# Laboratory 2 – Exploring Data Using R, Basic Graphing

Objectives:

This week, we will be delving into the specifics of R and its use as a data analysis toolkit. The commands and functionality that you will learn today can be used in conjunction with the programmatic structures from prior weeks to create very complex workflows. It’s good to keep that in mind as you work through this week’s lab, even though today we will be performing tasks very sequentially, working through a basic data processing workflow and hopefully scratching the surface of what R has to offer.

### Review and Introduction:

Today you will be putting into practice everything you know about R, data objects, indices and more, and apply it using new functions for object manipulation and plotting. The data you will be working with are simulated, and *very* loosely based on a real world problem. But I’m going to describe a hypothetical situation to you in order for you understand how what you learn could be applied to real data.

Imagine you get a call from a geographer friend, who has a dataset he’s having problems with. The data are from a long-term intertidal research station and were collected for a study looking at how nitrogen deposition from very-near-shore agriculture is impacting kelp growth. The data were collected at 100 marked kelp locations, sampled on a regular 10 x 10 grid. At each location there’s a remote sensing device recording a variety of local environmental conditions including an estimate of dissolved organic nitrogen.

The 10 x 10 station grid was set up to monitor the influx of nitrogen entering the ocean from increased agricultural runoff. The hypothesis is that over a relatively short period of time, increased nitrogen effects an increase in kelp growth, but that association is mediated by spatial location and length of exposure to increased nitrogen. The study has been running for 24 months, with nitrogen and kelp growth measured at low-tide, near mid-month at each of the 100 stations.

Your friend has collected the data from his research assistants and now needs help analyzing them. In the /data folder you will find three “CSV” files. These files are in comma-separated-value format and are useful because both R and Excel read and save to them natively. Feel free to open them in Excel to get a feel for the data and its structure in a familiar setting. Don’t save any changes to the files!

### The Problem

Your friend has gotten the data in three separate files. One file, growth.csv, contains the station ID, the timestamp from the day the data were entered, and the growth for that month. The second file, nitrogen.csv, contains the station ID, timestamp and the station’s measurement of dissolved organic nitrogen. The third file contains the station location in X/Y coordinates and the corresponding station ID.

Your friend is having problems getting everything into a format where he can begin to test hypotheses regarding the influence of space and time on the hypothesized association between increasing nitrogen and increasing growth. Your goal is to combine this data into a complete dataset and help him look at the structure and associations of the data. The process will need to take the following steps:

* **Create a new, empty script file**
* **Read in our growth.csv data**
* **Read in our nitrogen.csv data**
* **Merge the nitrogen measurement into the same dataframe as our growth**
* **Tag each record in our dataframe with the appropriate station ID’s X/Y coordinates**
* **Conduct a preliminary data exploration using descriptive statistics and plots**

## Questions

1. **This week’s lab score will be based on your written code alone. I will be running your code and reading your comments to make sure you’ve accomplished the goal and understood the process. [100 points]**

**[Total Points: 100]**

### Loading Our Data

First, we will want to set our working directory to the location where our data are located. To do this we need to use the following:

setwd( “directory”)

By running this command you will set the default location where R will look for files to load and will be the folder where “save” commands will put their output.

Next, we need to load our growth, nitrogen and station datasets. Because they are in CSV format R can read (and write) to these files directly. We will use one of the many “read.*type*()*”* commands to accomplish this:

read.csv( “filename”, header=TRUE)

If you look up the help file on read.csv() you’ll see that the header=TRUE argument is the default, and therefore doesn’t need to be explicitly passed to the function. If you have a CSV file without a header row (column names) then you need to explicitly set it to FALSE. Please also use the argument stringsAsFactors = T as we will need to have certain variables be factors later.

The output from this command needs to be assigned to a data object. You can choose whatever you’d like to name your objects, but I will present code under the assumption that you read the data in with the following:

growth <- read.csv(“growth.csv”, stringsAsFactors = T)

nitrogen <- read.csv(“nitrogen.csv”, stringsAsFactors = T)

station\_id <- read.csv(“station\_id.csv”)

Let’s look at the structure of each of our files and their data. Choosing the growth dataframe first, try the following:

head(growth)

class(growth)

str(growth)

summary(growth)

## Questions

1. **From looking at the structure and summary output, how many factor columns were created when reading the growth CSV file? [5 bonus points]**

* **Two columns are characterized as factors and other is characterized as a numerical**

1. **Knowing what you do about the data (24 months of data, 100 stations) what does summary() tell you about factor variables? [5 bonus points]**

* **Summary spits out how many observations for each factor, along with how many unique obs of each factor found in other**

station\_id timestamp growth

DB1 : 24 1/14/01 4:30 : 100 Min. :-178.7

DB10 : 24 1/14/02 10:30 : 100 1st Qu.: 117.9

DB100 : 24 10/14/00 21:00: 100 Median : 237.6

DB11 : 24 10/15/01 3:00 : 100 Mean : 257.5

DB12 : 24 11/14/00 7:30 : 100 3rd Qu.: 379.2

DB13 : 24 11/14/01 13:30: 100 Max. : 872.3

(Other):2256 (Other) :1800

The next step is to merge the growth and nitrogen data into a single data frame. Normally, we might be tempted to just take the columns from one data frame and bind them to another dataframe. You can do the same with rows. The functions to do this are “column bind” and “row bind.” Try the following functions and understand what they’re doing:

cbind( 1:10, 21:30)

rbind( 1:5, 9:13)

cbind(1:3, 1:8)

Notice in the last cbind() call how a short vector is repeated to fill as much of the column that is needed to match the length of the second.

Now let’s double check the structure of our growth and nitrogen datasets by looking at a few more rows than head() gives us:

growth[1:50,]

nitrogen[1:50,]

What do you notice about the ordering of the data? Can we just bind the columns together from the two separate files? No, because the data aren’t in the same order by station\_id and timestamp. We are going to instead use the merge() function to join these two dataframes together, using the common columns of station\_id and timestamp to make sure that the data values are aligned correctly. The syntax for the merge function requires the two data.frame objects to be merged and a “by” argument is a vector of column names on which to join. For example we can merge the following two data.frames using the following:

a <- data.frame(“id”=1:10, “a”=11:20)

b <- data.frame(“id”=1:10, “b”=21:30)

c <- merge(a, b, by=c(“id”))

Now knowing how merge() works, merge our growth and nitrogen dataframes using both “station\_id” and “timestamp” in the “by” vector. Let’s store the merge() output in a new dataframe called all\_data.

The last thing to do to complete our dataset, matched row for row so that we can begin our data analysis is to attach X/Y locations to our data based on our station\_id. We can use the same merge() command to merge our all\_data data.frame and the station\_id data.frame, this time using only “station\_id” as the “by” vector. You can store the merge() output in the same all\_data object.

all\_data <- merge(all\_data, station\_id, by=c(“station\_id”))

If you do a summary on your all\_data object you should now have a data.frame with six columns in it. Remember that running names() will allow you to view and set the names of data.frame columns if you want to (don’t necessarily run these, just be aware that you could rename the columns if you wanted to):

names(all\_data)

names(all\_data) = c(“a”, “b”, “c”, “d”, “e”, “f”)

### Looking for Problems

Running summary() on the all\_data, combined data.frame you should pay close attention for any missing values or other items that look out of order. You might notice negative values in nitrogen and growth, but you decide since you know nothing about the sensor or the way the data were collected that these are ok. But the NA’s (missing values) in the X/Y columns are particularly troubling.

Let’s see which records these are by pulling out any records where x or y is an NA:

all\_data[is.na(all\_data$x) | is.na(all\_data$y),]

It turns out that all the missing X/Y data belong to a single station, DB5. Let’s explore this data further by plotting out the locations of our stations:

plot(x=all\_data$x, y=all\_data$y)

And let’s label each point with the station\_id:

text(x=all\_data$x, y=all\_data$y, labels=all\_data$station\_id, cex=0.6, pos=4)

If you want to know what the cex and pos arguments are doing, feel free to look up the help file for the text() function. But for now, you can see that we are missing a point at location (5, 1), where the station ID DB5 should exist. We know that there are NA values at that location and need to replace the NAs in the x column with 5’s, and y column with 1’s before we proceed.

There are several options. We can edit those values by hand two ways. Both involve the use of the edit() command:

all\_data$x <- edit(all\_data$x)

all\_data$y <- edit(all\_data$y)

This is a quick and dirty way of hand editing the raw data structure as though you were typing the whole thing into R. It is crude at best. When you’re done editing the values you would close the Editor Window, allowing it to save, and the new values would replace the old in the all\_data dataframe.

A slightly less crude method is to edit the entire data frame in a crude GUI:

all\_data <- edit(all\_data)

Once you close the editor window the edited data.frame will be assigned to the data object you specified. But since we know that the missing coordinates are all one station, we can just set those values using the same indices that we used to pull those records above:

all\_data$x[is.na(all\_data$x)] <- 5

Double check that we now have the value 5 in the x column where there used to be NAs:

all\_data[is.na(all\_data$x) | is.na(all\_data$y),]

Now let’s do the same for y:

all\_data$y[is.na(all\_data$y)] <- 1

Re-plot the points and labels and re-run summary() on all\_data to confirm that we now have no missing values in our data.

### Finishing it Up

Recall that our timestamp is currently a factor. This won’t work well because factors don’t sort according to value, but instead according to alphabetical order. For example, let’s plot growth against our timestamp.

NOTE: Are you tired of typing in all\_data$... every time you want to reference a data column within the all\_data data.frame? We can “attach” it so that the columns within the data.frame show up when R looks for a variable with an appropriate name, but columns within the data.frame will be masked by any data objects with the same name already in memory. Since we have “growth” and “nitrogen” already defined as data.frames that we read in from CSV files, we need to remove them first:

ls()

rm(growth)

rm(nitrogen)

ls()

Now we can attach the all\_data data.frame and use those column names inside it directly:

attach(all\_data)

growth[1:10]

Oops! Do you see the warning? The station\_id column is also masked by the station\_id data.frame. We can fix this by detaching the data.frame, removing the station\_id data.frame and re attaching:

detach(all\_data)

rm(station\_id)

attach(all\_data)

Now that we have the dataframe attached, we can just use the column names in our code:

plot(x=timestamp, y=growth)

Now clearly there’s something funny that’s being caused by the treatment of the timestamp as a factor. So let’s fix it by setting the timestamp column to be a POSIX timestamp data class. These timestamps are stored as the number of seconds since January 1, 1970, 00:00:00 UTC. As such, they make it easy for computers to be able to calculate the differences between timestamps and convert between timezones, etc. To convert our string representations (stored as factor levels) of the timestamp column to POSIX timestamps we are going to use the as.POSIXct() function. But first we need to convert our timestamp column to a character (instead of a factor):

new\_timestamp <- as.character(all\_data$timestamp)

head(new\_timestamp)

new\_timestamp <- as.POSIXct(new\_timestamp, format="%m/%d/%Y %H:%M", tz="GMT")

Note that we have to specify the format argument of our existing timestamp in order for the function to know which fields are separated by what. You can very flexibly convert between different styles of date/time representations using this feature (lookup ?strptime for specific format options). Let’s examine our new timestamp:

head(new\_timestamp)

class(new\_timestamp)

We can now do interesting operations on the new\_timestamp values (remember, values are in seconds so operations, additions, differences yield values in seconds):

new\_timestamp[1]

new\_timestamp[1] + 1

as.numeric(new\_timestamp[1]) + 1

new\_timestamp[1] + 60\*60\*24

new\_timestamp[2] - new\_timestamp[1]

Now we can re-plot our data and it will order the time series appropriately:

plot(x=new\_timestamp, y=growth)

Notice how we lost our box plot representation of all data for each date. This is the default plot() behavior when dealing with factors. If you want that back you can coerce the time representation back to a factor, but now R is smart enough to order the factors correctly since we are converting from a format it knows:

plot(x=as.factor(new\_timestamp), y=growth)

Since we are happy with the way the new\_timestamp works, let’s replace the timestamp in our data.frame and do a final plot comparing all of our data variables with one another in what’s called a pairs plot:

all\_data$timestamp = new\_timestamp

pairs(all\_data)

Last, while viewing the plot (plot window is on top) let’s save this plot using the “File > Save As..” and choose PDF. And let’s save off our all\_data data.frame as both a CSV file and RData object for future processing. Furthermore, we will save our R workspace so that we can restore every object currently in memory and restart our session whenever we like:

write.csv(all\_data, file="all\_data.csv")

save(all\_data, file="all\_data.RData")

save.image(file="Lab 2.RData")

Just for fun let’s also examine local spatial autocorrelation in the dataset. These data are time stamped so if we want to examine spatial autocorrelation we have to look at each individual time stamp. First let’s install and load a couple of packages:

install.packages(“lctools”)

library(lctools)

library(sp)

Then we need to create a spatial object by combining the data coordinates with the rest of our dataset:

coords <- cbind(all\_data$x, all\_data$y)

alldata\_spdf <- SpatialPointsDataFrame( coords=coords, data=all\_data)

We can then select just the last time stamp:

lastob <- alldata\_spdf[alldata\_spdf$timestamp==alldata\_spdf$timestamp[2400],]

We can visualize nitrogen in the last time stamped observation:

lm.palette <- colorRampPalette(c("green", "yellow", "orange", "red"), space = "rgb")

spplot(lastob, zcol="nitrogen", col.regions=lm.palette(5), main="Dissolved Organic Nitrogen")

Calculate local moran’s i:

lmoran <- l.moransI(coords, 8, lastob$growth)

This gives clusters of 0 (non-sig), 1(HH), 2(LL), 3(LH), 4(HL) values for each observation. We can then extract these clusters and plot:

lastob$lmorancluster <- lmoran$Cluster

spplot(lastob, zcol="lmorancluster", col.regions=c("grey", "red", "blue", "green", "orange"), main="Local Moran’s I Clusters", cuts=c(0,1,2,3,4,5))

**To submit your assignment, please Zip the source code file, data and PDF and attach the Zip file to the Canvas Assignment submission for this week.**

**Thanks!**