Hydroinformatics

JP Gannon

2024-01-09

Table of contents

Welcome

Introduction

To help me keep get an idea of who is using this resource so I can improve it in the future, please consider filling out any or all of this survey: https://forms.gle/6Zcntzvr1wZZUh6S7 Thanks!

This bookdown contains the notes and most exercises for a course on data analysis techniques in hydrology using the programming language R. The material will be updated each time the course is taught. If new topics are added, the topics they replace will be left, in case they are useful to others.

I hope these materials can be a resource to those teaching themselves R for hydrologic analysis and/or for instructors who may want to use a lesson or two or the entire course. Each chapter in this bookdown is linked to a github repository where the code can be downloaded or copied to another github account.

If you have questions, suggestions, or would like activity answer keys, etc. please email me at jpgannon at vt.edu

The following resources, among others, were very helpful when compiling the chapters of this book. They are also linked in specific chapters, along with other resources. These are great resources if you want to dig deeper into topics covered in this bookdown.

R for Data Science: https://r4ds.had.co.nz/
Statistical Methods in Water Resources: https://pubs.er.usgs.gov/publication/tm4A3
Geocomputation with R: https://geocompr.robinlovelace.net/

How to use these materials

At the top of each chapter there is a link to a github repository. In each repository is the code that produces each chapter and a version where the code chunks within it are blank. These repositories are all template repositories, so you can easily copy them to your own github space by clicking *Use This Template* on the repo page.

In my class, I work through the each document, live coding with students following along.Typically I ask students to watch as I code and explain the chunk and then replicate it on their computer. Depending on the lesson, I will ask students to try some of the chunks before I show them the code as an in-class activity. Some chunks are explicitly designed for this purpose and are typically labeled a “challenge”.

Chapters called ACTIVITY are either homework or class-period-long in-class activities. The code chunks in these are therefore blank. If you would like a key for any of these, please just send me an email.

1 Intro to Plotting 3

Get this document and a version with empty code chunks at the template repository on github: https://github.com/VT-Hydroinformatics/1-Intro-plotting-R

1.1 Download and install tidyverse library

We will use the tidyverse a lot this semester. It is a suite of packages that handles plotting and data wrangling efficiently.

You only have to install the library once. You have to load it using the library() function each time you start an R session.

```
#install.packages("tidyverse")
library(tidyverse)
```

```
-- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
v dplyr     1.1.2     v readr     2.1.4
v forcats   1.0.0     v stringr   1.5.0
v ggplot2   3.4.2     v tibble    3.2.1
v lubridate 1.9.2     v tidyr     1.3.0
v purrr     1.0.1
-- Conflicts ------------------------------------------ tidyverse_conflicts() --
x dplyr::filter() masks stats::filter()
x dplyr::lag()    masks stats::lag()
i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to becom
```

```
#library(webexercises)
```

1.2 Reading data

The following lines will read in the data we will use for this exercise. Don't worry about this right now beyond running it, we will talk more about it later.

```r
Pine <- read_csv("PINE_Jan-Mar_2010.csv")
```

```
Rows: 2160 Columns: 8
-- Column specification --------------------------------------------------------
Delimiter: ","
chr  (2): StationID, surrogate
dbl  (5): cfs, year, quarter, month, day
dttm (1): datetime

i Use `spec()` to retrieve the full column specification for this data.
i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```r
SNP <- read_csv("PINE_NFDR_Jan-Mar_2010.csv")
```

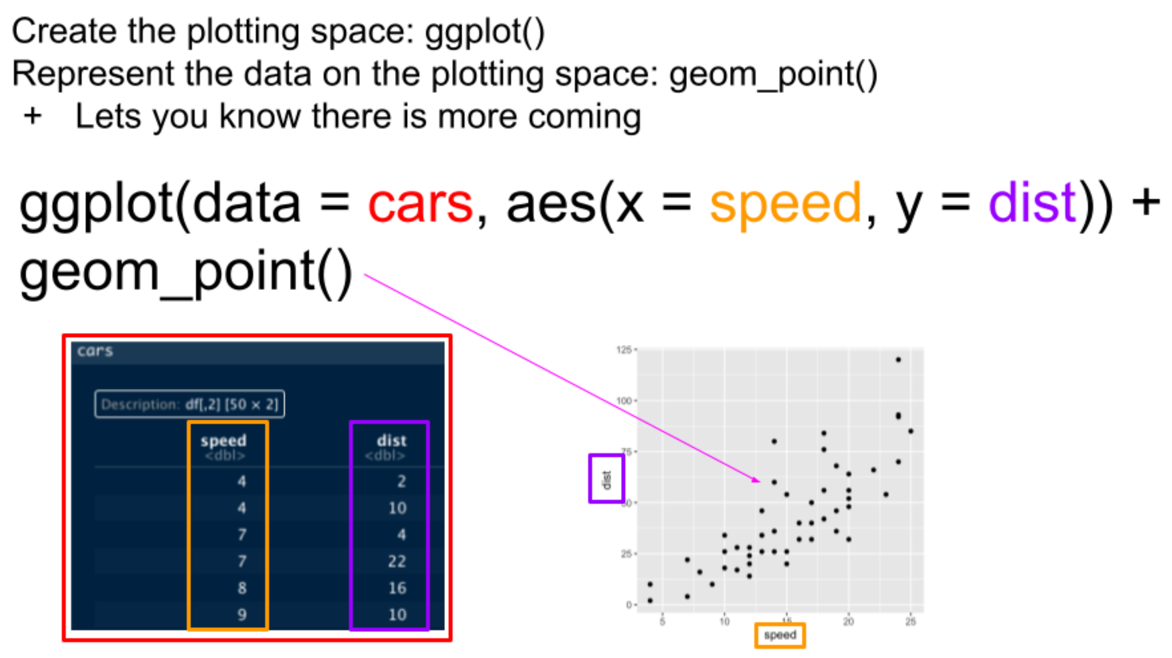
```
Rows: 4320 Columns: 8
-- Column specification --------------------------------------------------------
Delimiter: ","
chr  (2): StationID, surrogate
dbl  (5): cfs, year, quarter, month, day
dttm (1): datetime

i Use `spec()` to retrieve the full column specification for this data.
i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```r
RBI <- read_csv("Flashy_Dat_Subset.csv")
```

```
Rows: 49 Columns: 26
-- Column specification --------------------------------------------------------
Delimiter: ","
chr  (4): STANAME, STATE, CLASS, AGGECOREGION
dbl (22): site_no, RBI, RBIrank, DRAIN_SQKM, HUC02, LAT_GAGE, LNG_GAGE, PPTA...

i Use `spec()` to retrieve the full column specification for this data.
i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```


Create the plotting space: ggplot()
Represent the data on the plotting space: geom_point()
+ Lets you know there is more coming

```
ggplot(data = cars, aes(x = speed, y = dist)) +
geom_point()
```

cars

Description: df[,2] [50 × 2]

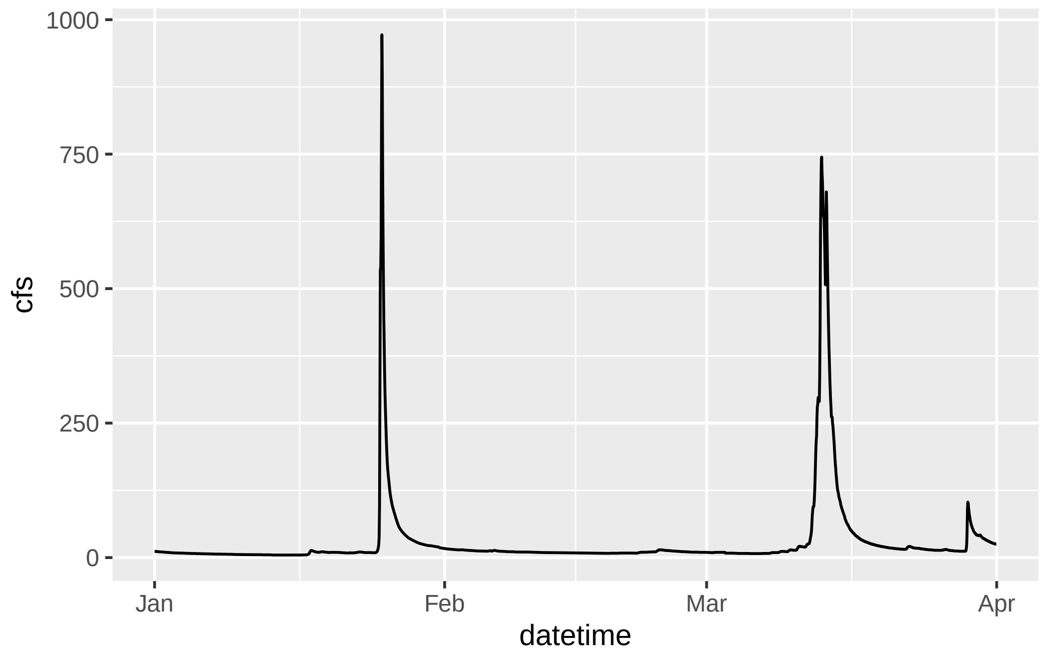
speed <dbl>	dist <dbl>
4	2
4	10
7	4
7	22
8	16
9	10

Figure 1.1: Basic ggplot syntax

1.3 Our first ggplot

Let's look at the Pine data, plotting streamflow (the cfs column) by the date (datetime column). We will show the time series as a line.

```
ggplot(data = Pine, aes(x = datetime, y = cfs))+
  geom_line()
```



1.4 Change point type

Now let's make the same plot but show the data as points, using the pch parameter in geom_point() we can change the point type to any of the following:

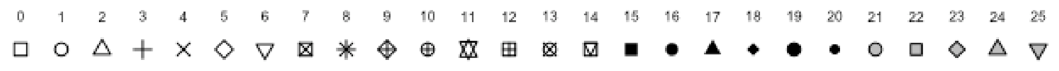
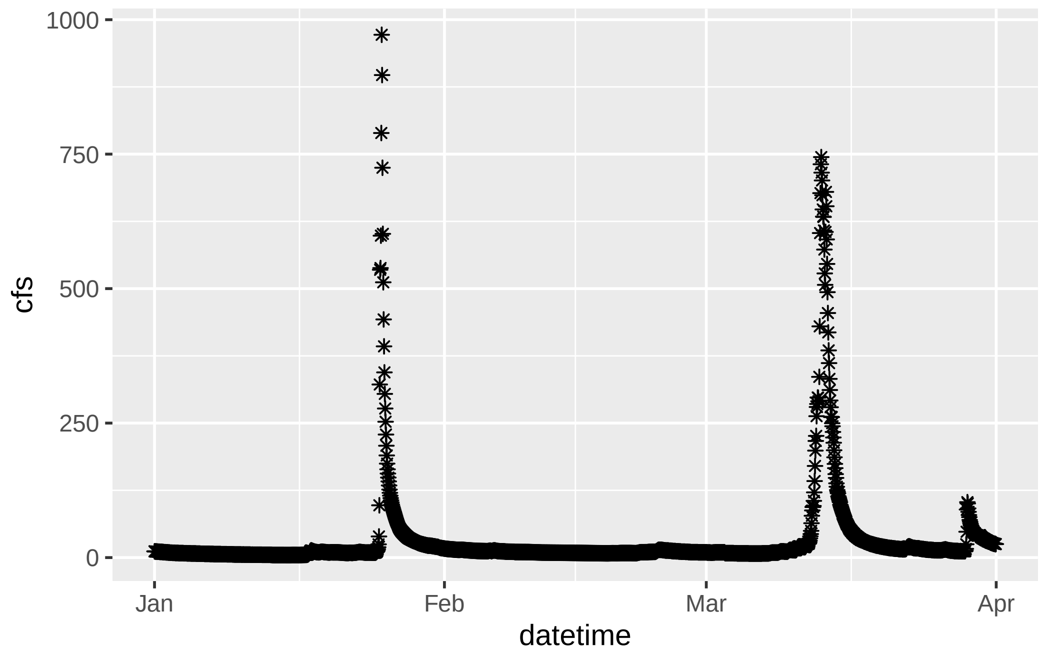


Figure 1.2: pch options from R help file

```r
ggplot(data = Pine, aes(x = datetime, y = cfs))+
  geom_point(pch = 8)
```



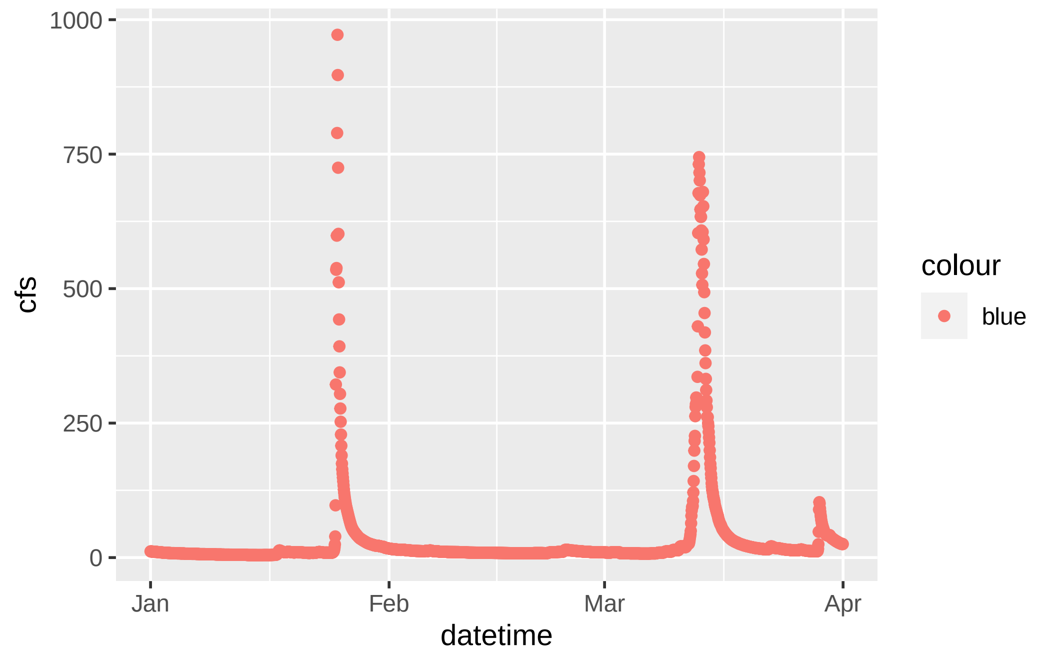
1.5 Set colors

We can also "easily" change the color. Easily is in quotes because this often trips people up. If you put color = "blue" in the aesthetic function, think about what that is telling ggplot. It says "control the color using"blue" ". That doesn't make a whole lot of sense, so neither does the output… Try it.

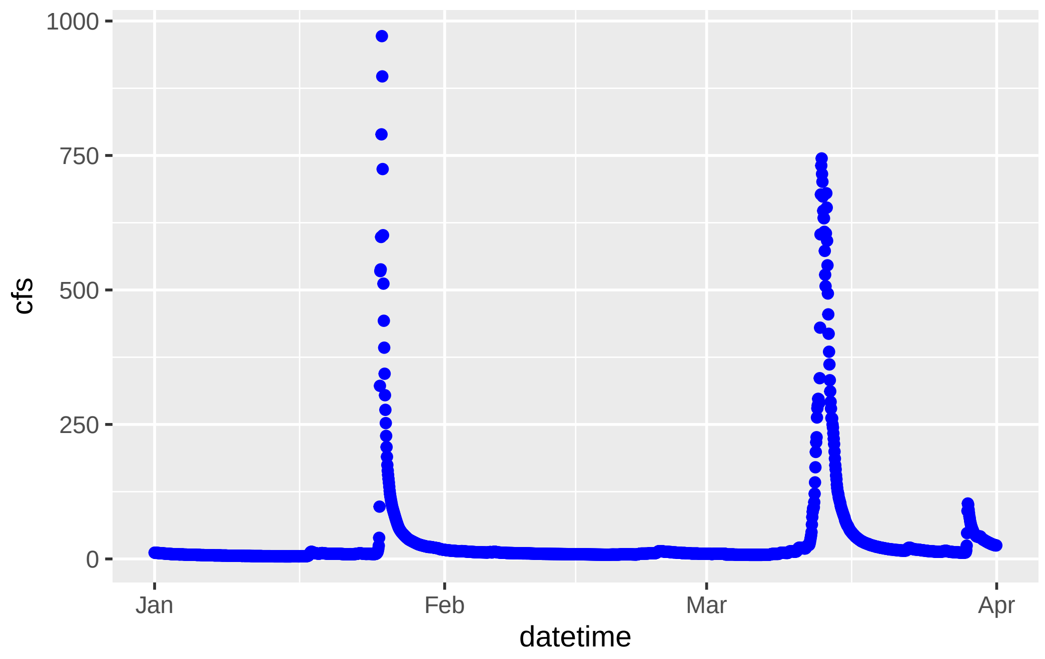
What happens is that if color = "blue" is in the aesthetic, you are telling R that the color used in the geom represents "blue". This is very useful if you have multiple geoms in your plot, are coloring them differently, and are building a legend. But if you are just trying to color the points, it kind of feels like R is trolling you… doesn't it?

Take the color = "blue" out of the aesthetic and you're golden.

```
ggplot(data = Pine, aes(datetime, y = cfs, color = "blue"))+
  geom_point()
```



```r
ggplot(data = Pine, aes(x = datetime, y = cfs))+
  geom_point(color = "blue")
```



1.6 Controlling color with a third variable and other functions

Let's plot the data as a line again, but play with it a bit.

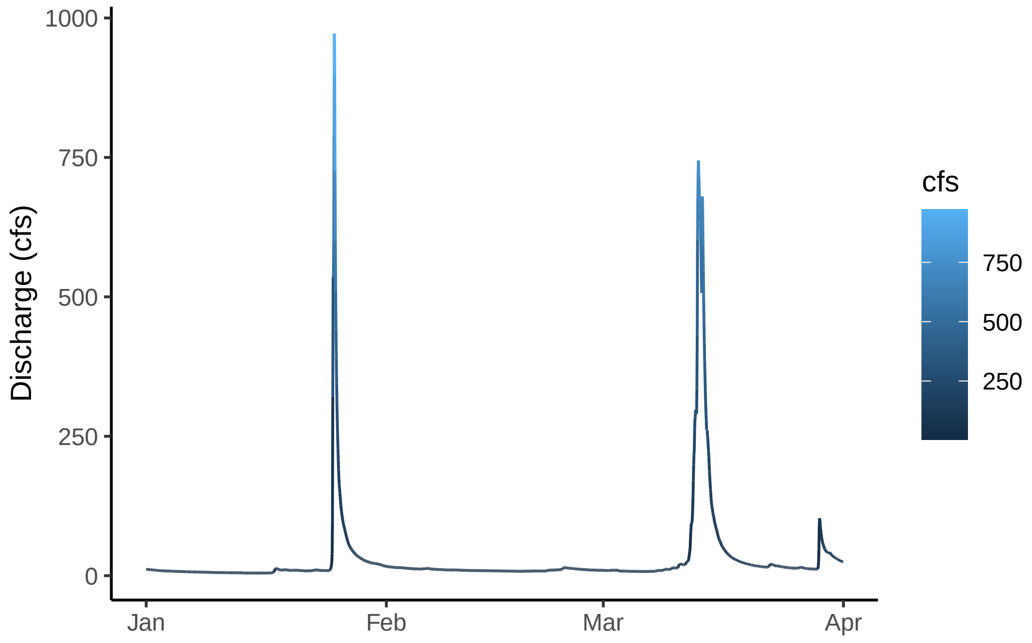
First: make the line blue

Second: change the theme

Third: change the axis labels

Fourth: color by discharge

```
ggplot(data = Pine, aes(x = datetime, y = cfs, color = cfs))+
  geom_line()+
  ylab("Discharge (cfs)")+
  xlab(element_blank())+
  theme_classic()
```



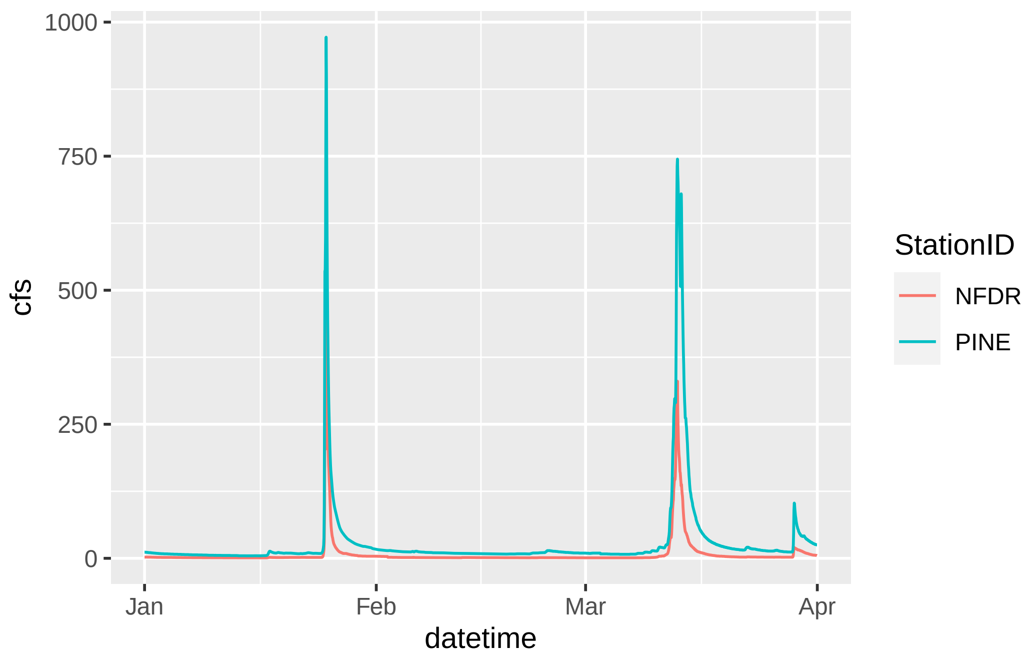
1.7 Plotting multiple groups

The SNP dataset has two different streams: Pine and NFDR

We can look at the two of those a couple of different ways

First, make two lines, colored by the stream by adding color = to your aesthetic.

```r
ggplot(data = SNP, aes(x = datetime,y = cfs, color = StationID)) +
  geom_line()
```

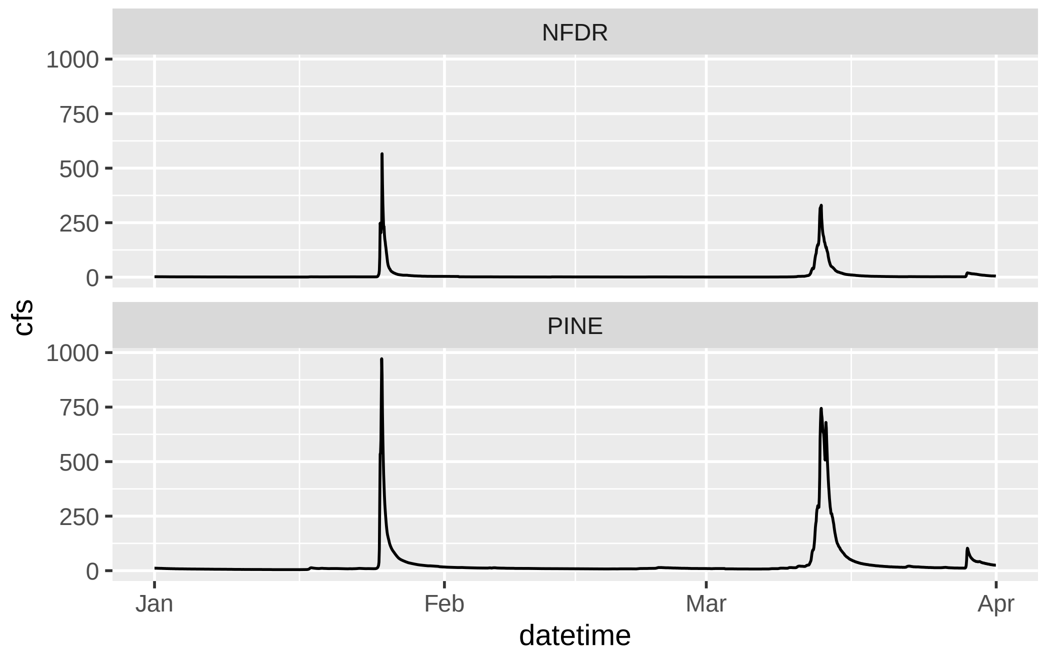


1.8 Facets

We can also use facets.

You must tell the facet_wrap what variable to use to make the separate panels (facet =). It'll decide how to orient them or you can tell it how. We want them to be on top of each other so we are going to tell it we want 2 rows by setting nrow = 2. Note that we have to put the column used to make the facets in quotes after facets =

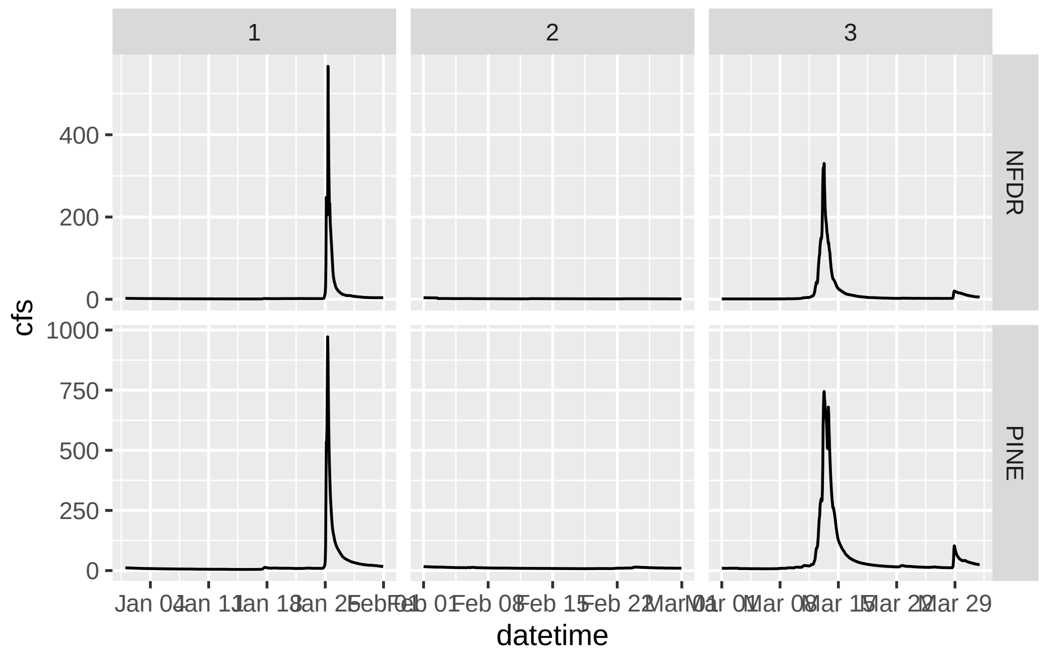
```r
ggplot(data = SNP, aes(x = datetime,y = cfs)) +
  geom_line() +
  facet_wrap(facets = "StationID", nrow = 2)
```



1.9 Two variable faceting

You can also use facet_grid() to break your plots up into panels based on two variables. Below we will create a panel for each month in each watershed. Adding scales = "free" allows facet_grid to change the axes. By default, all axes will be the same. This is often what we want, so we can more easily compare magnitudes, but sometimes we are looking for patterns more, so we may want to let the axes have whatever range works for the individual plots.

```
ggplot(data = SNP, aes(x = datetime,y = cfs)) +
  geom_line() +
  facet_grid(StationID ~ month, scales = "free")
```

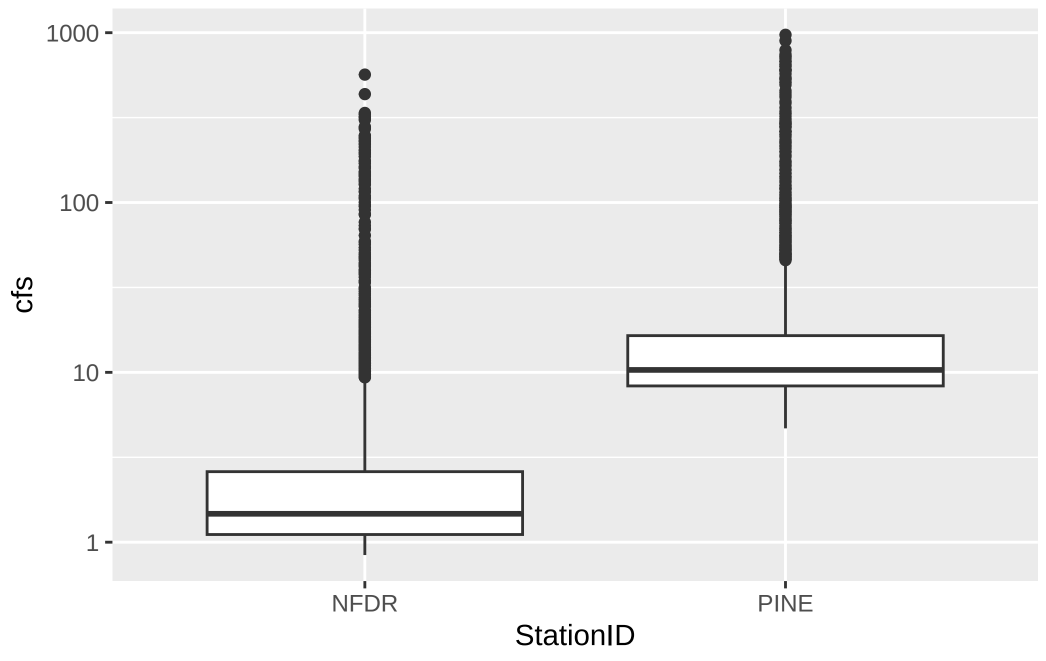


1.10 Boxplots

We can look at these data in other ways as well. A very useful way to look at the variation of two groups is to use a boxplot.

Because the data span several orders of magnitude, we will have to log the y axis to see the differences between the two streams. We do that by adding scale_y_log10()

```
ggplot(data = SNP, aes(x = StationID, y = cfs)) +
  stat_boxplot()+
  scale_y_log10()
```



1.11 More about color, size, etc

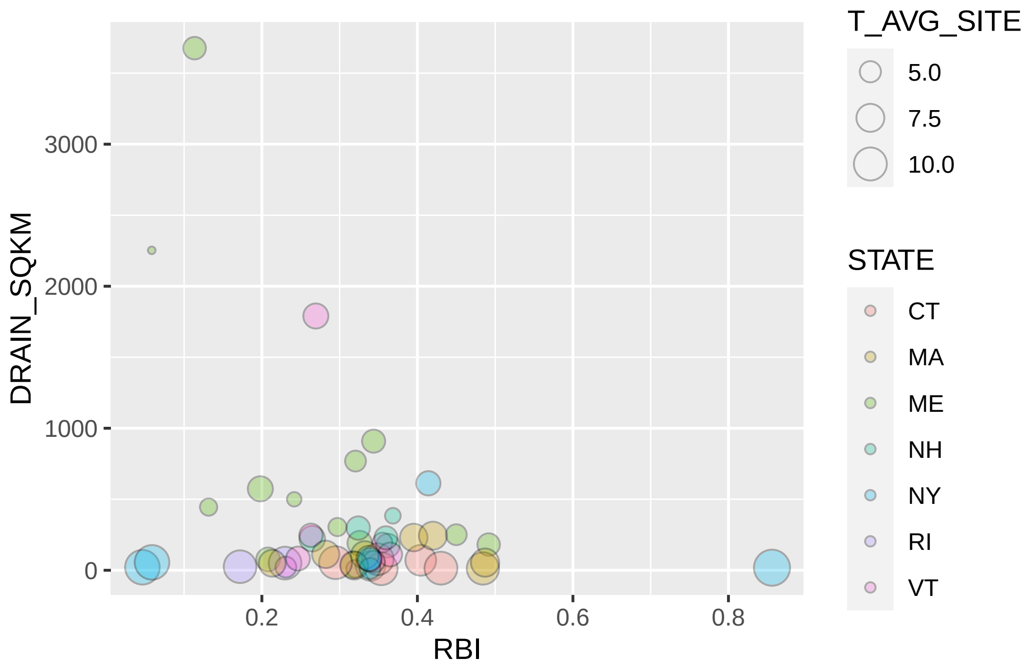
Let's play around a bit with controlling color, point size, etc with other data.

We can control the size of points by putting size = in the aes() and color by putting color =

If you use a point type that has a background, like #21, you can also set the background color using bg =

If points are too close together to see them all you can use a hollow point type or set the alpha lower so the points are transparent (alpha =)

```r
ggplot(RBI, aes(RBI, DRAIN_SQKM, size = T_AVG_SITE, bg = STATE))+
  geom_point(pch = 21, alpha = 0.3)
```

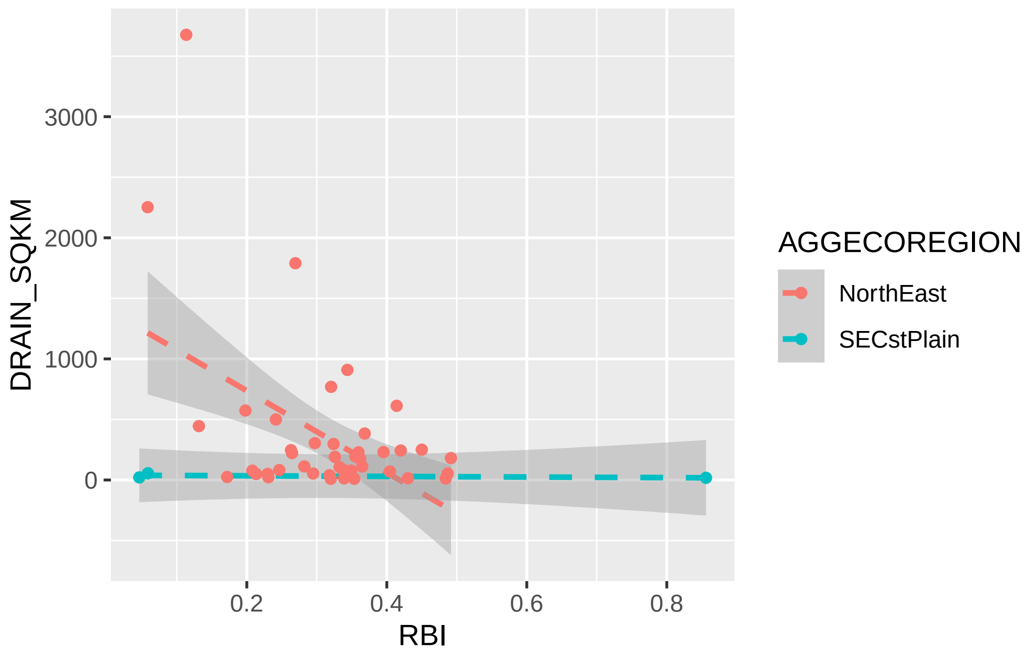


1.12 Multiple geoms

Finally: You can add multiple geoms to the same plot. Examples of when you might want to do this are when you are showing a line fit and want to show the points as well, or maybe showing a boxplot and want to show the data behind it. You simply add additional geom_… lines to add additional geoms.

```r
ggplot(RBI, aes(RBI, DRAIN_SQKM, color = AGGECOREGION))+
  stat_smooth(method = "lm", linetype = 2)+
  geom_point()
```

```
`geom_smooth()` using formula = 'y ~ x'
```


DRAIN_SQKM
3000
2000
1000
0
0.2
0.4
0.6
0.8
RBI
AGGECOREGION
NorthEast
SECstPlain

2 R Tidyverse Programming Basics

Get this document and a version with empty code chunks at the template repository on github: https://github.com/VT-Hydroinformatics/2-Programming-Basics

2.1 Introduction

We have messed around with plotting a bit and you've seen a little of what R can do. So now let's review or introduce you to some basics. Even if you have worked in R before, it is good to be remind of/practice with this stuff, so stay tuned in!

This exercise covers most of the same principles as two chapters in R for Data Science

Workflow: basics (https://r4ds.had.co.nz/workflow-basics.html)

Data transformation (https://r4ds.had.co.nz/transform.html)

2.2 You can use R as a calculator

If you just type numbers and operators in, R will spit out the results

```r
1 + 2
```

```
[1] 3
```

2.3 You can create new objects using <-

Yea yea, = does the same thing. But use <-. We will call <- assignment or assignment operator. When we are coding in R we use <- to assign values to objects and = to set values for parameters in functions. Using <- helps us differentiate between the two. Norms for formatting are important because they help us understand what code is doing, especially when stuff gets complex.

Oh, one more thing: Surround operators with spaces. Don't code like a gorilla.

x <- 1 looks better than x<-1 and if you disagree you are wrong. :)

You can assign single numbers or entire chunks of data using <-

So if you had an object called my_data and wanted to copy it into my_new_data you could do:

my_new_data <- my_data

You can then recall/print the values in an object by just typing the name by itself.

In the code chunk below, assign a 3 to the object "y" and then print it out.

```r
y <- 3
y
```

```
[1] 3
```

If you want to assign multiple values, you have to put them in the function c() c means combine. R doesn't know what to do if you just give it a bunch of values with space or commas, but if you put them as arguments in the combine function, it'll make them into a vector.

Any time you need to use several values, even passing as an argument to a function, you have to put them in c() or it won't work.

```r
a <- c(1,2,3,4)
a
```

```
[1] 1 2 3 4
```

When you are creating objects, try to give them meaningful names so you can remember what they are. You can't have spaces or operators that mean something else as part of a name. And remember, everything is case sensitive.

Assign the value 5.4 to water_pH and then try to recall it by typing "water_ph"

```r
water_pH <- 5.4

#water_ph
```

You can also set objects equal to strings, or values that have letters in them. To do this you just have to put the value in quotes, otherwise R will think it is an object name and tell you it doesn't exist.

Try: name <- "JP" and then name <- JP

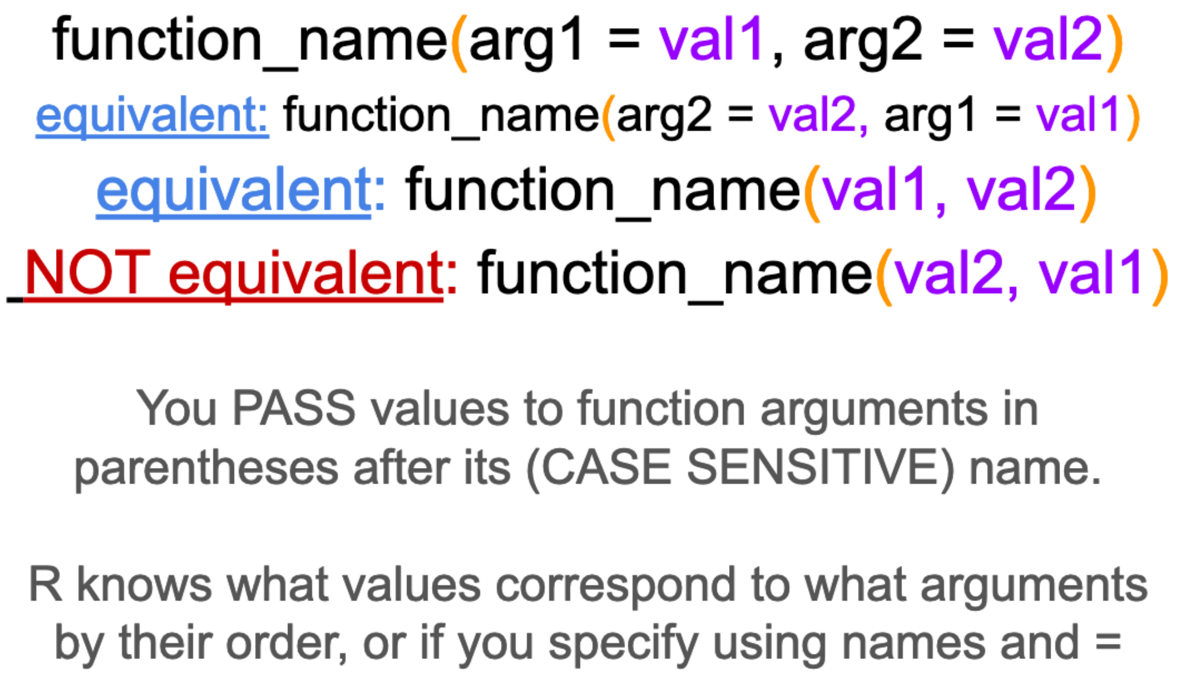
What happens if you forget the ending parenthesis?

Try: name <- “JP

R can be cryptic with it’s error messages or other responses, but once you get used to them, you know exactly what is wrong when they pop up.

```
name <- "JP"
#name <- JP
```

2.4 Using functions



function_name(arg1 = val1, arg2 = val2)

equivalent: function_name(arg2 = val2, arg1 = val1)

equivalent: function_name(val1, val2)

NOT equivalent: function_name(val2, val1)

You PASS values to function arguments in parentheses after its (CASE SENSITIVE) name.

R knows what values correspond to what arguments by their order, or if you specify using names and =

As an example, let’s try the seq() function, which creates a sequence of numbers.

```
seq(from = 1, to = 10, by = 1)
```

```
 [1]  1  2  3  4  5  6  7  8  9 10
```

```
#or

seq(1, 10, 1)
```

```
 [1]  1  2  3  4  5  6  7  8  9 10
```

```
#or

seq(1, 10)
```

```
 [1]  1  2  3  4  5  6  7  8  9 10
```

```
#what does this do
seq(10,1)
```

```
 [1] 10  9  8  7  6  5  4  3  2  1
```

2.5 Read in some data.

For the following demonstration we will use the RBI data from a sample of USGS gages we used last class. First we will load the tidyverse library, everything we have done so far is in base R.

Important: read_csv() is the tidyverse csv reading function, the base R function is read.csv(). read.csv() will not read your data in as a tibble, which is the format used by tidyverse functions.

```
library(tidyverse)
```

```
-- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
v dplyr     1.1.2     v readr     2.1.4
v forcats   1.0.0     v stringr   1.5.0
v ggplot2   3.4.2     v tibble    3.2.1
v lubridate 1.9.2     v tidyr     1.3.0
v purrr     1.0.1
-- Conflicts ------------------------------------------ tidyverse_conflicts() --
x dplyr::filter() masks stats::filter()
x dplyr::lag()    masks stats::lag()
i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to becom
```

```
rbi <- read_csv("Flashy_Dat_Subset.csv")
```

```
Rows: 49 Columns: 26
-- Column specification --------------------------------------------------------
Delimiter: ","
chr  (4): STANAME, STATE, CLASS, AGGECOREGION
dbl (22): site_no, RBI, RBIrank, DRAIN_SQKM, HUC02, LAT_GAGE, LNG_GAGE, PPTA...

i Use `spec()` to retrieve the full column specification for this data.
i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

2.6 Wait, hold up. What is a tibble?

Good question. It's a fancy way to store data that works well with tidyverse functions. Let's look at the rbi tibble.

```
head(rbi)
```

```
# A tibble: 6 x 26
  site_no    RBI RBIrank STANAME  DRAIN_SQKM HUC02 LAT_GAGE LNG_GAGE STATE CLASS
    <dbl>  <dbl>   <dbl> <chr>         <dbl> <dbl>    <dbl>    <dbl> <chr> <chr>
1 1013500 0.0584      35 Fish Ri~     2253.      1     47.2    -68.6 ME    Ref
2 1021480 0.208      300 Old Str~       76.7     1     44.9    -67.7 ME    Ref
3 1022500 0.198      286 Narragu~      574.      1     44.6    -67.9 ME    Ref
4 1029200 0.132      183 Seboeis~      445.      1     46.1    -68.6 ME    Ref
5 1030500 0.114      147 Mattawa~     3676.      1     45.5    -68.3 ME    Ref
6 1031300 0.297      489 Piscata~      304.      1     45.3    -69.6 ME    Ref
# i 16 more variables: AGGECOREGION <chr>, PPTAVG_BASIN <dbl>,
#   PPTAVG_SITE <dbl>, T_AVG_BASIN <dbl>, T_AVG_SITE <dbl>, T_MAX_BASIN <dbl>,
#   T_MAXSTD_BASIN <dbl>, T_MAX_SITE <dbl>, T_MIN_BASIN <dbl>,
#   T_MINSTD_BASIN <dbl>, T_MIN_SITE <dbl>, PET <dbl>, SNOW_PCT_PRECIP <dbl>,
#   PRECIP_SEAS_IND <dbl>, FLOWYRS_1990_2009 <dbl>, wy00_09 <dbl>
```

Now read in the same data with read.csv() which will NOT read the data as a tibble. How is it different? Output each one in the Console.

Knowing the data type for each column is super helpful for a few reasons.... let's talk about them.

Types: int, dbl, fctr, char, logical

```
rbi_NT <- read.csv("Flashy_Dat_Subset.csv")
```

```r
head(rbi_NT)
```

```
  site_no        RBI RBIrank                                   STANAME
1 1013500 0.05837454      35            Fish River near Fort Kent, Maine
2 1021480 0.20797008     300              Old Stream near Wesley, Maine
3 1022500 0.19805382     286     Narraguagus River at Cherryfield, Maine
4 1029200 0.13151299     183           Seboeis River near Shin Pond, Maine
5 1030500 0.11350485     147 Mattawamkeag River near Mattawamkeag, Maine
6 1031300 0.29718786     489       Piscataquis River at Blanchard, Maine
  DRAIN_SQKM HUC02 LAT_GAGE  LNG_GAGE STATE CLASS AGGECOREGION PPTAVG_BASIN
1     2252.7     1 47.23739 -68.58264    ME   Ref    NorthEast        97.42
2       76.7     1 44.93694 -67.73611    ME   Ref    NorthEast       115.39
3      573.6     1 44.60797 -67.93524    ME   Ref    NorthEast       120.07
4      444.9     1 46.14306 -68.63361    ME   Ref    NorthEast       102.19
5     3676.2     1 45.50097 -68.30596    ME   Ref    NorthEast       108.19
6      304.4     1 45.26722 -69.58389    ME   Ref    NorthEast       119.83
  PPTAVG_SITE T_AVG_BASIN T_AVG_SITE T_MAX_BASIN T_MAXSTD_BASIN T_MAX_SITE
1       93.53        3.00        3.0        9.67          0.202       10.0
2      117.13        5.71        5.8       11.70          0.131       11.9
3      129.56        5.95        6.3       11.90          0.344       12.2
4      103.24        3.61        4.0        9.88          0.231       10.4
5      113.13        4.82        5.4       10.75          0.554       11.7
6      120.93        3.60        4.2        9.57          0.431       11.0
  T_MIN_BASIN T_MINSTD_BASIN T_MIN_SITE   PET SNOW_PCT_PRECIP PRECIP_SEAS_IND
1       -2.49          0.269       -2.7 504.7            36.9           0.102
2       -0.85          0.123       -0.6 554.2            39.5           0.046
3        0.06          0.873        1.4 553.1            38.2           0.047
4       -2.13          0.216       -1.5 513.0            36.4           0.070
5       -1.49          0.251       -1.2 540.8            37.2           0.033
6       -2.46          0.268       -1.7 495.8            40.2           0.030
  FLOWYRS_1990_2009 wy00_09
1                20      10
2                11      10
3                20      10
4                11      10
5                20      10
6                13      10
```

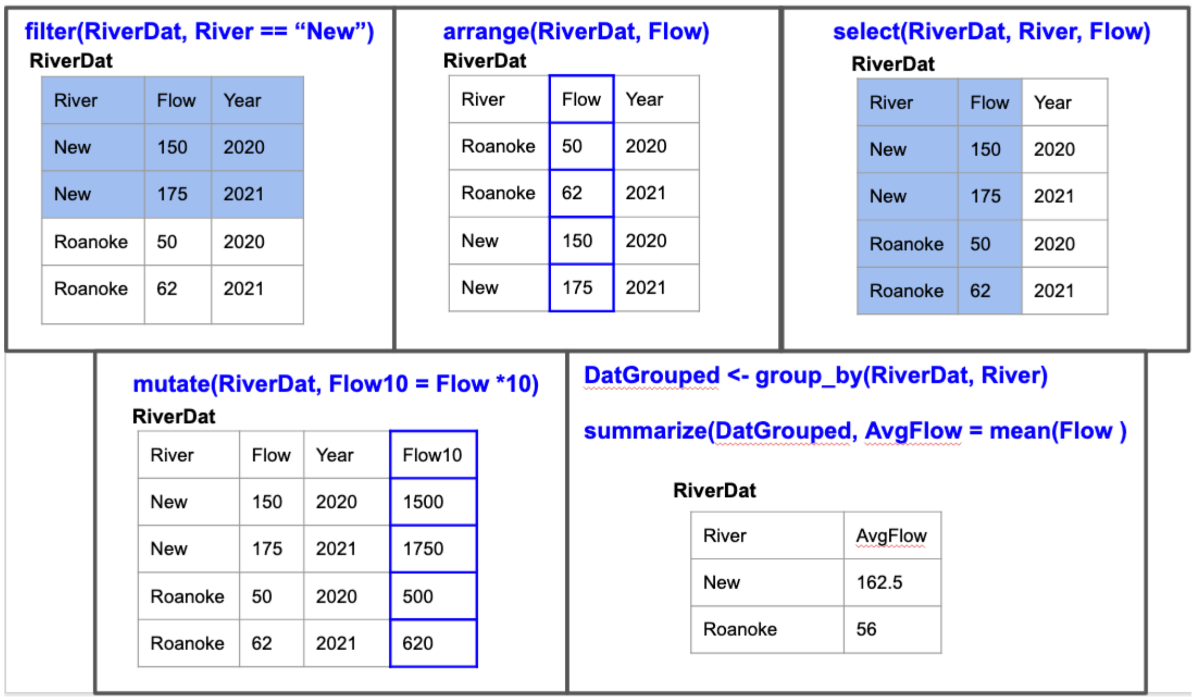
2.7 Data wrangling in dplyr

If you forget syntax or what the following functions do, here is an excellent cheat sheet: https://rstudio.com/wp-content/uploads/2015/02/data-wrangling-cheatsheet.pdf

We will demo five functions below:

- **filter()** - returns rows that meet specified conditions
- **arrange()** - reorders rows
- **select()** - pull out variables (columns)
- **mutate()** - create new variables (columns) or reformat existing ones
- **summarize()** - collapse groups of values into summary stats

With all of these, the first argument is the data and then the arguments after that specify what you want the function to do.



filter(RiverDat, River == "New")

RiverDat

River	Flow	Year
New	150	2020
New	175	2021
Roanoke	50	2020
Roanoke	62	2021

arrange(RiverDat, Flow)

RiverDat

River	Flow	Year
Roanoke	50	2020
Roanoke	62	2021
New	150	2020
New	175	2021

select(RiverDat, River, Flow)

RiverDat

River	Flow	Year
New	150	2020
New	175	2021
Roanoke	50	2020
Roanoke	62	2021

mutate(RiverDat, Flow10 = Flow *10)

RiverDat

River	Flow	Year	Flow10
New	150	2020	1500
New	175	2021	1750
Roanoke	50	2020	500
Roanoke	62	2021	620

DatGrouped <- group_by(RiverDat, River)

summarize(DatGrouped, AvgFlow = mean(Flow)

RiverDat

River	AvgFlow
New	162.5
Roanoke	56

2.8 Filter

Write an expression that returns data in rbi for the state of Maine (ME)

Operators:
== equal
!= not equal

>= , <= greater than or equal to, less than or equal to
>, < greater than or less then
%in% included in a list of values
& and
| or

```
filter(rbi, STATE == "ME")
```

```
# A tibble: 13 x 26
   site_no    RBI RBIrank STANAME DRAIN_SQKM HUC02 LAT_GAGE LNG_GAGE STATE CLASS
     <dbl>  <dbl>   <dbl> <chr>        <dbl> <dbl>    <dbl>    <dbl> <chr> <chr>
 1 1013500 0.0584      35 Fish R~     2253.      1     47.2    -68.6 ME    Ref
 2 1021480 0.208      300 Old St~       76.7     1     44.9    -67.7 ME    Ref
 3 1022500 0.198      286 Narrag~      574.      1     44.6    -67.9 ME    Ref
 4 1029200 0.132      183 Seboei~      445.      1     46.1    -68.6 ME    Ref
 5 1030500 0.114      147 Mattaw~     3676.      1     45.5    -68.3 ME    Ref
 6 1031300 0.297      489 Piscat~      304.      1     45.3    -69.6 ME    Ref
 7 1031500 0.320      545 Piscat~      769       1     45.2    -69.3 ME    Ref
 8 1037380 0.318      537 Ducktr~       39       1     44.3    -69.1 ME    Ref
 9 1044550 0.242      360 Spence~      500.      1     45.3    -70.2 ME    Ref
10 1047000 0.344      608 Carrab~      909.      1     44.9    -70.0 ME    Ref
11 1054200 0.492      805 Wild R~      181       1     44.4    -71.0 ME    Ref
12 1055000 0.450      762 Swift ~      251.      1     44.6    -70.6 ME    Ref
13 1057000 0.326      561 Little~      191.      1     44.3    -70.5 ME    Ref
# i 16 more variables: AGGECOREGION <chr>, PPTAVG_BASIN <dbl>,
#   PPTAVG_SITE <dbl>, T_AVG_BASIN <dbl>, T_AVG_SITE <dbl>, T_MAX_BASIN <dbl>,
#   T_MAXSTD_BASIN <dbl>, T_MAX_SITE <dbl>, T_MIN_BASIN <dbl>,
#   T_MINSTD_BASIN <dbl>, T_MIN_SITE <dbl>, PET <dbl>, SNOW_PCT_PRECIP <dbl>,
#   PRECIP_SEAS_IND <dbl>, FLOWYRS_1990_2009 <dbl>, wy00_09 <dbl>
```

2.8.1 Multiple conditions

How many gages are there in Maine with an rbi greater than 0.25

```
filter(rbi, STATE == "ME" & RBI > 0.25)
```

```
# A tibble: 7 x 26
  site_no   RBI RBIrank STANAME   DRAIN_SQKM HUC02 LAT_GAGE LNG_GAGE STATE CLASS
    <dbl> <dbl>   <dbl> <chr>          <dbl> <dbl>    <dbl>    <dbl> <chr> <chr>
1 1031300 0.297     489 Piscataq~       304.     1     45.3    -69.6 ME    Ref
```

```
2 1031500 0.320     545 Piscataq~       769       1     45.2    -69.3 ME    Ref  
3 1037380 0.318     537 Ducktrap~        39       1     44.3    -69.1 ME    Ref  
4 1047000 0.344     608 Carrabas~       909.      1     44.9    -70.0 ME    Ref  
5 1054200 0.492     805 Wild Riv~       181       1     44.4    -71.0 ME    Ref  
6 1055000 0.450     762 Swift Ri~       251.      1     44.6    -70.6 ME    Ref  
7 1057000 0.326     561 Little A~       191.      1     44.3    -70.5 ME    Ref  
# i 16 more variables: AGGECOREGION <chr>, PPTAVG_BASIN <dbl>,
#   PPTAVG_SITE <dbl>, T_AVG_BASIN <dbl>, T_AVG_SITE <dbl>, T_MAX_BASIN <dbl>,
#   T_MAXSTD_BASIN <dbl>, T_MAX_SITE <dbl>, T_MIN_BASIN <dbl>,
#   T_MINSTD_BASIN <dbl>, T_MIN_SITE <dbl>, PET <dbl>, SNOW_PCT_PRECIP <dbl>,
#   PRECIP_SEAS_IND <dbl>, FLOWYRS_1990_2009 <dbl>, wy00_09 <dbl>
```

2.9 Arrange

Arrange sorts by a column in your dataset.

Sort the rbi data by the RBI column in ascending and then descending order

```r
arrange(rbi, RBI)
```

```
# A tibble: 49 x 26
   site_no    RBI RBIrank STANAME DRAIN_SQKM HUC02 LAT_GAGE LNG_GAGE STATE CLASS
     <dbl>  <dbl>   <dbl> <chr>        <dbl> <dbl>    <dbl>    <dbl> <chr> <chr>
 1 1305500 0.0464      18 SWAN R~       21.3     2     40.8    -73.0 NY    Non-~
 2 1013500 0.0584      35 Fish R~     2253.      1     47.2    -68.6 ME    Ref  
 3 1306460 0.0587      37 CONNET~       55.7     2     40.8    -73.2 NY    Non-~
 4 1030500 0.114      147 Mattaw~     3676.      1     45.5    -68.3 ME    Ref  
 5 1029200 0.132      183 Seboei~      445.      1     46.1    -68.6 ME    Ref  
 6 1117468 0.172      244 BEAVER~       25.3     1     41.5    -71.6 RI    Ref  
 7 1022500 0.198      286 Narrag~      574.      1     44.6    -67.9 ME    Ref  
 8 1021480 0.208      300 Old St~       76.7     1     44.9    -67.7 ME    Ref  
 9 1162500 0.213      311 PRIEST~       49.7     1     42.7    -72.1 MA    Ref  
10 1117370 0.230      338 QUEEN ~       50.5     1     41.5    -71.6 RI    Ref  
# i 39 more rows
# i 16 more variables: AGGECOREGION <chr>, PPTAVG_BASIN <dbl>,
#   PPTAVG_SITE <dbl>, T_AVG_BASIN <dbl>, T_AVG_SITE <dbl>, T_MAX_BASIN <dbl>,
#   T_MAXSTD_BASIN <dbl>, T_MAX_SITE <dbl>, T_MIN_BASIN <dbl>,
#   T_MINSTD_BASIN <dbl>, T_MIN_SITE <dbl>, PET <dbl>, SNOW_PCT_PRECIP <dbl>,
#   PRECIP_SEAS_IND <dbl>, FLOWYRS_1990_2009 <dbl>, wy00_09 <dbl>
```

```r
arrange(rbi, desc(RBI))
```

```
# A tibble: 49 x 26
   site_no   RBI RBIrank STANAME  DRAIN_SQKM HUC02 LAT_GAGE LNG_GAGE STATE CLASS
     <dbl> <dbl>   <dbl> <chr>         <dbl> <dbl>    <dbl>    <dbl> <chr> <chr>
 1 1311500 0.856    1017 VALLEY ~       18.1     2     40.7    -73.7 NY    Non-~
 2 1054200 0.492     805 Wild Ri~      181       1     44.4    -71.0 ME    Ref
 3 1187300 0.487     800 HUBBARD~       53.9     1     42.0    -72.9 MA    Ref
 4 1105600 0.484     797 OLD SWA~       12.7     1     42.2    -70.9 MA    Non-~
 5 1055000 0.450     762 Swift R~      251.      1     44.6    -70.6 ME    Ref
 6 1195100 0.430     744 INDIAN ~       14.8     1     41.3    -72.5 CT    Ref
 7 1181000 0.420     732 WEST BR~      244.      1     42.2    -72.9 MA    Ref
 8 1350000 0.414     721 SCHOHAR~      612.      2     42.3    -74.4 NY    Ref
 9 1121000 0.404     710 MOUNT H~       70.3     1     41.8    -72.2 CT    Ref
10 1169000 0.395     688 NORTH R~      231.      1     42.6    -72.7 MA    Ref
# i 39 more rows
# i 16 more variables: AGGECOREGION <chr>, PPTAVG_BASIN <dbl>,
#   PPTAVG_SITE <dbl>, T_AVG_BASIN <dbl>, T_AVG_SITE <dbl>, T_MAX_BASIN <dbl>,
#   T_MAXSTD_BASIN <dbl>, T_MAX_SITE <dbl>, T_MIN_BASIN <dbl>,
#   T_MINSTD_BASIN <dbl>, T_MIN_SITE <dbl>, PET <dbl>, SNOW_PCT_PRECIP <dbl>,
#   PRECIP_SEAS_IND <dbl>, FLOWYRS_1990_2009 <dbl>, wy00_09 <dbl>
```

2.10 Select

There are too many columns! You will often want to do this when you are manipulating the structure of your data and need to trim it down to only include what you will use.

Select Site name, state, and RBI from the rbi data

Note they come back in the order you put them in in the function, not the order they were in in the original data.

You can do a lot more with select, especially when you need to select a bunch of columns but don't want to type them all out. But we don't need to cover all that today. For a taste though, if you want to select a group of columns you can specify the first and last with a colon in between (first:last) and it'll return all of them. Select the rbi columns from site_no to DRAIN_SQKM.

```r
select(rbi, STANAME, STATE, RBI)
```

```
# A tibble: 49 x 3
   STANAME                                      STATE    RBI
   <chr>                                        <chr>  <dbl>
 1 Fish River near Fort Kent, Maine             ME    0.0584
 2 Old Stream near Wesley, Maine                ME    0.208
 3 Narraguagus River at Cherryfield, Maine      ME    0.198
 4 Seboeis River near Shin Pond, Maine          ME    0.132
 5 Mattawamkeag River near Mattawamkeag, Maine  ME    0.114
 6 Piscataquis River at Blanchard, Maine        ME    0.297
 7 Piscataquis River near Dover-Foxcroft, Maine ME    0.320
 8 Ducktrap River near Lincolnville, Maine      ME    0.318
 9 Spencer Stream near Grand Falls, Maine       ME    0.242
10 Carrabassett River near North Anson, Maine   ME    0.344
# i 39 more rows
```

```
select(rbi, site_no:DRAIN_SQKM)
```

```
# A tibble: 49 x 5
   site_no    RBI RBIrank STANAME                                     DRAIN_SQKM
     <dbl>  <dbl>   <dbl> <chr>                                            <dbl>
 1 1013500 0.0584      35 Fish River near Fort Kent, Maine                2253.
 2 1021480 0.208      300 Old Stream near Wesley, Maine                     76.7
 3 1022500 0.198      286 Narraguagus River at Cherryfield, Maine          574.
 4 1029200 0.132      183 Seboeis River near Shin Pond, Maine              445.
 5 1030500 0.114      147 Mattawamkeag River near Mattawamkeag, Maine     3676.
 6 1031300 0.297      489 Piscataquis River at Blanchard, Maine            304.
 7 1031500 0.320      545 Piscataquis River near Dover-Foxcroft, Mai~      769
 8 1037380 0.318      537 Ducktrap River near Lincolnville, Maine           39
 9 1044550 0.242      360 Spencer Stream near Grand Falls, Maine           500.
10 1047000 0.344      608 Carrabassett River near North Anson, Maine       909.
# i 39 more rows
```

2.11 Mutate

Use mutate to add new columns based on additional ones. Common uses are to create a column of data in different units, or to calculate something based on two columns. You can also use it to just update a column, by naming the new column the same as the original one (but be careful because you'll lose the original one!). I commonly use this when I am changing the datatype of a column, say from a character to a factor or a string to a date.

Create a new column in rbi called T_RANGE by subtracting T_MIN_SITE from T_MAX_SITE

```
mutate(rbi, T_RANGE = T_MAX_SITE - T_MIN_SITE)
```

```
# A tibble: 49 x 27
   site_no    RBI RBIrank STANAME DRAIN_SQKM HUC02 LAT_GAGE LNG_GAGE STATE CLASS
     <dbl>  <dbl>   <dbl> <chr>        <dbl> <dbl>    <dbl>    <dbl> <chr> <chr>
 1 1013500 0.0584      35 Fish R~     2253.      1     47.2    -68.6 ME    Ref
 2 1021480 0.208      300 Old St~       76.7     1     44.9    -67.7 ME    Ref
 3 1022500 0.198      286 Narrag~      574.      1     44.6    -67.9 ME    Ref
 4 1029200 0.132      183 Seboei~      445.      1     46.1    -68.6 ME    Ref
 5 1030500 0.114      147 Mattaw~     3676.      1     45.5    -68.3 ME    Ref
 6 1031300 0.297      489 Piscat~      304.      1     45.3    -69.6 ME    Ref
 7 1031500 0.320      545 Piscat~      769       1     45.2    -69.3 ME    Ref
 8 1037380 0.318      537 Ducktr~       39       1     44.3    -69.1 ME    Ref
 9 1044550 0.242      360 Spence~      500.      1     45.3    -70.2 ME    Ref
10 1047000 0.344      608 Carrab~      909.      1     44.9    -70.0 ME    Ref
# i 39 more rows
# i 17 more variables: AGGECOREGION <chr>, PPTAVG_BASIN <dbl>,
#   PPTAVG_SITE <dbl>, T_AVG_BASIN <dbl>, T_AVG_SITE <dbl>, T_MAX_BASIN <dbl>,
#   T_MAXSTD_BASIN <dbl>, T_MAX_SITE <dbl>, T_MIN_BASIN <dbl>,
#   T_MINSTD_BASIN <dbl>, T_MIN_SITE <dbl>, PET <dbl>, SNOW_PCT_PRECIP <dbl>,
#   PRECIP_SEAS_IND <dbl>, FLOWYRS_1990_2009 <dbl>, wy00_09 <dbl>,
#   T_RANGE <dbl>
```

When downloading data from the USGS through R, you have to enter the gage ID as a character, even though they are all made up of numbers. So to practice doing this, update the site_no column to be a character datatype

```
mutate(rbi, site_no = as.character(site_no))
```

```
# A tibble: 49 x 26
   site_no    RBI RBIrank STANAME DRAIN_SQKM HUC02 LAT_GAGE LNG_GAGE STATE CLASS
   <chr>    <dbl>   <dbl> <chr>        <dbl> <dbl>    <dbl>    <dbl> <chr> <chr>
 1 1013500 0.0584      35 Fish R~     2253.      1     47.2    -68.6 ME    Ref
 2 1021480 0.208      300 Old St~       76.7     1     44.9    -67.7 ME    Ref
 3 1022500 0.198      286 Narrag~      574.      1     44.6    -67.9 ME    Ref
 4 1029200 0.132      183 Seboei~      445.      1     46.1    -68.6 ME    Ref
 5 1030500 0.114      147 Mattaw~     3676.      1     45.5    -68.3 ME    Ref
 6 1031300 0.297      489 Piscat~      304.      1     45.3    -69.6 ME    Ref
 7 1031500 0.320       545 Piscat~      769        1      45.2    -69.3 ME    Ref
 8 1037380 0.318       537 Ducktr~       39        1      44.3    -69.1 ME    Ref
 9 1044550 0.242       360 Spence~      500.       1      45.3    -70.2 ME    Ref
10 1047000 0.344       608 Carrab~      909.       1      44.9    -70.0 ME    Ref
# i 39 more rows
# i 16 more variables: AGGECOREGION <chr>, PPTAVG_BASIN <dbl>,
#   PPTAVG_SITE <dbl>, T_AVG_BASIN <dbl>, T_AVG_SITE <dbl>, T_MAX_BASIN <dbl>,
#   T_MAXSTD_BASIN <dbl>, T_MAX_SITE <dbl>, T_MIN_BASIN <dbl>,
#   T_MINSTD_BASIN <dbl>, T_MIN_SITE <dbl>, PET <dbl>, SNOW_PCT_PRECIP <dbl>,
#   PRECIP_SEAS_IND <dbl>, FLOWYRS_1990_2009 <dbl>, wy00_09 <dbl>
```

2.12 Summarize

Summarize will perform an operation on all of your data, or groups if you assign groups.

Use summarize to compute the mean, min, and max rbi

```
summarize(rbi, meanrbi = mean(RBI), maxrbi = max(RBI), minrbi = min(RBI))
```

```
# A tibble: 1 x 3
  meanrbi maxrbi minrbi
    <dbl>  <dbl>  <dbl>
1   0.316  0.856 0.0464
```

Now use the group function to group by state and then summarize in the same way as above

```
rbistate <- group_by(rbi, STATE)
summarize(rbistate, meanrbi = mean(RBI), maxrbi = max(RBI), minrbi = min(RBI))
```

```
# A tibble: 7 x 4
  STATE meanrbi maxrbi minrbi
  <chr>   <dbl>  <dbl>  <dbl>
1 CT      0.366  0.430 0.295
2 MA      0.367  0.487 0.213
3 ME      0.269  0.492 0.0584
4 NH      0.336  0.368 0.265
5 NY      0.342  0.856 0.0464
6 RI      0.201  0.230 0.172
7 VT      0.299  0.365 0.231
```

2.13 Multiple operations with pipes

The pipe operator %>% allows you to perform multiple operations in a sequence without saving intermediate steps. Not only is this more efficient, but structuring operations with pipes is also more intuitive than nesting functions within functions (the other way you can do multiple operations).

2.13.1 Let's say we want to tell R to make a PB&J sandwich by using the pbbread(), jbread(), and joinslices() functions and the data "ingredients". If we do this saving each step if would look like this:

> sando <- pbbread(ingredients)
>
> sando <- jbread(sando)
>
> sando <- joinslices(sando)

2.13.2 If we nest the functions together we get this

> joinslice(jbread(pbbread(ingredients)))

Efficient… but tough to read/interpret

2.13.3 Using the pipe it would look like this

> ingredients %>%
> pbbread() %>%
> jbread() %>%
> joinslices()

Much easier to follow!

2.13.4 When you use the pipe, it basically takes whatever came out of the first function and puts it into the data argument for the next one

so rbi %>% group_by(STATE) is the same as group_by(rbi, STATE)

Take the groupby and summarize code from above and perform the operation using the pipe

```
rbi %>%
  group_by(STATE) %>%
  summarize(meanrbi = mean(RBI), maxrbi = max(RBI), minrbi = min(RBI))
```

```
# A tibble: 7 x 4
  STATE meanrbi maxrbi minrbi
  <chr>   <dbl>  <dbl>  <dbl>
1 CT      0.366  0.430 0.295
2 MA      0.367  0.487 0.213
3 ME      0.269  0.492 0.0584
4 NH      0.336  0.368 0.265
5 NY      0.342  0.856 0.0464
6 RI      0.201  0.230 0.172
7 VT      0.299  0.365 0.231
```

2.14 Save your results to a new tibble

We have just been writing everything to the screen so we can see what we are doing… In order to save anything we do with these functions to work with it later, we just have to use the assignment operator (<-) to store the data.

One kind of awesome thing about the assignment operator is that it works both ways…

x <- 3 and 3 -> x do the same thing (WHAT?!)

So you can do the assignment at the beginning of the end of your dplyr workings, whatever you like best.

Use the assignment operator to save the summary table you just made.

```
stateRBIs <- rbi %>%
  group_by(STATE) %>%
  summarize(meanrbi = mean(RBI), maxrbi = max(RBI), minrbi = min(RBI))

# Notice when you do this it doesn't output the result...
# You can see what you did by clickon in stateRBIs in your environment panel
# or just type stateRBIs

stateRBIs
```

```
# A tibble: 7 x 4
  STATE meanrbi maxrbi minrbi
  <chr>   <dbl>  <dbl>  <dbl>
```

```
1 CT       0.366  0.430 0.295
2 MA       0.367  0.487 0.213
3 ME       0.269  0.492 0.0584
4 NH       0.336  0.368 0.265
5 NY       0.342  0.856 0.0464
6 RI       0.201  0.230 0.172
7 VT       0.299  0.365 0.231
```

2.15 What about NAs?

We will talk more about this when we discuss stats, but some operations will fail if there are NA's in the data. If appropriate, you can tell functions like mean() to ignore NAs. You can also use drop_na() if you're working with a tibble. But be aware if you use that and save the result, drop_na() gets rid of the whole row, not just the NA. Because what would you replace it with…. an NA?

```r
x <- c(1,2,3,4,NA)
mean(x, na.rm = TRUE)
```

```
[1] 2.5
```

2.16 What are some things you think I'll ask you to do for the activity next class?