Programming in R: Handout 1

Step 1: Get your data into R.

- For this class, the command `source()` or `load()` will be used to load both data and code. Any file with the extension .r, .R, .RData, or .rda can be read this way.
- Although there are exceptions, in this class, we will use `source()` with .r and .R extensions. Use `load()` for the rest.
- Other commands include `read.table()` and `scan`. These are good for reading in data saved as comma separated value (csv) and text files.
- R can also read NetCDF files using the ncdf package.

Step 2: What does my data look like?

- Regardless of how you get your data in, you need to make sure you know what it looks like.
  1. You need to make sure you get what you expect.
  2. If you don’t know what to expect you should be able to explore the structure of your data.
- What do the following commands do? How does it relate to the “class” of the item?
  - `objects()`
  - `class()`
  - `length()`
  - `dim()`
  - `names()`, `rownames()`, `colnames()`
  - `summary()`

Step 3: Exploring named data:

- You ran the code `names(world.temp)` and got a list of names. I am just interested in data from Cairo.
- `class(world.temp$Cairo)`
- `length(world.temp$Cairo)`
- `summary(world.temp$Cairo)`
- `world.temp[,"Cairo"]` vs. `world.temp$Cairo`
  - What did this do? It took just the column of the data named “Cairo”

Step 4: Subsetting the data.

- Just like in the previous section, we can pull out certain rows, columns, or specific values from the data.
• What does 1:5 give you?

• What does cru.lat[1:5] give you?

• Compare dim(world.temp), dim(world.temp[,1:5]) to dim(world.temp[1:5,]). What about dim(world.temp[1:10,1:5])?
  - Which spot specifies the row? Which specifies the column?
  - What if there are more than one or two dimensions?

  * dim(cru.mean)
  * dim(cru.mean [1, , ])
  * dim(cru.mean [, 1 , ])
  * dim(cru.mean [, ,1 ])
  * In words, what would each of these subsets describe?

Step 5: Data creation and manipulation

• Create two vectors: x=1:20 and y=101:120

• What do we get when we do the following?
  - x + 5
  - x + y
  - 5*x
  - x*y
  - x%*%y
  - matrix(x, nrow=4, ncol=5)
  - matrix(x, nrow=5, ncol=5)
  - matrix(x, nrow=4, ncol=5, byrow=T)

• Math functions are mostly intuitive.
  - mean(x)
  - sd(x)
  - cor(x, y)
  - sin(x)

Step 6: Basic plotting

• plot(world.temp$Moscow)

• plot(world.temp$Moscow,type='l')

• plot(world.temp$Moscow, type='b')

• plot(world.temp$Moscow, type='b', pch=19)

• plot(world.temp$Moscow, world.temp$Seoul)
• plot(world.temp$Moscow, world.temp$Seoul, col='blue')
• plot(world.temp$Moscow, world.temp$Seoul, col='burlywood4',
  xlab='Moscow Temperature', ylab='Seoul Temperature',
  main='Burlywood is a funny name for a color')
• par(mfrow=c(1,2))
  plot(world.temp$Moscow, world.temp$Seoul, col='blue')
  plot(world.temp$Moscow, world.temp$Jakarta, col='green')
• par(mfrow=c(1,2))
  plot(world.temp$Moscow, world.temp$Seoul, col='blue', ylim=c(-5,30))
  plot(world.temp$Moscow, world.temp$Jakarta, col='green', ylim=c(-5,30))

  - A good plot should always include labels on the axis and a main title.
  - How do you learn all this?
    - ?par will get you more help than you ever wanted about plotting. Of note is
      * pch has options for your plotting character
      * lty gives options for line types
      * mfrow gives the number of plots in the window as (#in row, #in column)
      * The best bet for colors is to guess. All the obvious ones are there: black, blue, green, purple, orange, etc.
      · When that fails, R has functions that will generate unique colors for you. Try plot(1:10, col=rainbow(10))
      · If you still need more colors, try the command colors() but only if you have a lot of time.

Step 7: Other types of plots
• These plots almost always have the same options as the original plot command. For more information,
type ?boxplot, ?image.plot, etc.
  - boxplot(world.temp$HongKong)
  - boxplot(world.temp[,2:19], col='gray')
  - hist(world.temp$Mumbai)
  - quilt.plot(city.info[,"lon"],city.info[,"lat"],as.matrix(world.temp[,2:19]))
    map("world",add=T)
  - image.plot(cru.lon,cru.lat,cru.mean[,6])
    map("world",add=T)

Step 8: Dates
• When formatted correctly, dates in R can be really convenient. There are two classes of interest to us:
  POSIXct and POSIXlt.
• class(world.temp$date)
test = as.POSIXlt(world.temp$date)
  names(test)
  table(test$mon)
  january = which(test$mon == 0)
  plot(world.temp$Moscow[january], ylim=c(-30,30), col='blue', type='l')
  july = which(test$mon == 6)
  lines(world.temp$Moscow[july], ylim=c(-30,30), col='red')

• We can also do math with dates.
  world.temp$date[1000] - world.temp$date[500]

Step 9: for-loops and writing functions

• For-loop syntax is pretty simple.

  for (i in values)
    Do this!

  – For example:

    for (i in 1:10)
      print("My favorite number is:")
      print(i)

  – A more practical example:

    monthMeans=matrix(nrow=12, ncol=18)
    for (i in 1:12)
      keep = which(test$mon == (i - 1))
      monthMeans[i,] = colMeans(world.temp[keep,2:19], na.rm=T)
    matplot(monthMeans, type='l')

• Creating your own function is another way to simplify tasks you will do repeatedly.

  computeYearlyMean = function(time, dat)
    ## assume time was brought in as POSIXct vector
    ## assume dat is a vector
    plt = as.POSIXlt(time)
    yrs = unique(plt$year)
    means1 = sapply(yrs, function(i) mean(dat[plt$year==i], na.rm=T))
    return(means1)

  plot(computeYearlyMean(world.temp$date, world.temp$Singapore),
       type='l', ylab='Mean Temp', pch=19, ylim=c(0, 30))
points(computeYearlyMean(world.temp$date, world.temp$Jakarta), col='blue', pch=19, type='l')
points(computeYearlyMean(world.temp$date, world.temp$Boulder), col='green', pch=19, type='l')
points(computeYearlyMean(world.temp$date, world.temp$Paris), col='red', pch=19, type='l')