Gee whiz R is fun

Brian O’Meara

2023-02-14

This is an Rmarkdown document. It is an easy way to combine text, code, analyses, and figures. It’s a great way to write up your analyses and share them with others. I have caching turned on so this only reruns the things we need: once something is computed, it won’t be computed again unless something changes. To convert this to a word document, you can click the “knit” button if you’re doing this in RStudio or rmarkdown::render("main.Rmd") if you’re doing it in the console (as I recommend). There’s a new technology, quarto, which will replace Rmarkdown over the next few years.

knitr::opts\_chunk$set(echo = TRUE, cache=TRUE)

## Libraries

R is a language; it has basic functions baked in. But anyone can create new functions that make it even more useful. You often install them from a central repository using a command like:

install.packages("ape")

##   
## There is a binary version available but the source version is later:  
## binary source needs\_compilation  
## ape 5.6-2 5.7 TRUE

## installing the source package 'ape'

This installs a package called ape. You can then use it by calling:

library(ape)

So R knows to use them. You only need to install them once but call library each time; the following code does this for us automatically. This uses a workflow from <https://statsandr.com/blog/an-efficient-way-to-install-and-load-r-packages/> to make this easier for teaching.

# Package names  
packages <- c("datasauRus", "tidyverse", "magrittr", "ggplot2", "geodata", "rgbif", "ggmap", "ape", "datelife", "strap", "knitr", "rinat")  
  
# Where to install them from  
  
options(repos = c(  
 CRAN = 'https://cloud.r-project.org',  
 phylotastic = 'https://phylotastic.r-universe.dev')  
 )  
  
# Install packages not yet installed  
installed\_packages <- packages %in% rownames(installed.packages())  
if (any(installed\_packages == FALSE)) {  
 install.packages(packages[!installed\_packages])  
}  
  
# Packages loading  
invisible(lapply(packages, library, character.only = TRUE))

## Basic stats

Ok, so we’ve installed some packages. Let’s use them!

# We now have a dataset loaded into R. Let's get a quick view of the beginning of it:  
  
head(datasaurus\_dozen)

## # A tibble: 6 × 3  
## dataset x y  
## <chr> <dbl> <dbl>  
## 1 dino 55.4 97.2  
## 2 dino 51.5 96.0  
## 3 dino 46.2 94.5  
## 4 dino 42.8 91.4  
## 5 dino 40.8 88.3  
## 6 dino 38.7 84.9

# It actually has 12 datasets in it.   
  
dataset\_name\_vector <- unique(datasaurus\_dozen$dataset)  
  
print(dataset\_name\_vector)

## [1] "dino" "away" "h\_lines" "v\_lines" "x\_shape"   
## [6] "star" "high\_lines" "dots" "circle" "bullseye"   
## [11] "slant\_up" "slant\_down" "wide\_lines"

R started as a statistical language. It’s still very good at that. Let’s do some stats.

First, let’s make a place to store the results. There are more efficient ways to do this; this is the easiest to learn without getting into bad habits. A good thing to do is make an empty data frame (basically, a matrix like in Excel where each column can store a different kind of data, like numbers in one, character strings in another), with the right number of rows and columns, then fill them in. A common mistake is to grow a data frame as you need it, which is slow and inefficient.

dataset\_summaries <- data.frame(  
 dataset=rep(NA, length(dataset\_name\_vector)),   
 mean\_x=rep(NA, length(dataset\_name\_vector)),   
 mean\_y=rep(NA, length(dataset\_name\_vector)),   
 corr\_x\_y=rep(NA, length(dataset\_name\_vector)),  
 corr\_p\_value=rep(NA, length(dataset\_name\_vector))  
)

Now we can do a loop to fill in the data frame. This is a bit of a pain to do in Excel, but it’s easy in R. We’ll use a for loop, which is the most basic kind of loop. We’ll use the sequence() function to make a vector of numbers that we can use to index the dataset\_name\_vector: get the first element, second element, etc.. We’ll use the subset() function to get the data for each dataset.

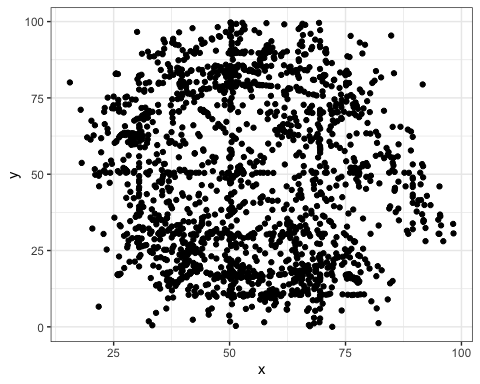
for (dataset\_index in sequence(length(dataset\_name\_vector))) { # dataset\_index will be 1, then 2, then 3, etc.  
 focal\_name <- dataset\_name\_vector[dataset\_index] # get the name of the dataset  
 focal\_dataset <- subset(datasaurus\_dozen, dataset==focal\_name) # get the data for that dataset  
 dataset\_summaries$dataset[dataset\_index] <- focal\_name # put the name in the data frame  
 dataset\_summaries$mean\_x[dataset\_index] <- mean(focal\_dataset$x) # put the mean x in the data frame  
 dataset\_summaries$mean\_y[dataset\_index] <- mean(focal\_dataset$y) # put the mean y in the data frame  
 dataset\_summaries$corr\_x\_y[dataset\_index] <- stats::cor(focal\_dataset$x, focal\_dataset$y) # put the correlation in the data frame  
 dataset\_summaries$corr\_p\_value[dataset\_index] <- stats::cor.test(focal\_dataset$x, focal\_dataset$y)$p.value # put the p-value in the data frame: this uses the cor.test function from the stats package, then gets the p.value from the result  
}  
  
print(dataset\_summaries) # Ugly but basic table plotting

## dataset mean\_x mean\_y corr\_x\_y corr\_p\_value  
## 1 dino 54.26327 47.83225 -0.06447185 0.4458966  
## 2 away 54.26610 47.83472 -0.06412835 0.4483288  
## 3 h\_lines 54.26144 47.83025 -0.06171484 0.4656268  
## 4 v\_lines 54.26993 47.83699 -0.06944557 0.4115226  
## 5 x\_shape 54.26015 47.83972 -0.06558334 0.4380777  
## 6 star 54.26734 47.83955 -0.06296110 0.4566492  
## 7 high\_lines 54.26881 47.83545 -0.06850422 0.4179063  
## 8 dots 54.26030 47.83983 -0.06034144 0.4756316  
## 9 circle 54.26732 47.83772 -0.06834336 0.4190029  
## 10 bullseye 54.26873 47.83082 -0.06858639 0.4173467  
## 11 slant\_up 54.26588 47.83150 -0.06860921 0.4171915  
## 12 slant\_down 54.26785 47.83590 -0.06897974 0.4146744  
## 13 wide\_lines 54.26692 47.83160 -0.06657523 0.4311664

What do you notice about the different datasets?

It is ALWAYS a good idea to plot data. We’ll use the ggplot2 package, which is a very powerful plotting package. It’s a bit complicated to learn, but it’s worth it (and it’s popular so there are lots of examples).

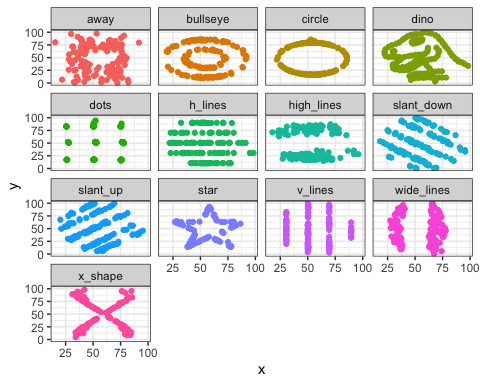
ggplot(datasaurus\_dozen, aes(x=x, y=y)) + geom\_point() + theme\_bw()



What does that scatter plot tell you about the data?

Hmm, let’s look by dataset:

ggplot(datasaurus\_dozen, aes(x = x, y = y, colour = dataset))+ # colour by dataset  
 geom\_point()+  
 theme\_bw()+  
 theme(legend.position = "none")+  
 facet\_wrap(~dataset, ncol = 4) # automatically split up into one subplot per dataset



Ah, looking at data was important! Even though the summary stats for the datasets are very similar, the data are very different.

Ok, so we’ve learned the basics of handling data in R. Let’s see what else we can do with R. Let’s focus on the area around Knoxville. We’re a salamander hotspot. What can we learn about them?

# Let's get info for a bounding box around Knoxville and the Smokies. I am putting in the coordinates by hand, but you could also get them from a map or do something like get the bounding box for the national park and Knox County and then take the union of the two. You want to minimize anytime you have to do something by hand, because it's easy to make mistakes (what if I mistyped the coordinates?).  
  
min\_lat <- 34.715593  
max\_lat <- 36.151430  
min\_lon <- -84.933809  
max\_lon <- -82.557402  
  
#elevations\_knoxville <- geodata::elevation\_3s(lon=-83.9, lat=36, path=tempdir())  
#elevations\_knoxville <- terra::crop(elevations\_knoxville, raster::extent(min\_lon, max\_lon, min\_lat, max\_lat))

## Salamanders

A huge advantage of R is that it’s easy to use other people’s code. We’ll use the rgbif package to get data from the Global Biodiversity Information Facility (GBIF). GBIF is a huge database of species occurrence data.

taxon\_key <- rgbif::name\_backbone(name="Caudata", kingdom="Animalia", rank="order")$usageKey # get the taxon key for salamanders: Caudata means taxon 953 within GBIF, but what if they change the number later? Avoid hard-coding numbers like this whenever possible.  
  
# Normally we wouldn't add line feeds, but it makes it easier to read the code. GBIF has a LOT of points, so we do things to limit to just the region we want  
  
salamander\_occurrences <- rgbif::occ\_search(  
 taxonKey=taxon\_key, # from above  
 country="US", # only get data from the US  
 decimalLongitude=paste0(min\_lon, ',', max\_lon), # GBIF can use a range like -65,-60 to get all data between -65 and -60  
 decimalLatitude=paste0(min\_lat, ',', max\_lat),   
 year='2022,2023', # Only getting salamander data from these two years  
 limit=10000 # limit to 10,000 points  
 )$data # rgbif::occ\_search returns a list, and we want the data part of the list  
  
salamander\_occurrences <- subset(salamander\_occurrences, !is.na(salamander\_occurrences$species) ) # clean up the data to remove ones that are just identified to family

We have gotten 1130 salamander observations [and that isn’t hardcoded – it’s a variable that is automatically put into the document!]

Let’s summarize that. We’re going to use some functions within the tidyverse universe of packages for this – it’s very handy for data analysis.

salamander\_summary <- salamander\_occurrences %>% group\_by(species, family) %>% summarise( n\_observations=n()) # summarize the data by species and family

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

knitr::kable(salamander\_summary) # print the data in a nice table

| species | family | n\_observations |
| --- | --- | --- |
| Ambystoma maculatum | Ambystomatidae | 28 |
| Ambystoma opacum | Ambystomatidae | 10 |
| Ambystoma talpoideum | Ambystomatidae | 6 |
| Aneides aeneus | Plethodontidae | 8 |
| Cryptobranchus alleganiensis | Cryptobranchidae | 10 |
| Desmognathus adatsihi | Plethodontidae | 16 |
| Desmognathus aeneus | Plethodontidae | 15 |
| Desmognathus balsameus | Plethodontidae | 2 |
| Desmognathus carolinensis | Plethodontidae | 6 |
| Desmognathus conanti | Plethodontidae | 16 |
| Desmognathus fuscus | Plethodontidae | 18 |
| Desmognathus imitator | Plethodontidae | 25 |
| Desmognathus marmoratus | Plethodontidae | 5 |
| Desmognathus monticola | Plethodontidae | 64 |
| Desmognathus ocoee | Plethodontidae | 30 |
| Desmognathus orestes | Plethodontidae | 3 |
| Desmognathus santeetlah | Plethodontidae | 1 |
| Desmognathus wrighti | Plethodontidae | 25 |
| Eurycea aquatica | Plethodontidae | 2 |
| Eurycea chamberlaini | Plethodontidae | 1 |
| Eurycea cirrigera | Plethodontidae | 35 |
| Eurycea guttolineata | Plethodontidae | 25 |
| Eurycea junaluska | Plethodontidae | 5 |
| Eurycea longicauda | Plethodontidae | 34 |
| Eurycea lucifuga | Plethodontidae | 2 |
| Eurycea wilderae | Plethodontidae | 117 |
| Gyrinophilus porphyriticus | Plethodontidae | 55 |
| Notophthalmus viridescens | Salamandridae | 155 |
| Plethodon amplus | Plethodontidae | 7 |
| Plethodon aureolus | Plethodontidae | 3 |
| Plethodon chattahoochee | Plethodontidae | 16 |
| Plethodon cheoah | Plethodontidae | 19 |
| Plethodon chlorobryonis | Plethodontidae | 3 |
| Plethodon cinereus | Plethodontidae | 3 |
| Plethodon cylindraceus | Plethodontidae | 15 |
| Plethodon glutinosus | Plethodontidae | 41 |
| Plethodon jordani | Plethodontidae | 92 |
| Plethodon metcalfi | Plethodontidae | 36 |
| Plethodon montanus | Plethodontidae | 4 |
| Plethodon serratus | Plethodontidae | 37 |
| Plethodon shermani | Plethodontidae | 28 |
| Plethodon teyahalee | Plethodontidae | 4 |
| Plethodon ventralis | Plethodontidae | 22 |
| Plethodon yonahlossee | Plethodontidae | 11 |
| Pseudotriton diastictus | Plethodontidae | 2 |
| Pseudotriton ruber | Plethodontidae | 68 |

### Images

Let’s get some images of the most common salamander in the area, Notophthalmus viridescens. We’ll use the rinat package to get data from the iNaturalist API.

obs <- rinat::get\_inat\_obs(query=salamander\_summary$species[which.max(salamander\_summary$n\_observations)], maxresults=50, bounds=c(min\_lat, min\_lon, max\_lat, max\_lon)) # southern latitude, western longitude, northern latitude, and eastern longitude  
  
knitr::kable(obs[1:10,c("place\_guess", "observed\_on", "user\_name")]) # print out the first ten values

| place\_guess | observed\_on | user\_name |
| --- | --- | --- |
| Jones Gap State Park, Marietta, SC, US | 2021-08-03 | Michael B. Mann, PhD |
| Pisgah National Forest, Horse Shoe, NC, US | 2023-01-30 |  |
| Flag Pond, TN 37657, USA | 2022-07-03 | Lincoln Durey |
| Flag Pond, TN 37657, USA | 2022-07-03 | Lincoln Durey |
| Old Newport Hwy, Sevierville, TN, US | 2022-12-31 |  |
| Cherokee National Forest, Tellico Plains, TN, US | 2017-08-20 | Tristan A. McKnight |
| North Carolina, US | 2022-12-13 |  |
| North Carolina, US | 2022-12-13 |  |
| Seven Islands State Birding Park, Kodak, TN, US | 2022-12-14 | John Williams |
| North Carolina, US | 2022-12-13 |  |

images <- obs[nchar(obs$image\_url)>0,] # get rid of observations that don't have images

Here’s one image. It’s from the Jones Gap State Park, Marietta, SC, US on 2021-08-03 by Michael B. Mann, PhD.



Here’s another image. It’s from the Pisgah National Forest, Horse Shoe, NC, US on 2023-01-30 by .



And a third image. It’s from the Flag Pond, TN 37657, USA on 2022-07-03 by Lincoln Durey.



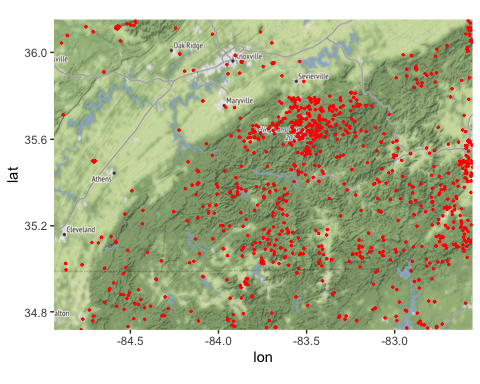
### Mapping

Let’s make a map of the salamander observations. We’ll use the ggmap package to get a map from the Stamen Maps API and then plot the points on top of it. This is all free, remember: we’re using free data and free software.

location <- c(min\_lon, min\_lat, max\_lon, max\_lat) #left, bottom, right, top, using variables from above  
  
knox\_area\_map <- get\_map(location, source="stamen") # stamen is free; google maps requires an API key

## ℹ Map tiles by Stamen Design, under CC BY 3.0. Data by OpenStreetMap, under ODbL.

plotted\_map <- ggmap(knox\_area\_map)  
plotted\_map <- plotted\_map + geom\_point(data=salamander\_occurrences, aes(x=decimalLongitude, y=decimalLatitude), colour="red", size=0.5)  
print(plotted\_map)

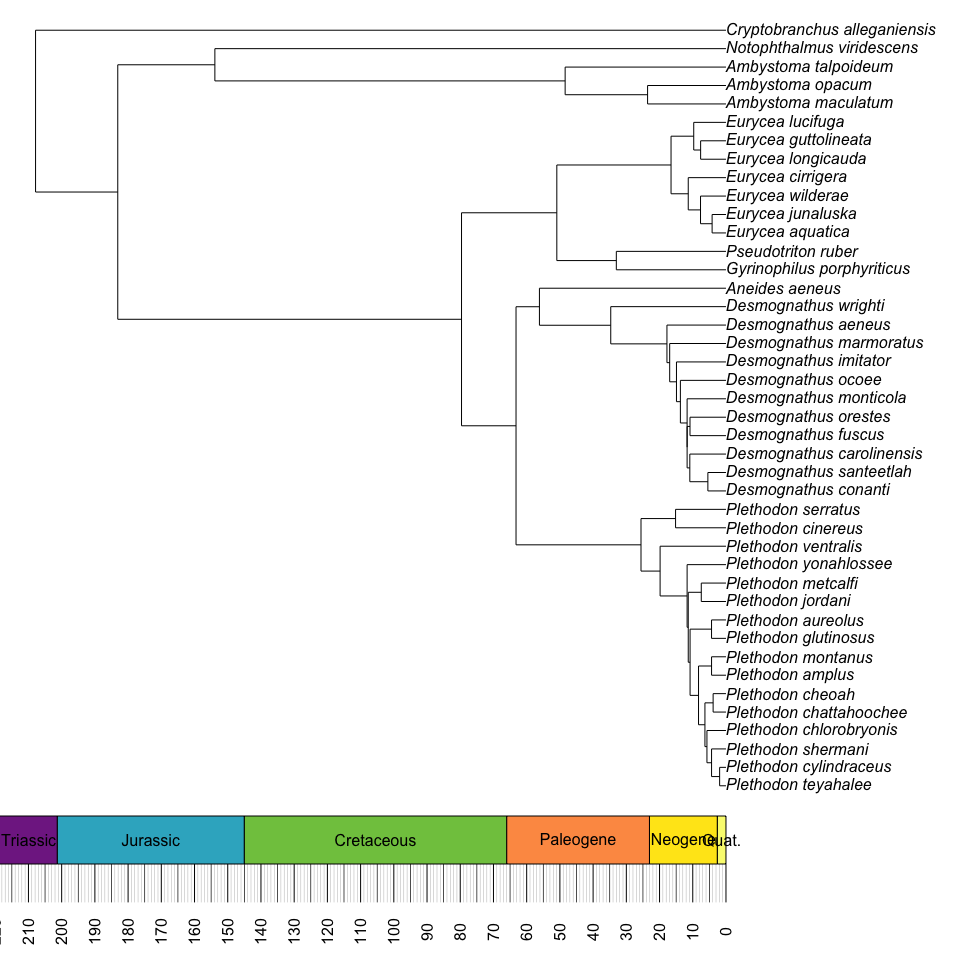


Just like the datasaurus above, if we wanted to we could facet by species to do one map per species. But that’s a lot of maps, so let’s just the overview.

### Phylogenetics

Let’s get a phylogenetic tree for the salamanders of the region. We’ll use the datelife package to get the tree.

datelife\_result <- datelife::datelife\_search(input=unique(salamander\_occurrences$species))  
salamander\_tree <- datelife\_result[[which.max(sapply(datelife\_result, ape::Ntip))]]  
salamander\_tree$root.time <- max(ape::branching.times(salamander\_tree))  
strap::geoscalePhylo(salamander\_tree, units="Period", cex.tip=1, cex.age=1, cex.ts=1)



In just our region, we have a tree for 42 species of salamander that were seen here since 2022. The tree is rooted at 208 million years ago; the total evolutionary history represented is 1.45 **billion** years.

This should give you a taste of what we can do in R. Just to review, in just the salamander part, all we entered was the name of the group (Caudata), and the dimensions of the area we wanted to look at. Everything else was done automatically. And it’s easy to extend this: pull in more images, get elevations of the points, load in information on the biome or climatic factors, find DNA sequences for the species, etc. And looping across species is easy, too.