21273.83022 7776.5168  
## 2 21932.14533 7164.3609  
## 3 29241.09954 1151.1077  
## 4 29120.36033 1133.2730  
## 5 29058.96235 1136.7185  
## 6 29883.95506 1214.4204  
## 7 29505.81263 827.8534  
## 8 13186.96408 15642.3681  
## 9 14042.93105 14789.5076  
## 10 12627.05676 16199.7238  
## 11 1677.84130 30426.6644  
## 12 1387.05567 30124.7364  
## 13 295.81706 29117.5697  
## 14 64.82476 28884.0940  
## 15 0.00000 28826.7493  
## 16 28826.74929 0.0000

The output of the distm function is a matrix of distances between our points in meters. You’ll have to reference the original list to identify which bays each distance refers to.

In the distm command, we specified fun = distHaversine. There are other options - distGeo or distCosine being two.

#### Topic 8: working with weather data (or other messy data)

Often when working with data, we will look to outside sources to supplement our data. Sometimes, those outside sources will give us data that is not easily worked with. In this example, we’ll go through what cleaning data may look like,

The data in this example were collected from the [NOAA Climate Data Online page](https://www.ncei.noaa.gov/cdo-web/).  
This source gives weather data from multiple weather stations in the area near a city (Augusta in this case). We may choose to only use some weather stations depending on how complete these data are.

weather <- read.csv("Augusta\_weather\_data.csv")  
  
#what is the structure of the data  
str(weather)

## 'data.frame': 5934 obs. of 40 variables:  
## $ STATION : chr "US1SCAK0004" "US1SCAK0004" "US1SCAK0004" "US1SCAK0004" ...  
## $ NAME : chr "NORTH AUGUSTA 4.2 NE, SC US" "NORTH AUGUSTA 4.2 NE, SC US" "NORTH AUGUSTA 4.2 NE, SC US" "NORTH AUGUSTA 4.2 NE, SC US" ...  
## $ LATITUDE : num 33.6 33.6 33.6 33.6 33.6 ...  
## $ LONGITUDE: num -81.9 -81.9 -81.9 -81.9 -81.9 ...  
## $ ELEVATION: num 74.7 74.7 74.7 74.7 74.7 74.7 74.7 74.7 74.7 74.7 ...  
## $ DATE : chr "1/4/2009" "1/7/2009" "1/11/2009" "1/18/2009" ...  
## $ AWND : num NA NA NA NA NA NA NA NA NA NA ...  
## $ DAPR : int NA NA NA NA NA NA NA NA NA NA ...  
## $ FMTM : int NA NA NA NA NA NA NA NA NA NA ...  
## $ MDPR : num NA NA NA NA NA NA NA NA NA NA ...  
## $ PGTM : int NA NA NA NA NA NA NA NA NA NA ...  
## $ PRCP : num 0.12 0.39 0.45 0.29 0.04 0.27 0.01 0.13 0.12 0.08 ...  
## $ SNOW : num NA NA NA NA 0 NA NA NA NA NA ...  
## $ SNWD : num NA NA NA NA NA NA NA NA NA NA ...  
## $ TAVG : logi NA NA NA NA NA NA ...  
## $ TMAX : int NA NA NA NA NA NA NA NA NA NA ...  
## $ TMIN : int NA NA NA NA NA NA NA NA NA NA ...  
## $ TOBS : logi NA NA NA NA NA NA ...  
## $ WDF2 : int NA NA NA NA NA NA NA NA NA NA ...  
## $ WDF5 : int NA NA NA NA NA NA NA NA NA NA ...  
## $ WSF2 : num NA NA NA NA NA NA NA NA NA NA ...  
## $ WSF5 : num NA NA NA NA NA NA NA NA NA NA ...  
## $ WT01 : int NA NA NA NA NA NA NA NA NA NA ...  
## $ WT02 : int NA NA NA NA NA NA NA NA NA NA ...  
## $ WT03 : int NA NA NA NA NA NA NA NA NA NA ...  
## $ WT04 : logi NA NA NA NA NA NA ...  
## $ WT05 : int NA NA NA NA NA NA NA NA NA NA ...  
## $ WT06 : logi NA NA NA NA NA NA ...  
## $ WT07 : int NA NA NA NA NA NA NA NA NA NA ...  
## $ WT08 : int NA NA NA NA NA NA NA NA NA NA ...  
## $ WT09 : int NA NA NA NA NA NA NA NA NA NA ...  
## $ WT11 : int NA NA NA NA NA NA NA NA NA NA ...  
## $ WT13 : int NA NA NA NA NA NA NA NA NA NA ...  
## $ WT14 : int NA NA NA NA NA NA NA NA NA NA ...  
## $ WT16 : int NA NA NA NA NA NA NA NA NA NA ...  
## $ WT17 : int NA NA NA NA NA NA NA NA NA NA ...  
## $ WT18 : int NA NA NA NA NA NA NA NA NA NA ...  
## $ WT19 : int NA NA NA NA NA NA NA NA NA NA ...  
## $ WT21 : int NA NA NA NA NA NA NA NA NA NA ...  
## $ WT22 : int NA NA NA NA NA NA NA NA NA NA ...

#how many weather stations are there?  
length(unique(weather$STATION))

## [1] 14

#let's rename some of these column names  
weather <- weather %>% rename(precipitation = PRCP,  
 max\_temp = TMAX,  
 min\_temp = TMIN,  
 average\_temp = TAVG)  
  
#let's look at which of the weather stations has the most observations each of these weather variables  
# !is.na filters out the NA values and n() counts the number of rows (non-NA observations)  
  
weather %>% filter(!is.na(precipitation)) %>%  
 group\_by(STATION) %>%  
 summarize(precip\_obs = n())

## # A tibble: 14 × 2  
## STATION precip\_obs  
## <chr> <int>  
## 1 US1GACU0001 158  
## 2 US1GACU0002 433  
## 3 US1GACU0003 269  
## 4 US1GACU0004 25  
## 5 US1GARC0001 124  
## 6 US1GARC0002 720  
## 7 US1SCAK0004 153  
## 8 US1SCAK0006 13  
## 9 US1SCAK0010 712  
## 10 US1SCAK0013 721  
## 11 US1SCAK0017 621  
## 12 US1SCAK0019 323  
## 13 USW00003820 790  
## 14 USW00013837 789

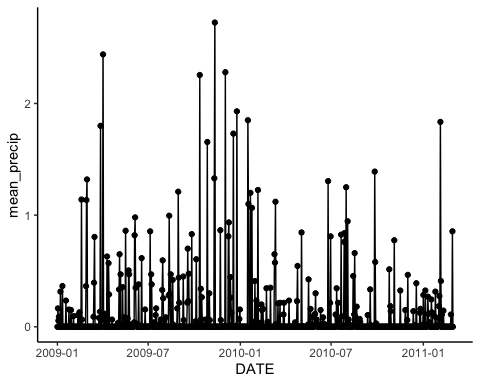
weather %>% filter(!is.na(max\_temp)) %>%  
 group\_by(STATION) %>%  
 summarize(precip\_obs = n())

## # A tibble: 2 × 2  
## STATION precip\_obs  
## <chr> <int>  
## 1 USW00003820 790  
## 2 USW00013837 787

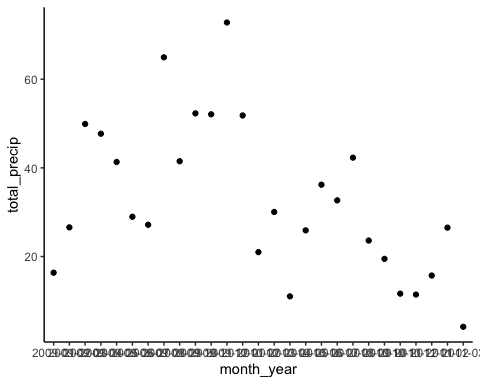
There are two weather stations, USW00003820 and USW00013837, that have more complete data than the rest. We’ll filter our data to just those weather stations.

Now we’ll summarize the weather data and look at precipitation and temperature changes over time.

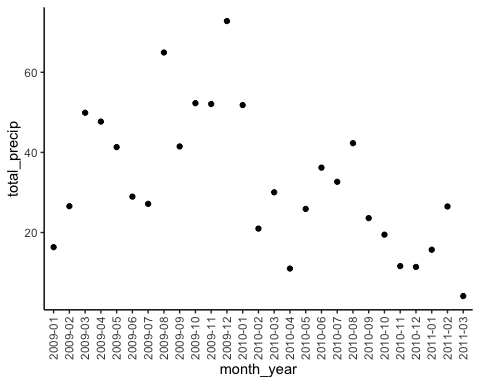
#first, let's make sure our date is in date format  
weather$DATE <- mdy(weather$DATE)  
  
#filter the data and summarize precipitation and max temperature data  
weather\_summary <- weather %>% filter(STATION=="USW00003820"|STATION=="USW00013837") %>%  
 group\_by(DATE) %>%  
 summarize(mean\_precip = mean(precipitation),  
 mean\_maxT = mean(max\_temp))  
  
#let's look at precipitation over time  
ggplot(weather\_summary,aes(x=DATE,y=mean\_precip)) +  
 geom\_point() +  
 geom\_line() +  
 theme\_classic()



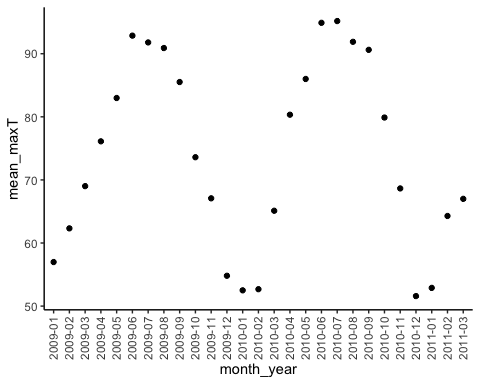
#that's hard to interpret. Maybe it would be easier to look at with summaries for each month  
#first we'll need a column with the month and year and then use that to group by and summarize like we did before  
weather\_summary2 <- weather %>%   
 mutate(month\_year = format(DATE,"%Y-%m")) %>%  
 group\_by(month\_year) %>%  
 summarize(total\_precip = sum(precipitation,na.rm=T),  
 mean\_maxT = mean(max\_temp,na.rm=T))  
  
ggplot(weather\_summary2,aes(x=month\_year,y=total\_precip)) +  
 geom\_point() +  
 theme\_classic()



#it's hard to read the x-axis, so we can use another line in our ggplot code to rotate the text  
ggplot(weather\_summary2,aes(x=month\_year,y=total\_precip)) +  
 geom\_point() +  
 theme\_classic() +  
 theme(axis.text.x = element\_text(angle = 90, vjust = 0.5, hjust=1))



#we can use the same code but replace total\_precip with mean\_maxT to look at average temperature  
ggplot(weather\_summary2,aes(x=month\_year,y=mean\_maxT)) +  
 geom\_point() +  
 theme\_classic() +  
 theme(axis.text.x = element\_text(angle = 90, vjust = 0.5, hjust=1))



**And that’s the end of the cheat sheet!**