Data exploration in R

Questions:

"How can I use R to do explore my data?"

Objectives: Get an introduction to:

- date formats
- loops
- if statements
- plotting
- subsetting data
- annotating code

Keypoints:

- R can save you from doing repetative things manually
- Google is your friend
- Well annotated code will make your future self really happy with your current self, particularly after vacation

```
temps <- read.csv(file = "data/stream-temps.csv")
temps_raw <- read.csv(file = "data/stream-temps-raw.csv")
```

Keep track of who wrote your code and its intended purpose

Starting your code with an annotated description of what the code does when it is run will help you when you have to look at or change it in the future. Just one or two lines at the beginning of the file can save you or someone else a lot of time and effort when trying to understand what a particular script does.

```
# Stream Temperature Data Exploration
# Kendra Kaiser
# February 11, 2020
```

Dates

One of the most common things we have to deal with when working with timeseries is converting dates into a format that is readable by R. The `as.Date` function converts from a character to R's `date` class, where the first argument is the character you want to convert, and the second is the structure of the date that is in the first argument. When I am trying something new that I dont have existing code to copy from, I often like to test things on a single value.

So when we check the first `MonthYear` value

```
temps$MonthYear[1]
```

```
## [1] 07-2010
## 229 Levels:  01-1993 01-1994 01-1995 01-1996 01-1997 01-1998 ... 12-2011
```

we can see that the format is month - year, which according to https://www.r-bloggers.com/date-formats-in-r/ is equivilant to %m-%Y.

```
as.Date(temps$date[1], "%m-%Y")
```

```
## Error in as.Date.default(temps$date[1], "%m-%Y"): do not know how to convert 'temps$date[1]' to class
```

But for some reason we get an error, after searching around on the internet, I found that the `as.Date` function needs a day of the month. One way to get around this is to define the date as the first of the month using the `paste` function.

```
temps$date <- paste("01-",temps$MonthYear,sep="")
temps$date <- as.Date(temps$date, "%d-%m-%Y")
```

We can confrim this worked with the `anyNA` function which can tell you if there are missing values in a vector.

```
anyNA(temps$date)
```

```
## [1] TRUE
```

When we check the data frame to see what happened we can see come samples don't have a MonthYear value, but they do have `SampleMonth` and `SampleYear`!

How to do efficient QAQC using loops and if statements

Here, I'll combine a `loop` and an `if` statement to fill in the empty `temps$date` values. `loops` can be extremely useful for calculations that need to be done on individual values in an vector, matrix, or dataframe. On very large datasets there are other methods that are even faster than loops, but it is useful to understand how they work, and they are often sufficient, particularly as a beginner.

The format of a `loop` is:

> for(`index` in `length`){
> do this thing
> }

The format of a simple `if statement` is:
> if (`test expression`){
> do this thing
> }

We can now combine these with the `paste` function to make sure all samples have a date.

```
for (i in nrow(temps)) {
  if (is.na(temps$date[i])){
    temps$date <- as.Date(paste("01", temps$SampleMonth,    temps$SampleYear, sep="-"), "%d-%m-%Y")
  }
}
```

Now we can double check to see if our loop and if statement worked!

```
anyNA(temps$date)
```

```
## [1] FALSE
```

Annotating Code

If this is code that I will use over and over for QAQC, or will share with a colleauge it is *extremely valuable* to annotate it well.

What would you write to explain what we did there?

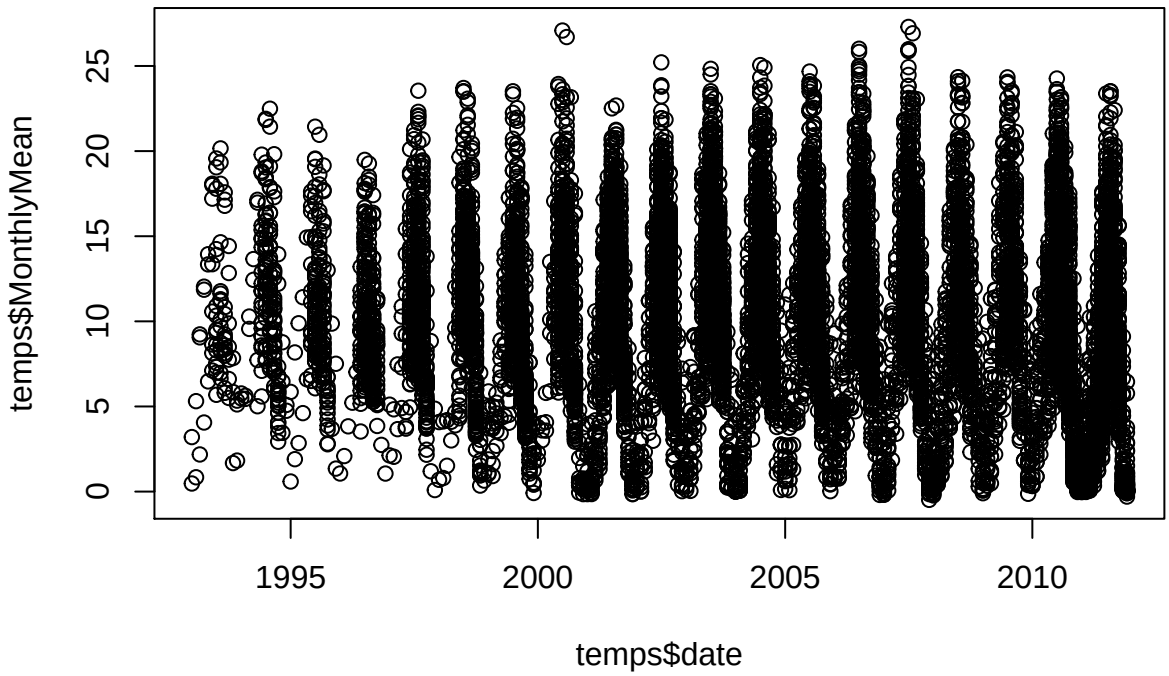
```r
# Combine year and month into date column to ensure samples have dates associated with them. Here, we h

for (i in nrow(temps)) {
  if (is.na(temps$date[i])){
    temps$date <- as.Date(paste("01", temps$SampleMonth,    temps$SampleYear, sep="-"), "%d-%m-%Y")
  }
}
```

Documentation: tell us what and why, not how

When you first start out, your comments will often describe what a command does, since you're still learning yourself and it can help to clarify concepts and remind you later. However, these comments aren't particularly useful later on when you don't remember what problem your code is trying to solve. Try to also include comments that tell you why you're solving a problem, and what problem that is. The how can come after that: it's an implementation detail you ideally shouldn't have to worry about.

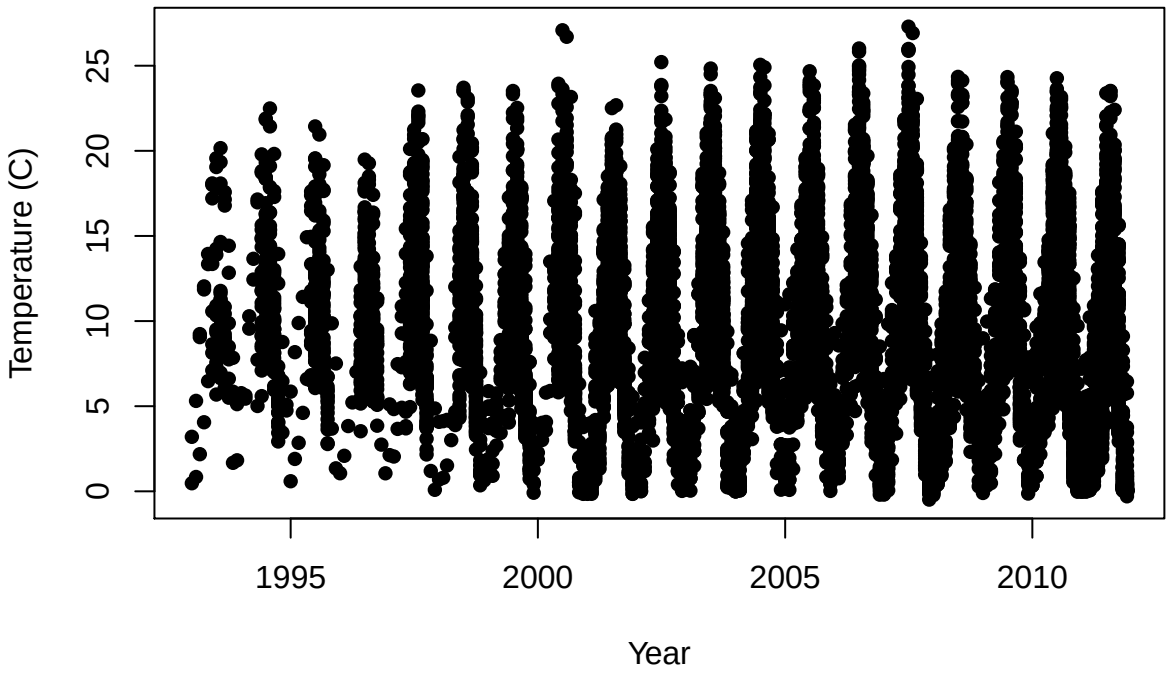
Basic Plotting

```r
plot(temps$date, temps$MonthlyMean)
```



R will automatically name the axes based on the code in the plotting funtion, we can adjust those very easily using `xlab` and `ylab`! The data is hard to visusalize as open circles, so we can change the shape using `pch` (https://www.statmethods.net/advgraphs/parameters.html).

```r
plot(temps$date, temps$MonthlyMean, pch=16, xlab="Year", ylab = "Temperature (C)")
```



R has some really amazing plotting capabilities, `ggplot` is the most commonly used package for making beautiful figures. There are so many resources online to get into all the details.

Subsetting data using logical operations

In addition to using indexes and slices, we can also extract elements by using logical vectors and operators.

Perhaps we only want to use data that has already been QAQC'd:

```
class(temps_raw$QAQC)
```

```
## [1] "integer"
```

```
temps_raw$QAQC[1:5]
```

```
## [1] 0 1 1 1 1
```

```
x<- c(1,2,3,4,5)
x[temps_raw$QAQC[1:5]]
```

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

This selects elements of x where the QAQC vector is 'TRUE', so if you apply this to the whole column of mean temperatures,

```
length(temps$MonthlyMean[temps_raw$QAQC])
```

```
## [1] 11233
```

we now only have 11,233 observations.

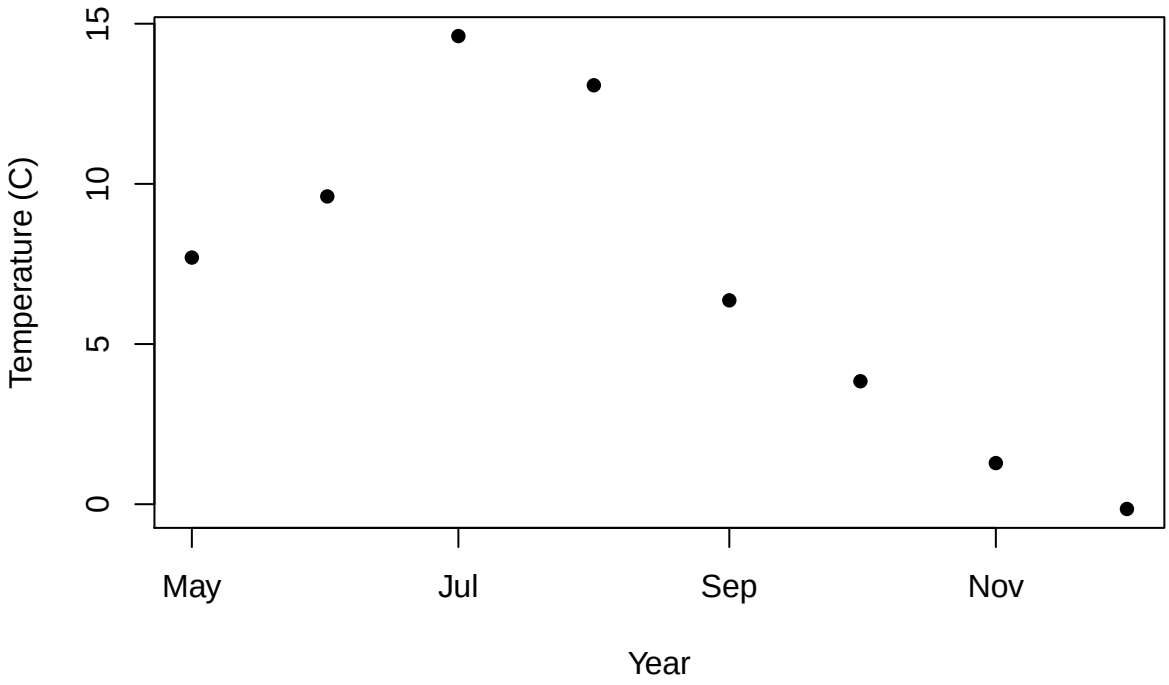
The == allows for direction comparison, this is particuarly useful with character identification, so if we wanted to plot the data for one reach,

```
#subset to reach of interest
temps$MonthlyMean[temps$NoRWeST_ID == 'MidSnake_175']
```

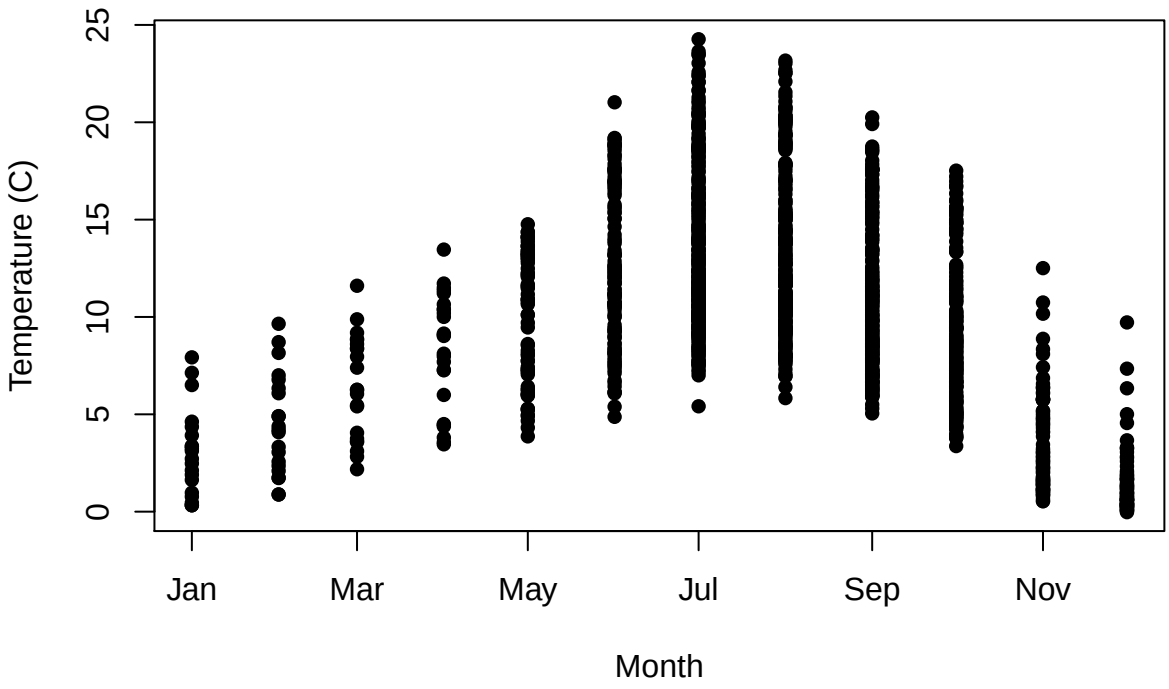
```
## [1]  7.7003788  9.6086111 14.6145161 13.0788690  6.3645000  3.8392742
```

```
## [7]  1.2838718 -0.1484409
```

```
plot(temps$date[temps$NoRWeST_ID == 'MidSnake_175'], temps$MonthlyMean[temps$NoRWeST_ID == 'MidSnake_175
```




```
#or zoom into a specific year
plot(temps$date[temps$SampleYear == 2010], temps$MonthlyMean[temps$SampleYear == 2010], pch=16, xlab="M
```



Combining logical conditions

We often want to combine multiple logical criteria. For example, we might want to find all the data located in 'MidSnake_174' **or** 'MidSnake_175' **and** have that have been QAQCd. Several operations for combining

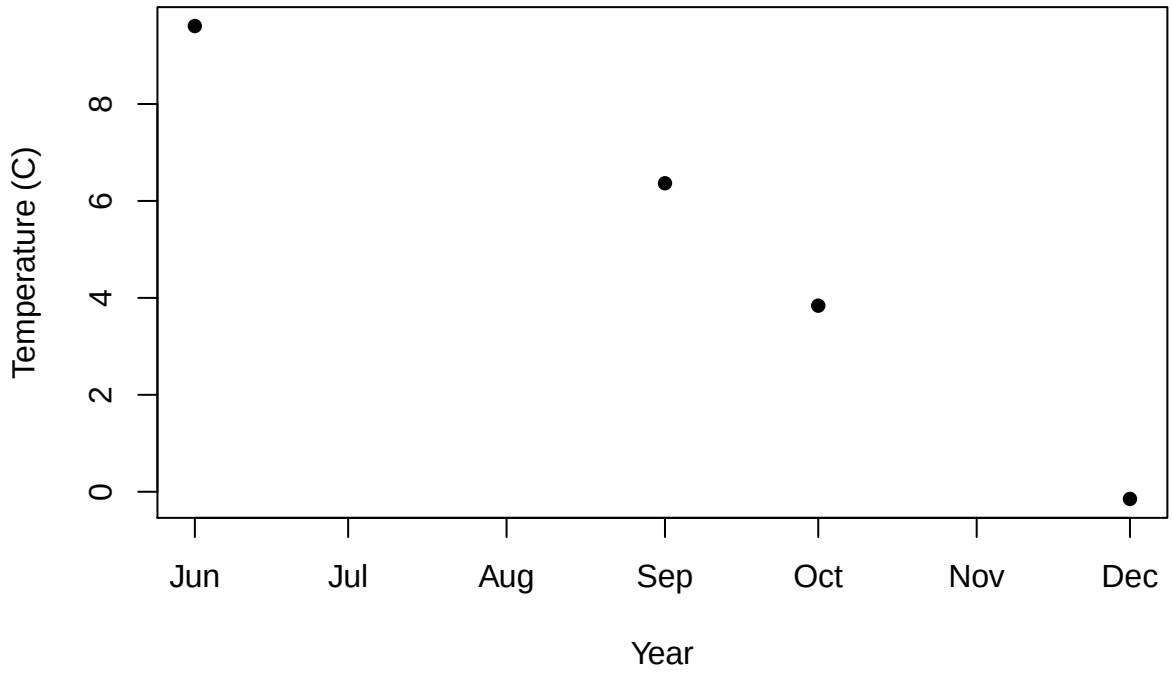
logical vectors exist in R:

- `&`, the "logical AND" operator: returns `TRUE` if both the left and right are `TRUE`.
- `|`, the "logical OR" operator: returns `TRUE`, if either the left or right (or both) are `TRUE`.

```
#check to see if the data in the reach has been QAQCd
temps_raw$QAQC[temps$NoRWeST_ID == 'MidSnake_175']
```

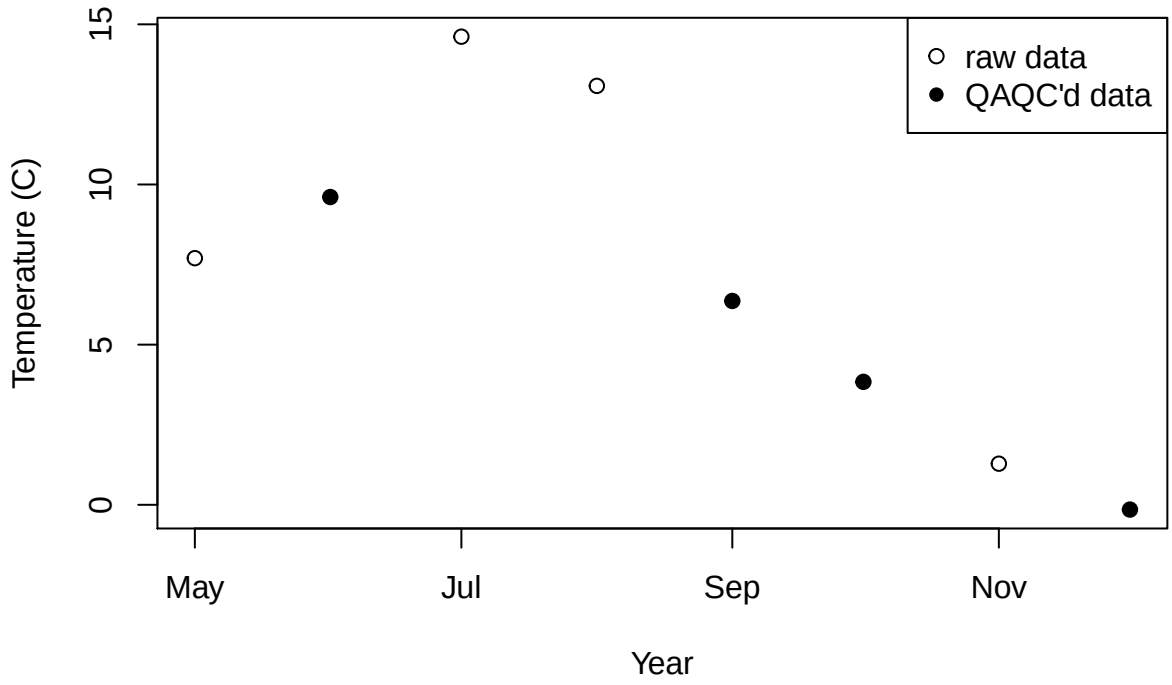
```
## [1] 0 1 0 0 1 1 0 1
```

```
#Only Plot data that has been QAQC'd
plot(temps$date[temps_raw$NoRWeST_ID == 'MidSnake_175' & temps_raw$QAQC], temps_raw$MonthlyMean[temps_ra
```



You can also combine two datasets in one figure:

```
#Plot both raw and QAQC'd data in one figure
plot(temps$date[temps$NoRWeST_ID == 'MidSnake_175'], temps$MonthlyMean[temps$NoRWeST_ID == 'MidSnake_175

points(temps$date[temps_raw$NoRWeST_ID == 'MidSnake_175' & temps_raw$QAQC], temps_raw$MonthlyMean[temps_

legend("topright", legend = c("raw data", "QAQC'd data"), pch = c(1, 16))
```



Plotting lots of things easily!

```
reaches <- unique(temps$NoRWeST_ID)

for (i in 175:178){
  plot(temps$date[temps$NoRWeST_ID == reaches[i]], temps$MonthlyMean[temps$NoRWeST_ID == reaches[i]], x

  points(temps$date[temps_raw$NoRWeST_ID == reaches[i] & temps_raw$QAQC], temps_raw$MonthlyMean[temps_ra

  legend("topright", legend = c("raw data", "QAQC'd data"), pch = c(1, 16))
}
```

Tips and Best Practices

Break down problem into bite size pieces

When you first start out, problem solving and function writing can be daunting tasks, and hard to separate from code inexperience. Try to break down your problem into digestible chunks and worry about the implementation details later: keep breaking down the problem into smaller and smaller functions until you reach a point where you can code a solution, and build back up from there.

Style

Be consistent with naming conventions, break up sections of code with `#`

```
# ----- Import Data -----#

# ----- QAQC Data -----#

# ----- Plot Data -----#
```

R packages for retrieving hydrologic data and modeling

https://cran.r-project.org/web/views/Hydrology.html

Doing GIS analysis in R and Python:

Ever crashed your computer using ArcMap? Ever pressed "go" on an ArcGIS tool and proceeded to clean out five years of old emails while you waited, just to do it again 10 more times? Automation using R or python will make your life better.

EarthLab https://www.earthdatascience.org/courses/earth-analytics/spatial-data-r/intro-vector-data-r/