Modelling Public Water Quality Data

Yezzen K

April 28, 2023

Contents

1 Why public datasets?

Working with large, open-access data sets can serve many purposes. It can be an excellent way to explore new ideas, before investing in field-work or experiments. It can be a great way to take local or experimental results and expand them to different ecosystems, places, or landscapes. Or it can be an excellent way to build, validate, and test ecological models on regional or national scales.

So why doesn't everyone use public data? Well, it's often collected by a variety of organizations, with different methods, units, and innconsistent metadata. Together these issues make large public data sets "messy." Messy data can be messy in many different ways, but at the basic level it means that data is hard to analyze; not because the data itself is bad, but because the way it is organized is unclear or inconsistent.

In this lab, we will learn some tricks to "tidying" data, making it analysis-ready. We will depend heavily on the tidyverse, an excellent series of packages that make data manipulation beautiful and easy. We will also be working with water quality portal data so we will also use the excellent dataRetrieval package for downloading data from the Water Quality Portal and the USGS.

1.1 Loading key packages

This lesson is meant to introduce the incredible variety of tools that one can use to clean data, many of these tools are captured by the `tidyverse` meta-package, a package of packages, but there are some additional ones that will help us locate our various water quality sites.

```r
library(tidyverse) #Package with dplyr, tibble, readr, and others to help clean coding
library(dataRetrieval) #Package to download data.
library(sf) #Geospatial package to plot and explore data
library(mapview) #Simple interface to leaflet interactive maps
library(broom) #Simplifies model outputs
library(knitr) #Makes nice tables
library(kableExtra) #Makes even nicer tables
library(lubridate) #Makes working with dates easier
library(ggthemes) #Makes plots prettier
library(tidyr) #Makes multiple simultaneous models easier

#Move the directory to the top folder level; DONT RUN THIS!!
#knitr::opts_knit$set(root.dir='..')
```

2 Downloading data

For this lab, we'll explore water quality data in the Colorado River basin as it moves from Colorado to Arizona. All data will be generated through the code you see below, with the only external information coming from knowing the SiteID's for the monitoring locations along the Colorado River and the water quality characteristic names.

The water quality portal can be accessed with the command `readWQPdata`, which takes a variety of parameters (like startdate, enddate, constituents, etc…). We'll generate these rules for downloading the data here.

2.1 Download prep

```r
# First we'll make a tibble (a tidyverse table) with Site IDs. Generally these are increasingly downstr
colorado <- tibble(sites=c('USGS-09034500', 'USGS-09069000',
                           'USGS-09085000', 'USGS-09095500', 'USGS-09152500'),
                   basin=c('colorado1', 'eagle',
                           'roaring', 'colorado3', 'gunnison'))

# Now we need to setup a series of rules for downloading data from the Water Quality Portal.
# We'll focus on cation and anion data from 1950-present. Each cation has a name that we might
# typically use like calcium or sulfate, but the name may be different in the water quality
# portal, so we have to check this website https://www.waterqualitydata.us/Codes/Characteristicname?mim
# to get our names correct.

paramater.names <- c('ca', 'mg', 'na', 'k', 'so4', 'cl', 'hco3')

ca <- 'Calcium'
mg <- 'Magnesium'
na <- 'Sodium'
k <- 'Potassium'
so4 <- c('Sulfate', 'Sulfate as SO4', 'Sulfur Sulfate', 'Total Sulfate')
cl <- 'Chloride'
hco3 <- c('Alkalinity, bicarbonate', 'Bicarbonate')

# Compile all these names into a single list
parameters <- list(ca, mg, na, k, so4, cl, hco3)

# Name each cation or anion in the list
names(parameters) <- paramater.names

# Notice that we aren't downloading any nutrients (P or N) because they are much messier (100s of diffe
# data) than other cation anion data.

# Start dates
start <- '1980-10-01'
end <- '2023-01-01'

# Sample media (no sediment samples)
sampleMedia = 'Water'

# Compile all this information into a list with arguments
site.args <- list(siteid = colorado$sites,
                  sampleMedia = sampleMedia,
                  startDateLo = start,
                  startDateHi = end,
                  characteristicName = NA) # We'll fill this in later in a loop
```

2.2 Concentration data download

Now that we have generated the commands to download the data, the code to download the data is here, but it is not run on purpose because it takes 15 minutes or so to run every time. You can always run it yourself by setting `eval = T`.

```
conc.list <- list() # Empty list to hold each data download

# We'll loop over each anion or cation and download all data at our sites for that constituent
for(i in 1:length(parameters)){

  # We need to rename the characteristicName (constituent) each time we go through the loop
  site.args$characteristicName <- parameters[[i]]

  # readWQPdata takes in our site.args list and downloads the data according to those rules
  # time, constituent, site, etc...

  # Don't forget about pipes "%>%"! Pipes pass forward the results of a previous command, so that
  # you don't have to constantly rename variables. I love them.

  conc.list[[i]] <- readWQPdata(site.args) %>%
    mutate(parameter = names(parameters)[i]) #Mutate just adds a new column to the data frame

  # Pipes make the above command simple and succinct versus something more complicated like:
  # conc.list[[i]] <- readWQPdata(site.args)
  # conc.list[[i]]$parameter <- names(parameters)[i]

}

conc.long <- conc.list %>% bind_rows()
```

3 Data tidying

Now that we have downloaded the data, we need to tidy it up. The water quality portal data comes with an incredible amount of metadata in the form of extra columns, but we don't need all this extra data.

Look at the data you downloaded:

```
#
head(conc.long) %>%
  kable(.,'html') %>%
  kable_styling() %>%
  scroll_box(width = '800px',height = '300px')
```

OrganizationIdentifier

OrganizationFormalName

ActivityIdentifier

ActivityTypeCode

ActivityMediaName

ActivityMediaSubdivisionName

ActivityStartDate

ActivityStartTime.Time

ActivityStartTime.TimeZoneCode

ActivityEndDate

ActivityEndTime.Time

ActivityEndTime.TimeZoneCode

ActivityDepthHeightMeasure.MeasureValue

ActivityDepthHeightMeasure.MeasureUnitCode

ActivityDepthAltitudeReferencePointText

ActivityTopDepthHeightMeasure.MeasureValue

ActivityTopDepthHeightMeasure.MeasureUnitCode

ActivityBottomDepthHeightMeasure.MeasureValue

ActivityBottomDepthHeightMeasure.MeasureUnitCode

ProjectIdentifier

ActivityConductingOrganizationText

MonitoringLocationIdentifier

ActivityCommentText

SampleAquifer

HydrologicCondition

HydrologicEvent

SampleCollectionMethod.MethodIdentifier

SampleCollectionMethod.MethodIdentifierContext

SampleCollectionMethod.MethodName

SampleCollectionEquipmentName

ResultDetectionConditionText

CharacteristicName

ResultSampleFractionText

ResultMeasureValue

ResultMeasure.MeasureUnitCode

MeasureQualifierCode

ResultStatusIdentifier

StatisticalBaseCode

ResultValueTypeName

ResultWeightBasisText

ResultTimeBasisText

ResultTemperatureBasisText

ResultParticleSizeBasisText

PrecisionValue

ResultCommentText

USGSPCode

ResultDepthHeightMeasure.MeasureValue

ResultDepthHeightMeasure.MeasureUnitCode

ResultDepthAltitudeReferencePointText

SubjectTaxonomicName

SampleTissueAnatomyName

ResultAnalyticalMethod.MethodIdentifier

ResultAnalyticalMethod.MethodIdentifierContext

ResultAnalyticalMethod.MethodName

MethodDescriptionText

LaboratoryName

AnalysisStartDate

ResultLaboratoryCommentText

DetectionQuantitationLimitTypeName

DetectionQuantitationLimitMeasure.MeasureValue

DetectionQuantitationLimitMeasure.MeasureUnitCode

PreparationStartDate

ProviderName

timeZoneStart

timeZoneEnd

ActivityStartDateTime

ActivityEndDateTime

parameter

USGS-CO

USGS Colorado Water Science Center

nwisco.01.01000518

Sample-Routine

Water

Surface Water

2009-11-10

15:10:00

MST

2009-11-10

15:20:59

MST

NA

NA

NA

NA

NA

NA

NA

NA

U.S. Geological Survey-Water Resources Discipline

USGS-09152500

L-3210048 X = RU not rcvd.. schedule 1 deleted, schedule 1840 added, okay per WSC, PCFF error, paa, 11/20/09

NA

Stable, normal stage

Routine sample

30

USGS parameter code 82398

Single vertical

US DH-81

NA

Calcium

Dissolved

115.0

mg/l

NA

Accepted

NA

Actual

NA

NA

NA

NA

NA

NA

00915

NA

NA

NA

NA

NA

PLA11

USGS

Metals, wf, ICP-AES (NWQL)

USGS OF 93-125, p 101

USGS-National Water Quality Lab, Denver, CO

2009-11-23

NA

Laboratory Reporting Level

0.044

mg/l

NA

NWIS

7

7

2009-11-10 22:10:00

2009-11-10 22:20:59

ca

USGS-CO

USGS Colorado Water Science Center

nwisco.01.98201921

Sample-Routine

Water

Surface Water

1982-05-21

11:35:00

MDT

NA

NA

NA

NA

NA

NA

NA

NA

NA

NA

NA

U.S. Bureau of Reclamation

USGS-09085000

NA

NA

Not determined

Routine sample

USGS

USGS

USGS

Unknown

NA

Calcium

Dissolved

51.0

mg/l

NA

Historical

NA

Actual

NA

NA

NA

NA

NA

NA

00915

NA

NA

NA

NA

NA

NA

NA

NA

NA

NA

NA

NA

NA

NA

NA

NA

NWIS

6

NA

1982-05-21 17:35:00

NA

ca

USGS-CO

USGS Colorado Water Science Center

nwisco.01.98403318

Sample-Routine

Water

Surface Water

1984-09-06

11:40:00

MDT

NA

NA

NA

NA

NA

NA

NA

NA

NA

NA

NA

U.S. Geological Survey-Water Resources Discipline

USGS-09085000

NA

NA

Stable, normal stage

Routine sample

USGS

USGS

USGS

Unknown

NA

Calcium

Dissolved

65.9

mg/l

NA

Historical

NA

Actual

NA

NA

NA

NA

NA

NA

00915

NA

NA

NA

NA

NA

NA

NA

NA

NA

NA

NA

NA

NA

NA

NA

NA

NWIS

6

NA

1984-09-06 17:40:00

NA

ca

USGS-CO

USGS Colorado Water Science Center

nwisco.01.98403810

Sample-Routine

Water

Surface Water

1984-05-30

14:00:00

MDT

NA

NA

NA

NA

NA

NA

NA

NA

NA

NA

NA

U.S. Geological Survey-Water Resources Discipline

USGS-09152500

NA

NA

Stable, high stage

Snowmelt

USGS

USGS

USGS

Unknown

NA

Calcium

Dissolved

30.0

mg/l

NA

Historical

NA

Actual

NA

NA

NA

NA

NA

NA

00915

NA

NA

NA

NA

NA

NA

NA

NA

NA

NA

NA

NA

NA

NA

NA

NA

NWIS

6

NA

1984-05-30 20:00:00

NA

ca

USGS-CO

USGS Colorado Water Science Center

nwisco.01.98301700

Sample-Routine

Water

Surface Water

1983-04-28

12:45:00

MDT

NA

NA

NA

NA

NA

NA

NA

NA

NA

NA

NA

U.S. Geological Survey-Water Resources Discipline

USGS-09085000

NA

NA

Not determined

Routine sample

USGS

USGS

USGS

Unknown

NA

Calcium

Dissolved

60.0

mg/l

NA

Historical

NA

Actual

NA

NA

NA

NA

NA

NA

00915

NA

NA

NA

NA

NA

NA

NA

NA

NA

NA

NA

NA

NA

NA

NA

NA

NWIS

6

NA

1983-04-28 18:45:00

NA

ca

USGS-CO

USGS Colorado Water Science Center

nwisco.01.98701623

Sample-Routine

Water

Surface Water

1987-06-03

12:30:00

MDT

NA

NA

NA

NA

NA

NA

NA

NA

NA

NA

NA

U.S. Geological Survey-Water Resources Discipline

USGS-09069000

NA

NA

Stable, low stage

Volcanic action

USGS

USGS

USGS

Unknown

NA

Calcium

Dissolved

28.0

mg/l

NA

Historical

NA

Actual

NA

NA

NA

NA

NA

NA

00915

NA

NA

NA

NA

NA

PLA11

USGS

Metals, wf, ICP-AES (NWQL)

USGS OF 93-125, p 101

NA

NA

NA

NA

NA

NA

NA

NWIS

6

NA

1987-06-03 18:30:00

NA

ca

3.1 Initial cleaning up

Wow, that looks messy! Lots of extraneous columns, lots of NAs, so much information we can hardly parse it. Let's pare it down to the essentials.

```
# This code mostly just grabs and renames the most important data columns
conc.clean <-  conc.long %>%
  dplyr::select(date = ActivityStartDate,
                parameter = CharacteristicName,
                units = ResultMeasure.MeasureUnitCode,
                SiteID = MonitoringLocationIdentifier,
                org = OrganizationFormalName,
                org_id = OrganizationIdentifier,
                time = ActivityStartTime.Time,
                value = ResultMeasureValue,
                sample_method = SampleCollectionMethod.MethodName,
                analytical_method = ResultAnalyticalMethod.MethodName,
                particle_size = ResultParticleSizeBasisText,
                date_time = ActivityStartDateTime,
                media = ActivityMediaName,
                sample_depth = ActivityDepthHeightMeasure.MeasureValue,
                sample_depth_unit = ActivityDepthHeightMeasure.MeasureUnitCode,
                fraction = ResultSampleFractionText,
                status = ResultStatusIdentifier) %>%
  # Remove trailing white space in labels
  mutate(units  =  trimws(units)) %>%
  # Keep only samples that are water samples
  filter(media == 'Water')
```

Now let's look at the tidier version:

```
head(conc.clean) %>%
  kable(.,'html') %>%
  kable_styling() %>%
  scroll_box(width = '800px', height = '300px')
```

date

parameter

units

SiteID

org

org_id

time

value

sample_method

analytical_method

particle_size

date_time

media

sample_depth

sample_depth_unit

fraction

status

2009-11-10

Calcium

mg/l

USGS-09152500

USGS Colorado Water Science Center

USGS-CO

15:10:00

115.0

Single vertical

Metals, wf, ICP-AES (NWQL)

NA

2009-11-10 22:10:00

Water

NA

NA

Dissolved

Accepted

1982-05-21

Calcium

mg/l

USGS-09085000

USGS Colorado Water Science Center

USGS-CO

11:35:00

51.0

USGS

NA

NA

1982-05-21 17:35:00

Water

NA

NA

Dissolved

Historical

1984-09-06

Calcium

mg/l

USGS-09085000

USGS Colorado Water Science Center

USGS-CO

11:40:00

65.9

USGS

NA

NA

1984-09-06 17:40:00

Water

NA

NA

Dissolved

Historical

1984-05-30

Calcium

mg/l

USGS-09152500

USGS Colorado Water Science Center

USGS-CO

14:00:00

30.0

USGS

NA

NA

1984-05-30 20:00:00

Water

NA

NA

Dissolved

Historical

1983-04-28

Calcium

mg/l

USGS-09085000

USGS Colorado Water Science Center

USGS-CO

12:45:00

60.0

USGS

NA

NA

1983-04-28 18:45:00

Water

NA

NA

Dissolved

Historical

1987-06-03

Calcium

mg/l

USGS-09069000

USGS Colorado Water Science Center

USGS-CO

12:30:00

28.0

USGS

Metals, wf, ICP-AES (NWQL)

NA

1987-06-03 18:30:00

Water

NA

NA

Dissolved

Historical

```r
site_info <- attr(conc.clean, 'siteInfo')
```

3.2 Final tidy dataset

Okay, that is getting better, but we still have lots of extraneous information. For our purposes, let's assume that the sample and analytical methods used by the USGS are reasonable and exchangeable (one method is equivalent to the other). If we make that assumption then the only remaining tidying step left is to make sure that all the data is in the same units.

3.2.1 Unit Check

```r
table(conc.clean$units)
```

```
##
## mg/l
## 9061
```

Wow! Almost all the data is in mg/L. That makes our job really easy.

We just need to remove these observations with a `dplyr::filter()` call and then select an even smaller subset of useful columns, while adding a time object column using the `lubridate::ymd()` call.

```r
conc.tidy <- conc.clean %>%
  filter(units == 'mg/l') %>%
  # ymd() converts characters into YYYY-MM-DD date formatting:
  mutate(date = lubridate::ymd(date)) %>%
  dplyr::select(date,
         parameter,
         SiteID,
         conc=value)
```

3.2.2 Daily data

Now we have a manageable data frame. But how do we want to organize the data? Since we are looking at a really long time-series of data (70 years), let's look at data as a daily average. The `dplyr::group_by()` and `dplyr::summarize()` commands make this really easy:

```r
# The amazing group_by function groups all the data so that the summary
# only applies to each subgroup (site, date, and parameter combination).
# So in the end you get a daily average concentratino for each site and parameter type.
conc.daily <- conc.tidy %>%
  group_by(date, parameter, SiteID) %>%
  summarize(conc = mean(conc, na.rm = T))
```

```
## `summarise()` has grouped output by 'date', 'parameter'. You can override using
## the `.groups` argument.
```

Taking daily averages looks like it eliminated 28 observations, meaning these site-date combinations had multiple observations on the same day.

4 Assignment!

Let's imagine you wanted to add data for your water quality analyses, but you also know that you need to do this analysis over and over again. Let's walk through how we would: 1) Add new data to our `conc.clean` data set, and 2) how to write a function to download, clean, and update our data with far less code.

4.1 Question 1.

Write a function that can repeat the above steps with a single function call. This function should take in a single tibble that is identical in structure to the `colorado` one above (e.g. it has columns named `sites`, and `basin`). The function should then take in that tibble and be able to download and clean the data to make the data structure/outcomes exactly like `conc.daily`. Use this function to download data for the three sites listed below.

```r
### THE FUNCTION

q1_function <- function(additional_data){
  #additional_data <- tibble(sites = c('USGS-09180000', 'USGS-09180500', 'USGS-09380000'),
                        #basin_q1 = c('dolores', 'colorado4', 'colorado5'))

    # Now we need to setup a series of rules for downloading data from the Water Quality Portal.
# We'll focus on cation and anion data from 1950-present. Each cation has a name that we might
# typically use like calcium or sulfate, but the name may be different in the water quality
# portal, so we have to check this website https://www.waterqualitydata.us/Codes/Characteristicname?mim
# to get our names correct.

paramater_names <- c('ca', 'mg', 'na', 'k', 'so4', 'cl', 'hco3')

ca <- 'Calcium'
mg <- 'Magnesium'
na <- 'Sodium'
k <- 'Potassium'
so4 <- c('Sulfate', 'Sulfate as SO4', 'Sulfur Sulfate', 'Total Sulfate')
cl <- 'Chloride'
hco3 <- c('Alkalinity, bicarbonate', 'Bicarbonate')

# Compile all these names into a single list
parameters <- list(ca, mg, na, k, so4, cl, hco3)

# Name each cation or anion in the list
names(parameters) <- paramater_names

# Notice that we aren't downloading any nutrients (P or N) because they are much messier (100s of diffe
# data) than other cation anion data.


# Start dates
start <- '1980-10-01'
end <- '2023-01-01'

# Sample media (no sediment samples)
sampleMedia = 'Water'
```

```r
# Compile all this information into a list with arguments
site_args_q1 <- list(siteid = additional_data$sites, # changed to work for current function
                  sampleMedia = sampleMedia,
                  startDateLo = start,
                  startDateHi = end,
                  characteristicName = NA) # We'll fill this in later in a loop

conc_list_q1 <- list() # Empty list to hold each data download

# We'll loop over each anion or cation and download all data at our sites for that constituent
for(i in 1:length(parameters)){

  # We need to rename the characteristicName (constituent) each time we go through the loop
  site_args_q1$characteristicName <- parameters[[i]]

  # readWQPdata takes in our site.args list and downloads the data according to those rules
  # time, constituent, site, etc...

  # Don't forget about pipes "%>%"! Pipes pass forward the results of a previous command, so that
  # you don't have to constantly rename variables. I love them.

  conc_list_q1[[i]] <- readWQPdata(site_args_q1) %>%
    mutate(parameter = names(parameters)[i]) #Mutate just adds a new column to the data frame

  # Pipes make the above command simple and succinct versus something more complicated like:
  # conc.list[[i]] <- readWQPdata(site.args)
  # conc.list[[i]]$parameter <- names(parameters)[i]

}

conc_long_q1 <- conc_list_q1 %>% bind_rows()

### DATA TIDYING - DELETE?
head(conc_long_q1) %>%
  kable(.,'html') %>%
  kable_styling() %>%
  scroll_box(width = '800px',height = '300px')

### Initial Cleaning Up

conc_clean_q1 <-  conc_long_q1 %>%
  dplyr::select(date = ActivityStartDate,
                parameter = CharacteristicName,
                units = ResultMeasure.MeasureUnitCode,
                SiteID = MonitoringLocationIdentifier,
                org = OrganizationFormalName,
                org_id = OrganizationIdentifier,
                time = ActivityStartTime.Time,
                value = ResultMeasureValue,
                sample_method = SampleCollectionMethod.MethodName,
                analytical_method = ResultAnalyticalMethod.MethodName,
                particle_size = ResultParticleSizeBasisText,
                date_time = ActivityStartDateTime,
                media = ActivityMediaName,
                sample_depth = ActivityDepthHeightMeasure.MeasureValue,
                sample_depth_unit = ActivityDepthHeightMeasure.MeasureUnitCode,
                fraction = ResultSampleFractionText,
                status = ResultStatusIdentifier) %>%
  # Remove trailing white space in labels
  mutate(units  =  trimws(units)) %>%
  # Keep only samples that are water samples
  filter(media == 'Water')

### Examine Tidier Data
head(conc_clean_q1) %>%
  kable(.,'html') %>%
  kable_styling() %>%
  scroll_box(width = '800px', height = '300px')
site_info <- attr(conc_clean_q1, 'siteInfo')

### Unit Check
table(conc_clean_q1$units)

conc_tidy_q1 <- conc_clean_q1 %>%
  filter(units == 'mg/l') %>%
  # ymd() converts characters into YYYY-MM-DD date formatting:
  mutate(date = lubridate::ymd(date)) %>%
  dplyr::select(date,
         parameter,
         SiteID,
         conc=value)

### Daily Data

conc_daily_q1 <- conc_tidy_q1 %>%
  group_by(date, parameter, SiteID) %>%
  summarize(conc = mean(conc, na.rm = T))

return(view(conc_daily_q1))

}
additional_data <- tibble(sites = c('USGS-09180000', 'USGS-09180500', 'USGS-09380000'),
                          basin_q1 = c('dolores', 'colorado4', 'colorado5'))

q1_function(additional_data)
```

```
## `summarise()` has grouped output by 'date', 'parameter'. You can override using
## the `.groups` argument.
```

4.2 Question 2.

Append the new data that the above function returned to `conc.daily` using `bind_rows()`. (Remember, this new data should be identical in structure to the `conc.daily` data set). Save this new data set as `tidied_full_wq.RData` using the `save()` function.

```
wq <- dplyr::bind_rows(conc.daily, q1_function(additional_data))
```

```
## `summarise()` has grouped output by 'date', 'parameter'. You can override using
## the `.groups` argument.
```

```
view(wq)

save(wq, file = 'tidied_full_wq.RData')
```

4.3 Question 3

We now have a dataset of stream water quality data for 9 sites throughout Colorado. However, one potential control on stream chemistry is stream discharge. One function that can allow you to easily download discharge data is `readNWISdv()` from the `dataRetrieval` package. Use this function to download daily discharge data for all eight of the sites you've already worked with above. Save this data as `data/Q.RData`. The site numbers are the same as what we used above, but you need to remove `USGS-` from each site. Reminder, discharge is `00060` for the `parameterCd` argument. Moreover, we can use `renameNWISColumns()` to automatically make the column names a little less annoying.

```
# Reminder! you can use ?readNWISdv to read about how the function works.
sites <-
  # Bind the two datasets to get all 8 sites
  bind_rows(colorado, additional_data) %>%
  # Grab just the column labeled sites
  pull(sites) %>%
  # Remove the USGS- prefix
  gsub('USGS-', '', .)

#PUT ADDITIONAL CODING STEPS HERE

# pulls USGS daily ('dv') stream flow data:
q_data <- dataRetrieval::readNWISdv(siteNumbers = c("09034500", "09069000", "09085000", "09095500", "09
                                     parameterCd = "00060", # USGS code for stream flow
                                     startDate = "1980-10-01", # YYYY-MM-DD formatting
                                     endDate = "2023-01-01") %>% # YYYY-MM-DD formatting
  rename(q_cfs = X_00060_00003) %>% # USGS code for stream flow units in cubic feet per second (CFS)
  mutate(Date = lubridate::ymd(Date)) # convert the Date column to "Date" formatting using the `lubrida
         #Site = case_when(site_no == "06752260" ~ "Lincoln",
                          #site_no == "06752280" ~ "Boxelder"))
save(q_data, file = 'Q.Rdata')
```

5 Final data prep

We have a 'tidy' data set from our previous work that includes both discharge data and concentration data. Let's look it! But first, where is the data?

```
library(tidyverse) # Package with dplyr, tibble, readr, and others to help clean coding
library(dataRetrieval) # Package to download data.
library(sf) # Geospatial package to plot and explore data
library(mapview) # Simple interface to leaflet interactive maps
```

```r
library(broom) # Simplifies model outputs
library(knitr) # Makes nice tables
library(kableExtra) # Makes even nicer tables
library(lubridate) # Makes working with dates easier
library(ggthemes) # Makes plots prettier
library(tidyr) # Makes multiple simultaneous models easier
library(trend) # Allows us to explore trends.
```

5.1 Data load

```r
load('Q.RData')
load('tidied_full_wq.RData')

## Site info so we can use names rather than long site codes
colorado <- tibble(SiteID = c('USGS-09034500', 'USGS-09069000',
                              'USGS-09085000','USGS-09095500', 'USGS-09152500'),
                   basin = c('colorado1', 'eagle',
                              'roaring', 'colorado3', 'gunnison')) %>%
  bind_rows(tibble(SiteID = c('USGS-09180000', 'USGS-09180500', 'USGS-09380000'),
                          basin = c('dolores', 'colorado4', 'colorado5'))
)

# Grab the basin name
wq <- wq %>%
  inner_join(colorado)
```

```
## Joining with `by = join_by(SiteID)`
```

5.2 Site info extraction

With all our data transformations in the previous .Rmd we lost a lot of the metadata for each site. We need to re-download this data using `whatWQPdata()`

```r
site_info <- whatWQPsites(siteid = unique(wq$SiteID)) %>%
  dplyr::select(SiteID = MonitoringLocationIdentifier,
                  name = MonitoringLocationName,
                  area = DrainageAreaMeasure.MeasureValue,
                  area.units = DrainageAreaMeasure.MeasureUnitCode,
                  elev = VerticalMeasure.MeasureValue,
                  elev_units = VerticalMeasure.MeasureUnitCode,
                  lat = LatitudeMeasure,
                  long = LongitudeMeasure) %>%
  distinct() %>%  # Distinct without any arguments just keeps the first of any duplicates rows
  # join this data to our `colorado` object to get the `basin` column:
  inner_join(colorado) # join the data to colorado
```

```
## Joining with `by = join_by(SiteID)`
```

5.2.1 Map

Here we use the `sf` package to project the site information data into a GIS type data object called a `simple feature (sf)`. The function `st_as_sf` converts the longitude (x) and latitude (y) coordinates into a projected point feature with the EPSG code 4326 (WGS 84). (See Lesson 1 and Lesson 4 for a more detailed explanation.) We can then use the `mapview` package and function to look at where these sites are.

```
# convert site info as an sf object
site_sf <- site_info %>%
  st_as_sf(.,coords = c('long', 'lat'), crs = 4326) # convert long, lat to spatial object

mapview(site_sf)
```

```
## PhantomJS not found. You can install it with webshot::install_phantomjs(). If it is installed, pleas
```

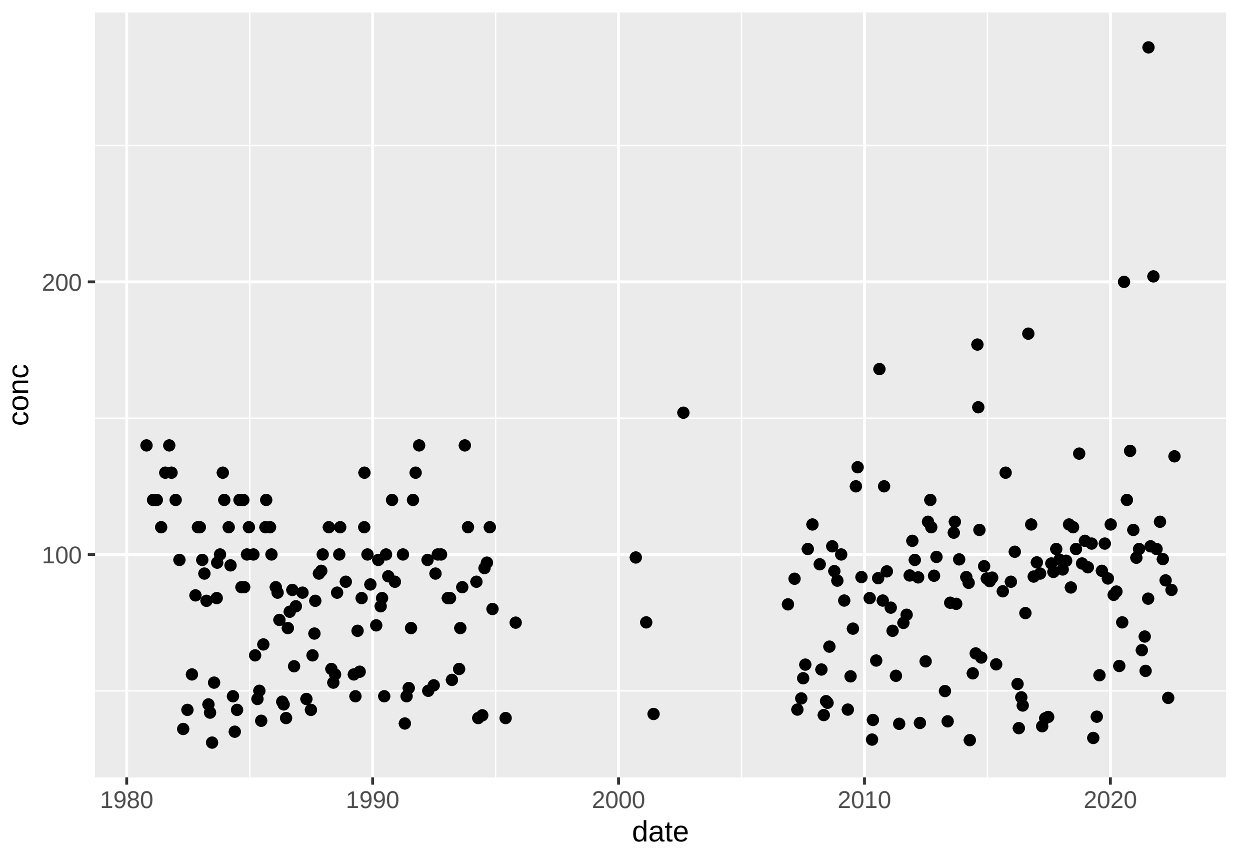
So these sites are generally in the Colorado River Basin with increasing size.

6 Modelling Data

6.1 Trend detection?

Now that we know where the data is coming from and we are happy with what it looks like, let's start modelling! The first question we might want to explore is: **Are concentrations of elements changing over time?**. Let's first focus on calcium in the Dolores River. As with all data work, the first thing you should do is look at our data.

```
dolores_ca <- wq %>%
  filter(basin == 'dolores', parameter == 'Calcium')  # is the dolores river over the years changing in

ggplot(dolores_ca, aes(x = date, y = conc)) +
  geom_point()
```

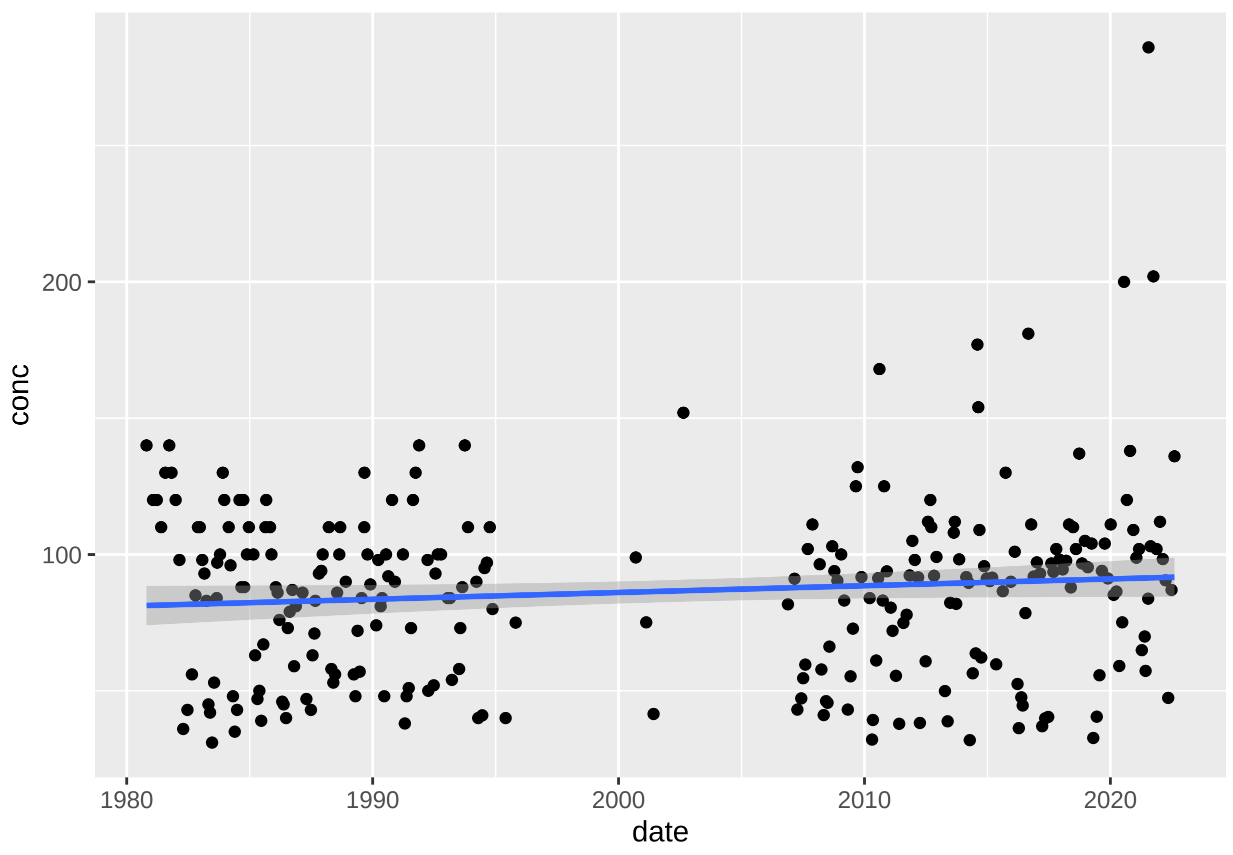


6.2 Adding a trend line with ggplot

`ggplot()` has an easy method for adding a trend line to plots (`stat_smooth`()). The code below uses a linear model (lm) to fit the line:

```
ggplot(dolores_ca, aes(x = date, y = conc)) +
  geom_point() +
  stat_smooth(method = 'lm') # the line that you add is the linear model
```

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



```
# minimizing the distance between the line and all the points in aggregate (it's what linear models do)
```

That line looks pretty flat!

6.2.1 Linear models for trend detection (the wrong way).

A very intuitive way to try to detect if there is a long term trend is to use linear models, as `ggplot()` does. So, let's go ahead and write out a model for daily calcium data using the `lm()` function. This class won't do a great job defining when you can use linear models, but this is one of the main functions that you will use. Your stats classes should give you more background on how to use `lm()` appropriately.

```
ca_model <- lm(conc ~ date, data = dolores_ca) # concentration as a function of date
summary(ca_model) # tells us the formula to run the model and the residuals. Big numbers mean bad fit.
```

```
##
## Call:
## lm(formula = conc ~ date, data = dolores_ca)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -58.145 -27.406   2.746  16.417 194.597
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) 7.859e+01  5.049e+00  15.565   <2e-16 ***
## date        6.806e-04  3.963e-04   1.717   0.0871 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 33.23 on 262 degrees of freedom
## Multiple R-squared:  0.01113,    Adjusted R-squared:  0.007355
## F-statistic: 2.949 on 1 and 262 DF,  p-value: 0.08712
```

```
# You want to look at:
# - R^2
# - Adjusted R^2
# - p-value
# NEVER USE LINEAR MODELS FOR TRENDS, YOU CANNOT USE LINEAR MODELS FOR TREND DETECTING
# Instead we should use Mann-Kendall tests and Tau's Sens Slope
```

6.2.2 The right way!

Using a linear model for trend detection breaks one of the cardinal rules of linear modelling, namely that each observation is assumed to be independent of any other observation. In a time-series like what we are looking at here, yesterday's calcium concentration is highly correlated with today's concentration. So linear models should never be used in trend detection. Instead we should use Mann-Kendall tests and Tau's Sens Slope.

6.2.2.1 Mann-Kendall test The Mann Kendall test is a non-parametric test of trends, you can use `?mk.test` to read more about the method, but it only requires an ordered time-series to run. Let's use it here.

```
dolores_ca <- dolores_ca %>%
  # Make sure data is arranged from 1980 onward.
  arrange(date)

dolores_mk <- mk.test(dolores_ca$conc)

print(dolores_mk) # we should get the tau sen's slope output: we should see that Ca is not really chang
```

```
##
##  Mann-Kendall trend test
##
## data:  dolores_ca$conc
## z = 1.1942, n = 264, p-value = 0.2324
## alternative hypothesis: true S is not equal to 0
## sample estimates:
##            S         varS          tau
## 1.713000e+03 2.055060e+06 4.958594e-02
```

The mk.test is really just a true/false where if the p-value is below some threshold (usually 0.05) then you can be mostly confident that there is a 'real' trend in the data. However it doesn't tell you the slope of that trend. For that you need to use `sens.slope()`.

```
dolores_slope <- sens.slope(dolores_ca$conc)

dolores_slope
```

```
##
##  Sen's slope
##
## data:  dolores_ca$conc
## z = 1.1942, n = 264, p-value = 0.2324
## alternative hypothesis: true z is not equal to 0
## 95 percent confidence interval:
##  -0.01470588  0.07080292
## sample estimates:
## Sen's slope
##  0.02356229
```

Notice that `sens.slope()` gives you a slope value, and a p-value (which is the same as an MK test). For this reason, I almost always just use `sens.slope()` so I get both significance and slope.

6.2.2.2 Cleaner output The output from these models is kind of messy if you are printing lots of model results. We can use the `tidy()` function from the `broom` package to clean up this output.

```
tidy(dolores_slope)
```

```
## # A tibble: 1 x 7
##   statistic p.value parameter conf.low conf.high method      alternative
##       <dbl>   <dbl>     <int>    <dbl>     <dbl> <chr>       <chr>
## 1      1.19   0.232       264  -0.0147    0.0708 Sen's slope two.sided
```

Some model objects don’t include both the p-value and the slope, which is slightly maddening, but we can make our own function to do this.

```
tidier <- function(mod = dolores_slope){

  tidy(mod) %>%
    mutate(slope = mod$estimates) # adds a new column that stores slopes as 'estimates'
  # be careful that you're putting down the slope and not the intercepts

}

tidier(mod = dolores_slope)
```

```
## # A tibble: 1 x 8
##   statistic p.value parameter conf.low conf.high method      alternative  slope
##       <dbl>   <dbl>     <int>    <dbl>     <dbl> <chr>       <chr>        <dbl>
## 1      1.19   0.232       264  -0.0147    0.0708 Sen's slope two.sided   0.0236
```

Ok, now we have an elaborate way to confirm what the plot already showed us. There is no long-term trend in calcium concentrations in the Dolores River.

7 Models everywhere!

Okay so we have already figured out how to model data at a single site for a single parameter, but is there an efficient way to do this for ALL sites and ALL parameters?

YES!

I'm glad you asked. We will use the magic of `nesting` data to apply our trend models to all our data. First let's alter the data a little to increase precision in our question.

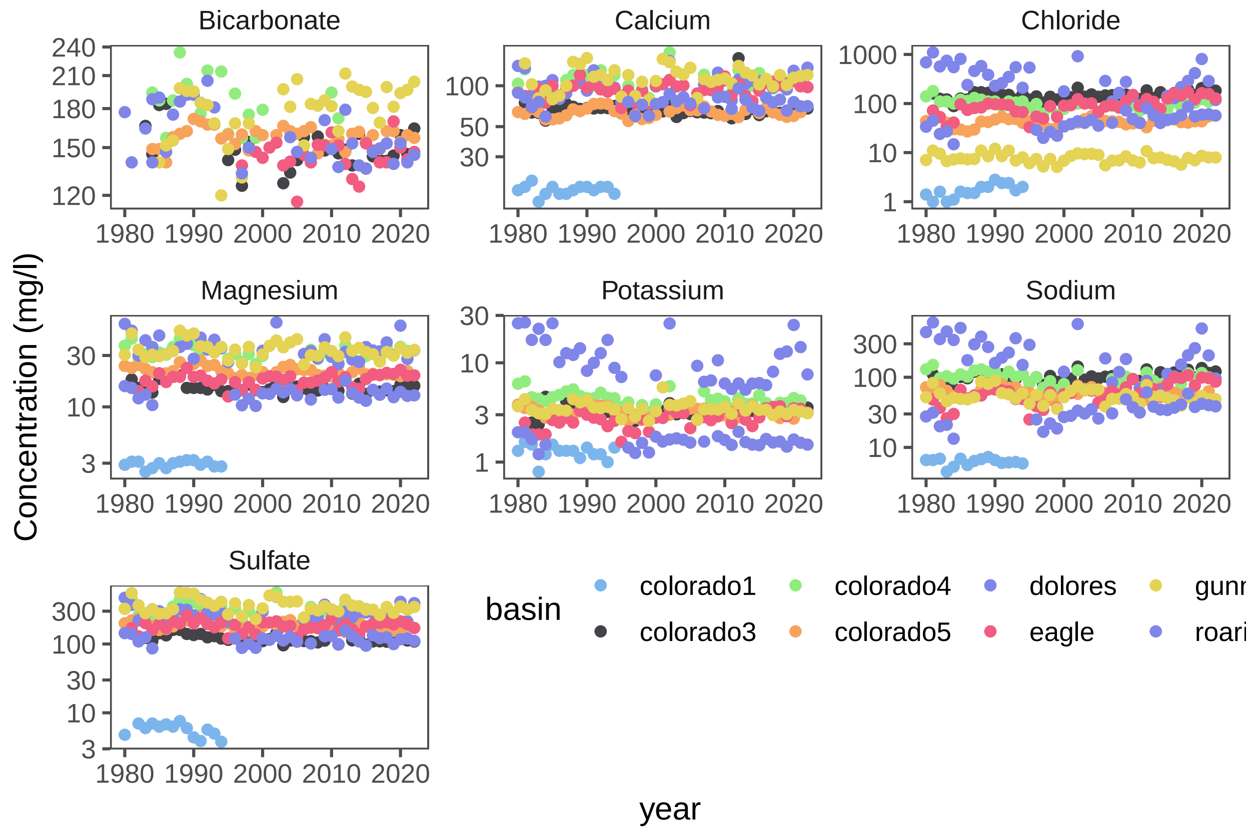
7.0.1 Converting data to late summer annual means

Water chemistry is heavily controlled by seasonality and water flow, so let's try to control for that and summarize our data to only include the low-flow periods of the year. Basically we will be focusing on: **are there trends in low flow concentrations of ions in the stream?**

```
low_flow <- wq %>%
  mutate(month = lubridate::month(date),
         year = lubridate::year(date)) %>%
  filter(month %in% c(8,9,10,11)) %>%
  group_by(basin, SiteID, parameter, year) %>%
  summarize(conc = median(conc, na.xrm = T))
```

```
## `summarise()` has grouped output by 'basin', 'SiteID', 'parameter'. You can
## override using the `.groups` argument.
```

```
ggplot(low_flow, aes(x = year, y = conc, color = basin)) +
  facet_wrap(~parameter, scales = 'free') +
  geom_point() +
  theme_few() +
  scale_y_log10() +
  scale_color_hc() +
  theme(legend.pos = c(0.7,0.2),
        legend.direction = 'horizontal') +
  ylab('Concentration (mg/l)')
```



7.1 The Magic of Nesting

Okay, so now we have a few things:

1. A dataset that has the data organized the way we want it.
2. A function (`sens.slope()`) we can use to look at if there are long-term trends in concentration.
3. A desire to apply this function to all of our sites and parameters.

To accomplish **3** we need to use the magic of `nest()`. Nesting allows us to group data by site and parameter (like with a `group_by` and a `summarize`) and apply models to each site and parameter separately. Effectively nesting bundles (or nests!) the data into tidy little packets that we can apply the model to. Let's try!

7.1.1 Nesting data

```
low_nest <- low_flow %>%
  # rename parameter as ion to make it more clear
  group_by(ion = parameter,basin) %>%
  nest()

head(low_nest)
```

```
## # A tibble: 6 x 3
## # Groups:   ion, basin [6]
##   basin     ion       data
##   <chr>     <chr>     <list>
## 1 colorado1 Calcium   <tibble [15 x 4]>
## 2 colorado1 Chloride  <tibble [15 x 4]>
## 3 colorado1 Magnesium <tibble [15 x 4]>
## 4 colorado1 Potassium <tibble [15 x 4]>
## 5 colorado1 Sodium    <tibble [15 x 4]>
## 6 colorado1 Sulfate   <tibble [14 x 4]>
```

The above code produces a tibble with three columns: `basin`, `parameter`, and `data`. The `data` column is our nested data (or 'bundled' data, as I like to think of it) for each basin-parameter combination.

7.1.2 Modelling over nested data

Now we just need to apply our model to the data. To do this we need to use the `map()` function. Map takes in an x (here, our `data` column) and then a function (in this case `sens.slope()`). We use `.x$conc` to indicate that we want to apply the model to the concentration column within each bundled (nested) data frame.

```
wq_models <- low_nest %>%
  mutate(mods = map(data, ~ sens.slope(.x$conc))) # This part is very critical. The function to call th

head(wq_models)
```

```
## # A tibble: 6 x 4
## # Groups:   ion, basin [6]
##   basin     ion       data              mods
##   <chr>     <chr>     <list>            <list>
## 1 colorado1 Calcium   <tibble [15 x 4]> <htest>
## 2 colorado1 Chloride  <tibble [15 x 4]> <htest>
## 3 colorado1 Magnesium <tibble [15 x 4]> <htest>
## 4 colorado1 Potassium <tibble [15 x 4]> <htest>
## 5 colorado1 Sodium    <tibble [15 x 4]> <htest>
## 6 colorado1 Sulfate   <tibble [14 x 4]> <htest>
```

Now we have a nested data set AND nested models (that are hard to see). We can look at a single model by indexing it.

```
# This provides the 15th model summary
wq_models$mods[15] # go into the water quality models, go to the 15th one and give me its output.
```

```
## [[1]]
##
##  Sen's slope
##
## data:  .x$conc
## z = 1.2567, n = 37, p-value = 0.2089
## alternative hypothesis: true z is not equal to 0
## 95 percent confidence interval:
##  -0.1857143  0.9076923
## sample estimates:
## Sen's slope
##   0.3366667
```

But that is a tedious way to look at our model summaries!

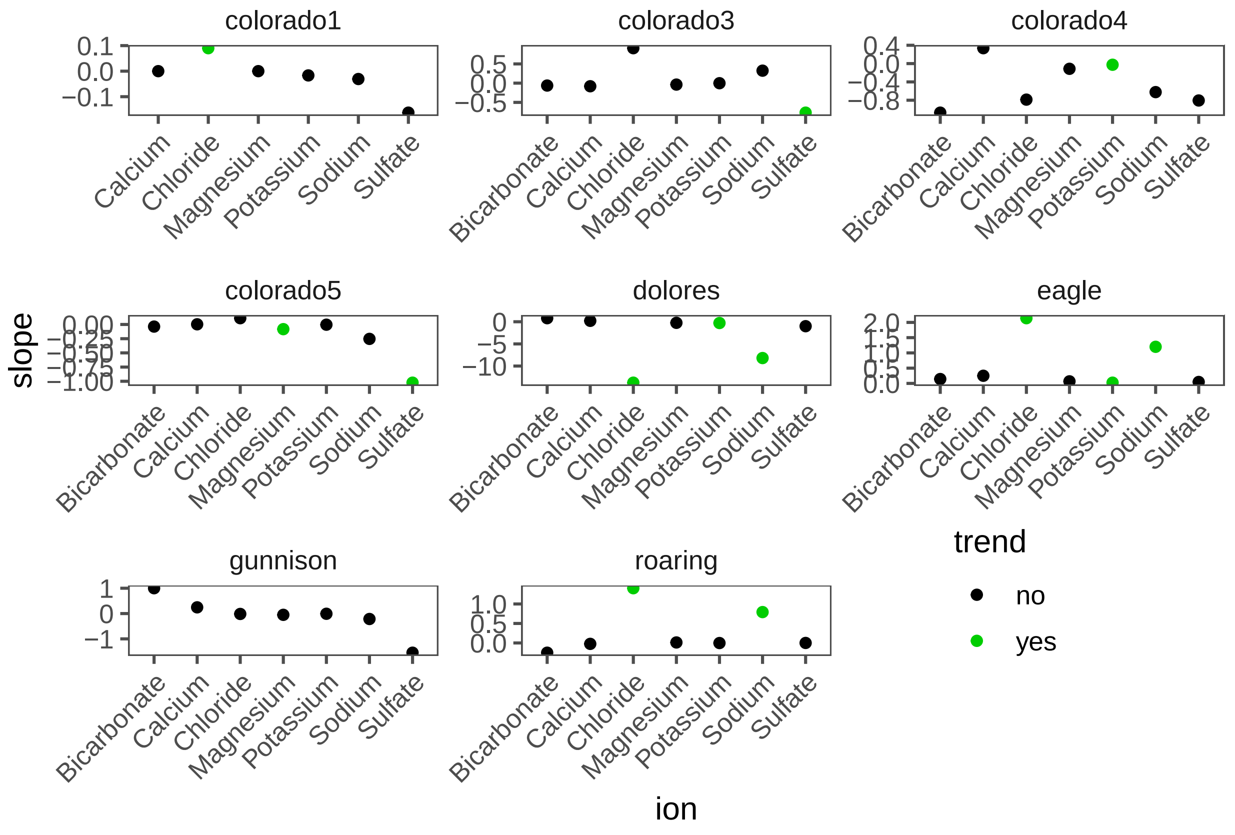
So now let's use the power of our `tidier()` function and `unnest()`. Again, we use `map()` to apply our `tidier()` function to all of the raw `sens.slope` models, and we extract p.value and slope in a clean table. We then use `unnest()` to unravel that data so we have a final data frame that contains all of the model outputs.

```
wq_mod_summaries <- wq_models %>%
  mutate(tidy_mods = map(mods, tidier)) %>% # make a new model called tidy_mods, and maps the mods to t
  unnest(tidy_mods) %>%
  dplyr::select(basin, ion, p.value, slope) %>%
  mutate(trend = ifelse(p.value < 0.01, 'yes', 'no')) # we run it on such a low p-value to avoid gettin

head(wq_mod_summaries)
```

```
## # A tibble: 6 x 5
## # Groups:   ion, basin [6]
##   basin     ion       p.value   slope trend
##   <chr>     <chr>       <dbl>   <dbl> <chr>
## 1 colorado1 Calcium   0.754    0      no
## 2 colorado1 Chloride  0.00320  0.09   yes
## 3 colorado1 Magnesium 0.725    0      no
## 4 colorado1 Potassium 0.247   -0.0167 no
## 5 colorado1 Sodium    0.690   -0.0308 no
## 6 colorado1 Sulfate   0.0419  -0.162  no
```

7.1.3 Visualizing model output.

```
ggplot(wq_mod_summaries,aes(x = ion, y = slope, color = trend)) +
  geom_point() +
  facet_wrap(~basin,scales = 'free') +
  theme_few() +
  scale_color_manual(values = c('black','green3')) +
  theme(axis.text.x = element_text(angle = 45, hjust = 1),
        legend.pos = c(0.8, 0.1))
```



8 Assignment

The above workflow really focuses on trend detection, but we want to focus some on actual linear models. As such we want to join our discharge (Q) data to our water quality (WQ) data and we want to look at the relationship between Q and WQ.

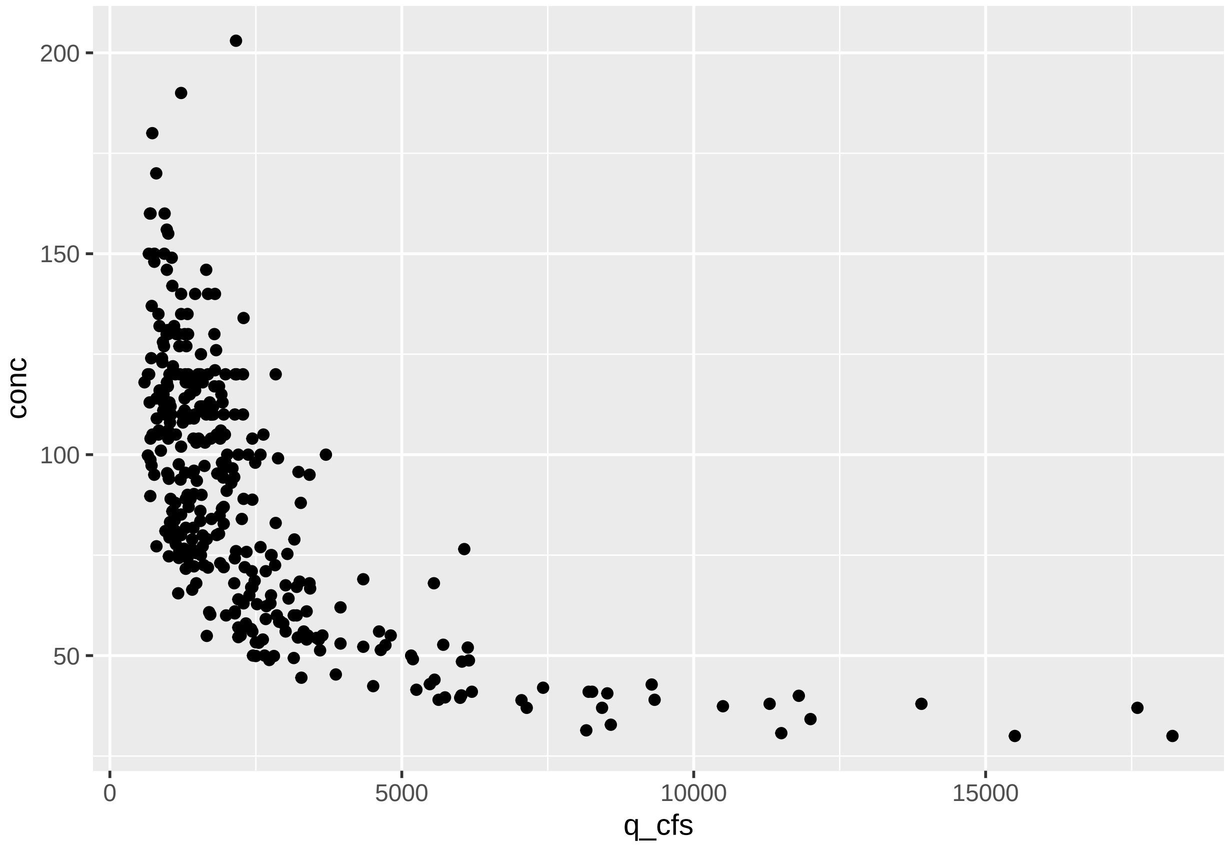
8.1 Join discharge and water quality data.

Use `inner_join()` to join our discharge data to our water quality data. You want to join by both date and siteid. Remember! the discharge data has ids that drop the `USGS-` so you will need to add that back in using a `paste()`.

```r
# make sure you join by site AND date
# mutate siteID
# rename

# Add "USGS-" prefix to site column in discharge data
usgs_q_data <- q_data %>%
  mutate(site_no = paste("USGS-", site_no, sep = "")) %>%
  rename(SiteID = site_no) %>%
  rename(date = Date)

# Inner join by date and site ID
joined_data <- inner_join(wq, usgs_q_data, by = c("SiteID", "date"))
```

Pick any site and ion combination and plot Q versus concentration. What do you see in this relationship?

```
gunnison_ca <- joined_data %>%
  filter(basin == 'gunnison', parameter == 'Calcium')  # is the gunnison river over the years changing

ggplot(gunnison_ca, aes(x = q_cfs, y = conc)) +
  geom_point()
```



```
### There is a negative logarithmic relationship between calcium concentrations and river flow speed
```

Group your data by basin and ion and nest the data, use the `head()` function to print the first several rows of your nested data

```
gunnison_mk <- mk.test(gunnison_ca$conc)

basin_ion_nest <- joined_data %>%
  # rename parameter as ion to make it more clear
  group_by(ion = parameter,basin) %>%
  nest()

head(basin_ion_nest)
```

```
## # A tibble: 6 x 3
## # Groups:   ion, basin [6]
##   basin   ion         data
##   <chr>   <chr>       <list>
## 1 roaring Bicarbonate <tibble [265 x 7]>
## 2 roaring Calcium     <tibble [317 x 7]>
## 3 roaring Chloride    <tibble [317 x 7]>
## 4 roaring Magnesium   <tibble [317 x 7]>
## 5 roaring Potassium   <tibble [317 x 7]>
## 6 roaring Sodium      <tibble [318 x 7]>
```

8.2 Apply a linear model to the data.

You will need to use a `map()` command like this: `map(data, ~lm(conc ~ q, data = .x))`

```
joined_models <- basin_ion_nest %>%
  mutate(mods = map(data, ~lm(conc ~ q_cfs, data = .x))) # This part is very critical. The function to

head(joined_models)
```

```
## # A tibble: 6 x 4
## # Groups:   ion, basin [6]
##   basin   ion         data               mods
##   <chr>   <chr>       <list>             <list>
## 1 roaring Bicarbonate <tibble [265 x 7]> <lm>
## 2 roaring Calcium     <tibble [317 x 7]> <lm>
## 3 roaring Chloride    <tibble [317 x 7]> <lm>
## 4 roaring Magnesium   <tibble [317 x 7]> <lm>
## 5 roaring Potassium   <tibble [317 x 7]> <lm>
## 6 roaring Sodium      <tibble [318 x 7]> <lm>
```

Summarize your data using `tidy()`. You should have a new column called `mods` or something similar and you need to "tidy" those mods.

```
tidy_joined_models <- joined_models %>%
  mutate(tidy_mods = map(mods, ~broom::tidy(.x))) %>%
  unnest(tidy_mods) %>%
  filter(term == "q_cfs") %>%
  dplyr::select(basin, ion, p.value, slope = estimate) %>%
  mutate(trend = ifelse(p.value < 0.01, 'yes', 'no'))
head(tidy_joined_models)
```

```
## # A tibble: 6 x 5
## # Groups:   ion, basin [6]
##   basin   ion          p.value     slope trend
##   <chr>   <chr>          <dbl>     <dbl> <chr>
## 1 roaring Bicarbonate 1.98e-38 -0.0112   yes
## 2 roaring Calcium     5.51e-76 -0.00856  yes
## 3 roaring Chloride    3.13e-35 -0.00583  yes
## 4 roaring Magnesium   1.87e-71 -0.00164  yes
## 5 roaring Potassium   3.41e- 8 -0.000147 yes
## 6 roaring Sodium      6.35e-44 -0.00425  yes
```

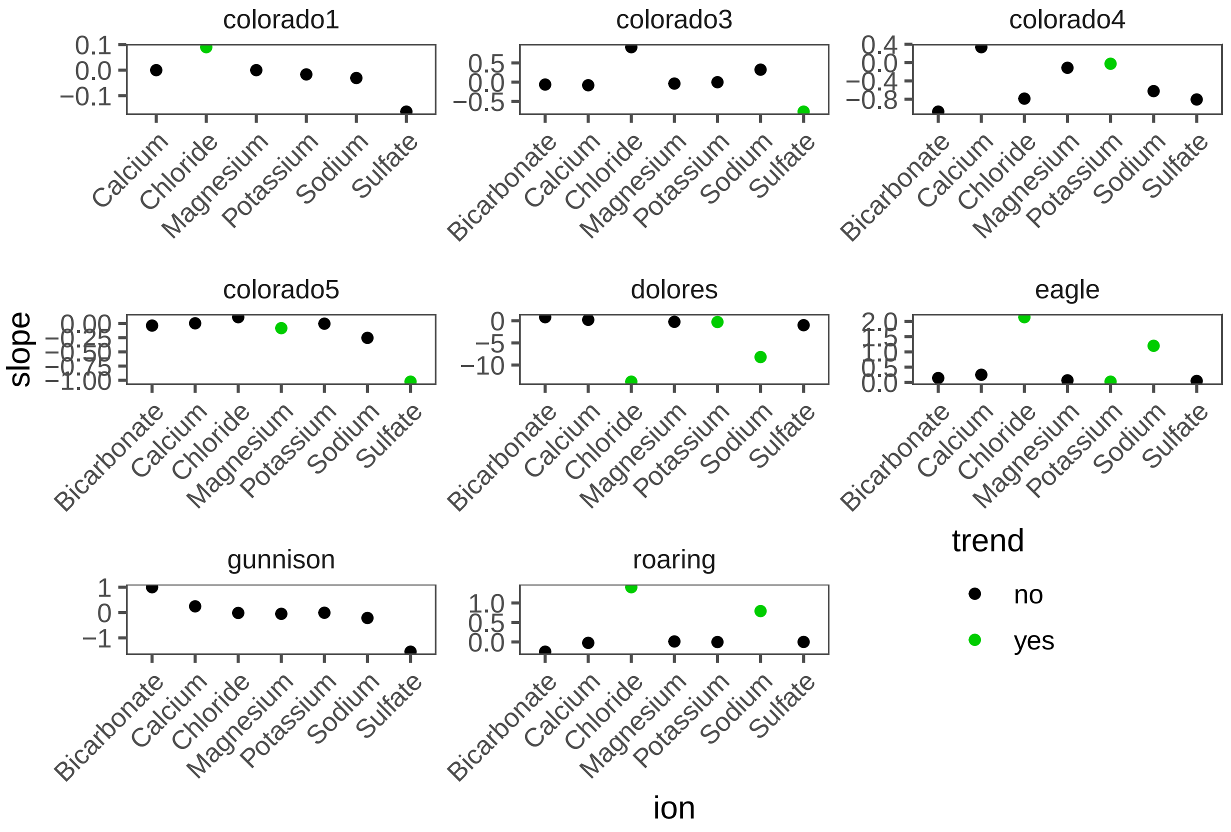
8.3 Visualize the data.

Make a visual of your model summaries that shows a) which sites have significant relationships between discharge and concentration, and b) the slope of that relationship.

```
joined_mod_summaries <- wq_models %>%
  mutate(tidy_mods = map(mods, tidier)) %>% # make a new model called tidy_mods, and maps the mods to t
  unnest(tidy_mods) %>%
  dplyr::select(basin, ion, p.value, slope) %>%
  mutate(trend = ifelse(p.value < 0.01, 'yes', 'no')) # we run it on such a low p-value to avoid gettin

head(wq_mod_summaries)
```

```
## # A tibble: 6 x 5
## # Groups:   ion, basin [6]
##   basin     ion       p.value   slope trend
##   <chr>     <chr>       <dbl>   <dbl> <chr>
## 1 colorado1 Calcium   0.754    0      no
## 2 colorado1 Chloride  0.00320  0.09   yes
## 3 colorado1 Magnesium 0.725    0      no
## 4 colorado1 Potassium 0.247   -0.0167 no
## 5 colorado1 Sodium    0.690   -0.0308 no
## 6 colorado1 Sulfate   0.0419  -0.162  no
```

```
ggplot(joined_mod_summaries,aes(x = ion, y = slope, color = trend)) +
  geom_point() +
  facet_wrap(~basin,scales = 'free') +
  theme_few() +
  scale_color_manual(values = c('black','green3')) +
  theme(axis.text.x = element_text(angle = 45, hjust = 1),
        legend.pos = c(0.8, 0.1))
```



8.4 Bonus

Look up the `furrr` package. What does `furrr::map()` do that is different from `purrr::map()`?

When would you want to use this `furrr::` function instead of `purrr::`?

You'd want to use the furrr:: function when dealing with an exceptionally large dataset or with computationally intensive code, because it runs functions in parallel (or in chunks/clusters), significantly reducing computation time and increasing efficiency.