BLUE
Biodiversity Literacy in Undergraduate Education

Introduction to R with Biodiversity Data

ML Gaynor

Contents

Chapter 1

General Information

1.1 Prerequisites

- Basics in file management and familiarity with data sheets (in Microsoft Excel, Google sheets, etc.)

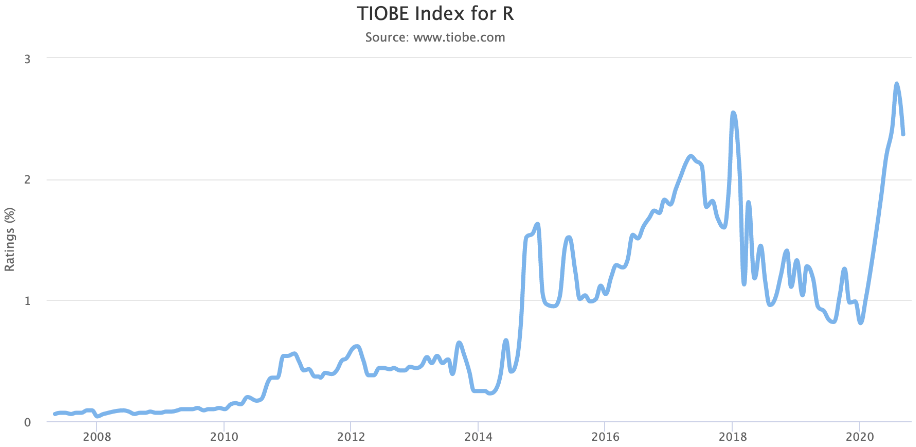
1.2 Learning objective

- Be able to navigate R Studio and write reproducible code.
- Execute lines of code, as well as complete scripts.
- Identify variables, functions, and operators.
- Approach basic troubleshooting.
- Know how to download biodiversity data using Application Programming Interfacen (API).

1.3 Introduction

Students will learn R basics while downloading biodiversity data from multiple data repositories. This module will walk students through installing R, navigating R,writing reproducible scripts in R, and using R to download biodiversity data.

1.4 Why R?



TIOBE Index for R

Source: www.tiobe.com

Figure 1.1: Graph of R ratings overtime, rating is based on commonly used search engines, (x-axis is year, y-axis is rating (%)) shows the rating of R increasing over time.

- R is a popular scripting language with readily available jobs (see jobs here: r-users.com)
- R allows for reproducibility
  - Every step is documented by each line of code in an analysis.
- R is free and open access

  - there is no associated cost
  - lots of community support for writing and troubleshooting code (ex. Overflow)

- R is interdisciplinary

- R can create beautiful figures

1.5 References

There are a lot of online resources for learning R. Throughout this activity we reference additional resources that may be useful. Below, we summarize the cited resources, as well as some additional references. We used many of these resources to create this activity.

General

- R Ecology Lesson
  - Introduction chapters for data carpentry ecology.
- RStudio Education
  - Resources for beginners by the creators of R Studio.
- R-bloggers
  - R news and tutorials by hundreds of R bloggers.
- R for cats and cat lovers
  - A cat -oriented way to learn R basics.
- R for Biologist NIMBioS
  - An introduction to R and descriptive statistics .
- RYouWithMe
  - Made by R-ladies Sydney. Fun tutorials on R basics, data cleaning, data visualization, and R markdowns.
- The Carpentries
  - “The Carpentries teaches foundational coding, and data science skills to researchers worldwide.” This organization hosts workshops and makes its lesson plans available freely for self-learning and reuse. Main lessons are divided into…”

- R for Data Science
  - “R for Data Science” textbook.
- Twitter for R programmers
  - Learn how to connect with the R community over twitter.

Tidyverse

Tidyverse is a must-have suite of packages (defined in section 3.3) for data wrangling and analysis that includes many packages. Important :

- dplyr
  - data manipulation and management
- tidyr
  - tidying your data
- ggplot2
  - data visualization

R based courses

- DataScienceBox
- STAT 545
- Data Carpentry for Biologist

Chapter 2

Setup

2.1 Learning Objectives

- Install R and RStudio.
- Describe the purpose of the RStudio Screen: Source Script & document, Console, Environment & History, and Files-Veiewer Panes.
- Be able to open and start an R Project, and understand the purpose of the working directory.

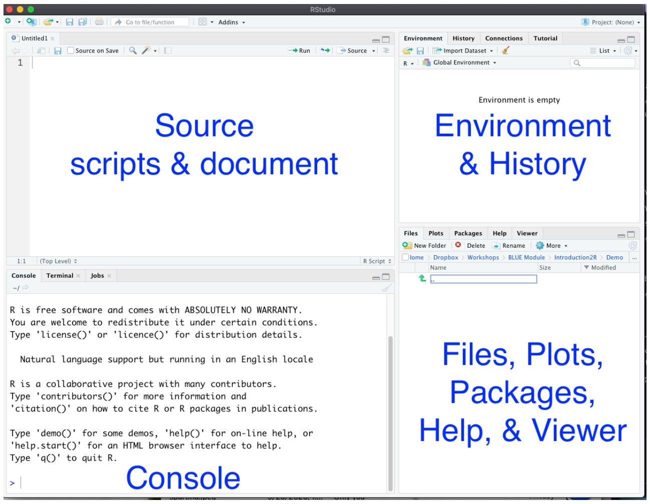
2.2 Install R and R Studio

1. Install R following the instructions on cran.rstudio.com.
2. Install R Studio, specifically the free version available on rstudio.com.

2.3 Introduction to RStudio

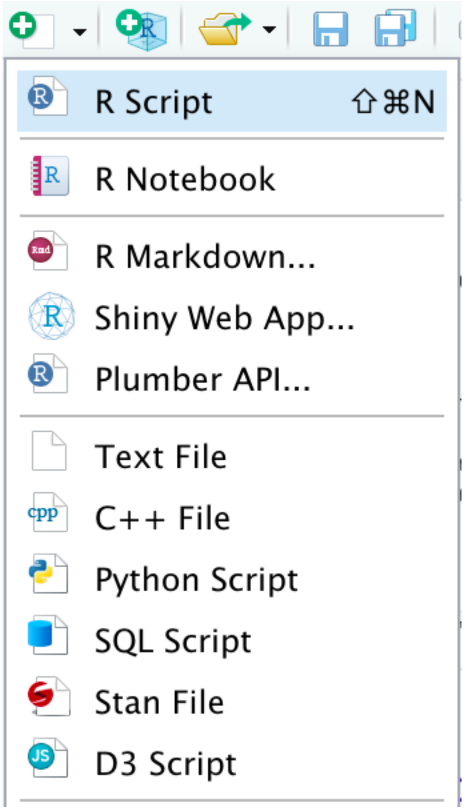
Once you have R and R studio installed, open R Studio and start to get orientated.

RStudio Screen



```
R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

  Natural language support but running in an English locale

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

>
```

- The **source** is where you can edit ".R" files or scripts, which document the lines of code you are using for your project. There are many ways to run code written in the source panel to the console (see more in Section 3.4). Other files can be viewed and edited in this panel, including those in the image below.



- The **console** is where you can run commands (or lines of code) and see any printed output.

- The **environment** tab shows active objects, such as a data file you read into R (see function read.csv). The **history** tab shows any past commands that were run during the current R session.

- The **file** tab shows any files in your working directory. The **plots** tab will show graphs and plots produced by running code. The **packages** tab will list all installed packages. Packages are units that contain a group of functions or commands for a specific purpose. For information about specific functions, you can search the **help** tab. The **viewer** can be used to visualize R Notebooks and R Markdowns (a fancy R script with markdown syntax, which make documents like this one).

    – Find out more about objects, commands, functions, and packages in section 3.3.

2.4 Creating your first project

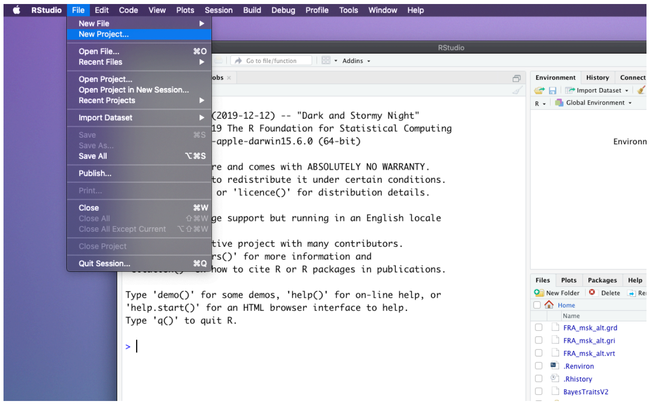
RStudio projects set R working directories. This allows all files associated with a single project to be stored in one location.

A **working directory** is a reference point that indicates the path to the files needed in your code. You can think of a directory path as a list of directions for the computer to follow to find or save files related to your project.

A project allows a working directory to be defined for your project. Within the project folder, you may make a folder called **scripts** and/or **data** for these files to be stored and organized.

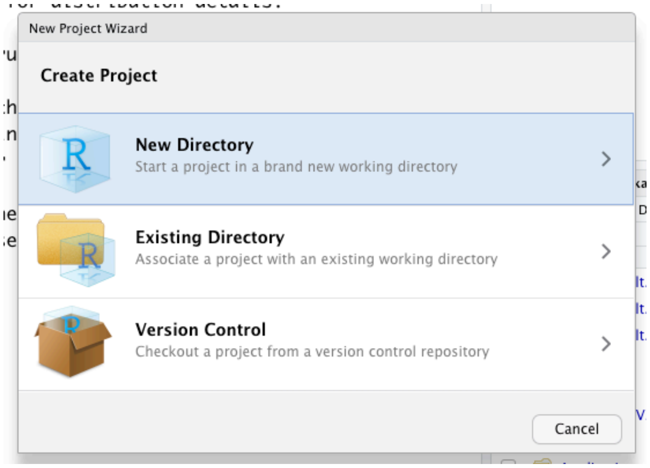
If you did not open a project in R, you may instead be using individual R scripts either within R Studio or R run on the command line. In these instances, you can check your working directory using `getwd()`. To set your working directory, you can use `setwd("/path/to/working/directory")`.

1. Open R studio, then click “File” and “New Project…”

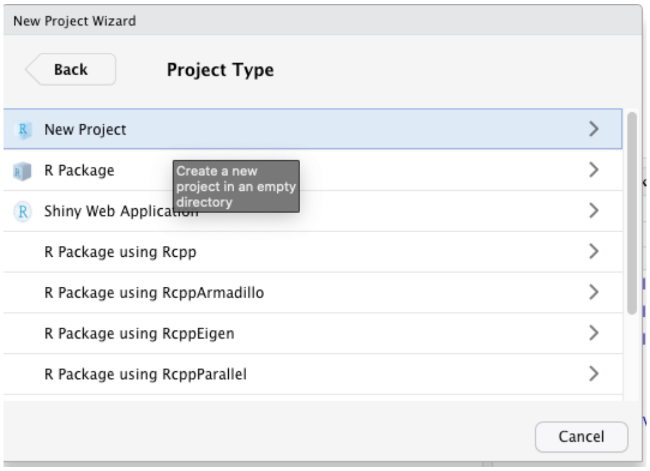


```
(2019-12-12) -- "Dark and Stormy Night"
19 The R Foundation for Statistical Computing
-apple-darwin15.6.0 (64-bit)

re and comes with ABSOLUTELY NO WARRANTY.
to redistribute it under certain conditions.
 or 'licence()' for distribution details.

ge support but running in an English locale

tive project with many contributors.
rs()' for more information and
how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

>
```

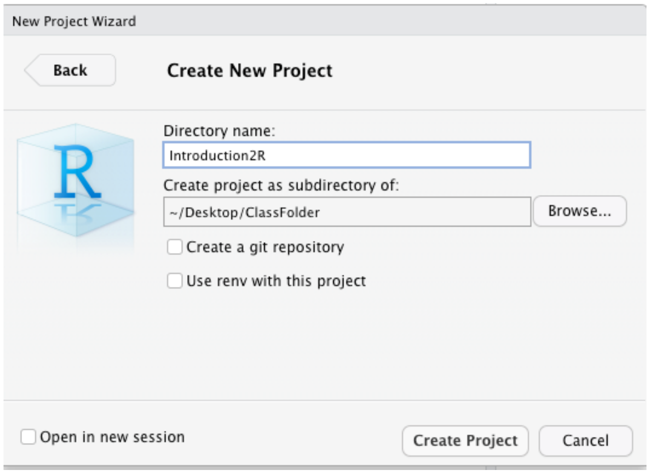
2. Choose “New Directory”


New Project Wizard
Create Project
New Directory
Start a project in a brand new working directory
Existing Directory
Associate a project with an existing working directory
Version Control
Checkout a project from a version control repository
Cancel

3. Choose “New Project”


New Project Wizard
Back
Project Type
New Project
R Package
Create a new project in an empty directory
Shiny Web Application
R Package using Rcpp
R Package using RcppArmadillo
R Package using RcppEigen
R Package using RcppParallel
Cancel

4. For this example, the “Directory name:” was set as “Introduction2R”. Make sure you always name your project with a pharse that is meaningful and indicates the purpose of the project. “Create project as subdirectory of” was set to “~/Desktop/ClassFolder” - this shows where the project will be saved on your computer.



Directory name: Introduction2R

Create project as subdirectory of: ~/Desktop/ClassFolder

☐ Create a git repository

☐ Use renv with this project

☐ Open in new session

Congrats! You made your first R project!
Test yourself: You now should be able to answer Question 1 in the assessment.

Chapter 3

Pre-Activity

In this pre-activity, we will go over some R Basics. At the end of this chapter, you will write your first R script!

3.1 Learning objective

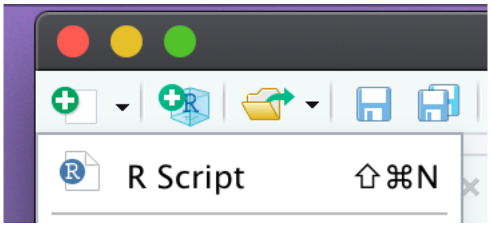
- Define objects, operators, functions, and R packages.
- Use the built-in RStudio help interface to search for more information on R functions.
- Run commands in the console and from a script.
- Demonstrate how to provide sufficient information for troubleshooting with the R user community.
- Be able to write R scripts in a reproducible manner.

3.2 R Scripts

- Below we summarize some R basics, but to find out more about R Scripts please review the link for some tips about:
    - Code Completion
    - Find and Replace
    - Extract Function

  - Comment/Uncomment
  - Executing Code

To open your first .R file or R Script, click the paper logo with the green plus sign OR “File” -> “New File” -> “R Script”.

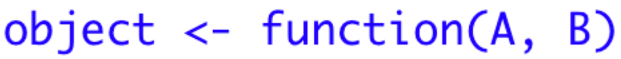


R Scripts are used to document the code you use in your project. These scripts can be used as a reference for yourself and allow others to reproduce your analysis. Learn more about running code from R Scripts in section 3.4.

Naming R Scripts

- Make sure you R Scripts file names are meaningful and end in .R.
- Avoid using special characters in file names. Instead use numbers, letters, dashes (-), and underscores (_).
- If files should be run in a particular order, prefix them with numbers. For example, 01_setup.R.

3.3 Important Terms



```
object <- function(A, B)
```

object: is what data is stored in your R environment. In the R programming language, this term is interchangeable with **variable**.

- Objects will have **classes** (eg. numeric, character, factor, logical)

```
weight <- 3
class(weight)
```

```
## [1] "numeric"
```

operator: assignment operators assign a value to an object. Assignment operators may be a back arrow <- or an equal sign =.

- Quick operator: For PC users, typing Alt + - (push the keys at the same time) will write `<-`. On a Mac, typing Option + - (push the keys at the same time) does the same.

command: the complete line shown above can be called a command, here an object is assigned a value. This value can be directly assigned (eg. x <- 10) or calculated through a function (see above).

function: a function is a set of processes that you can apply to an object or group of variables. .

R has base functions, or functions that come with it, like round:

```
weight <- 3.4759875
round(weight)
```

```
## [1] 3
```

If you need help with a specific function, let's say plot(), you can type:

```
?plot
```

You can also write your own functions. For example, if we have a weight in kilograms that we want to convert to pounds and we want to round the converted weight, we could write are own function to do this. Here the function we made is called convertkg2lb, this function will first convert kg to lb, then round this value, and finally return the rounded weight in lb. .

```
convertkg2lb <- function(weightkg){
  weightlb <- weightkg*2.20462
  rounded_weightlb <- round(weightlb)
  return(rounded_weightlb)
}

weight <- 3.4759875
convertkg2lb(weight)
```

```
## [1] 8
```

This function may not be very useful to write when we only have one weight to convert, however if you need to convert and round 100 weights, you can reduce the number of lines of codes in half by using a function.

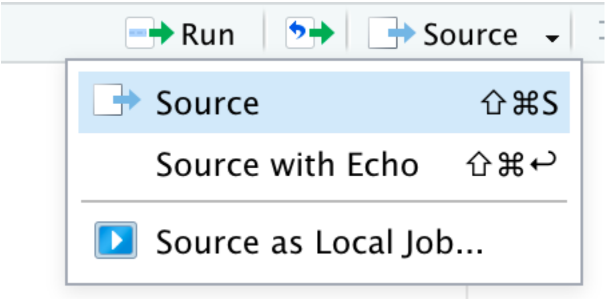
R package: is a group of functions. Offical R packages are avaliable via CRAN. For the class activity, we will be using packages from Tidyverse.

3.4 Running Code

You can type commands directly into the console and press `Enter` to execute those commands. Sadly, commands run in the console will be forgotten when you close the session.

To document your code and allow for reproducibility, it is better to type the commands we want in the script editor, and save the script. RStudio allows you to execute commands directly from the script editor by using on PC the `Ctrl` + `Enter` shortcut, and on Macs, `Cmd` + `Return`.

Individual lines of code can also be run by clicking the "Run" button or you can run the whole script by clicking "Source".



You can find other keyboard shortcuts in this RStudio cheatsheet about the RStudio IDE.

3.5 Basic Troubleshooting

1. Read the error message

- For example, if your message reads "incorrect field specification", check the function (?function) to make sure you specified the correct variables.

2. Google the error message

- Sometimes the error message will be confusing and reading it will not provide any insight. This is when google is helpful!
- If your error message is super generic, also include the name of the function or package when googling.

3. Ask for help

- If google did not answer your question, the next step would be to ask your classmates and/or instructor for help.
- Include the sessionInfo() - which prints the version of R, the packages loaded, and other useful information.

Where to ask for help?

- First start by ask your classmates and instructor.
- Stack Overflow: Check out this awesome blog post on "How do I ask a good question?" before posting.
- How to ask for R help: Make sure you are asking good questions!

3.6 Reproducibility

Documenting steps in your work flow should be done in a way that is reproducible. There are many R style guides to guide how you annotate and organize your code - check out the tidyverse style guide

Hashtags

Hashtags, #, can be used within your code to add comments or annotations. R treats # in a special way, it will not run anything that follows a hashtag on a line. We also call these hashtags *comment symbols*.

Titles

At the start of each .R script, your header should define the purpose of your code. This header should include the purpose of the script (why, what), as well as when the script has been updated. It should also include your name.

```
# Title of my document
## More information (why, what)
## Initial date: YYYY-MM-DD
## Update date: YYYY-MM-DD
## Name
```

Sections

I always start my scripts by loading packages (if the script includes any packages), this allows any lines to follow to use the functions in these packages.

```
# Load Packages
library(dplyr)
```

Each section should have a "#" hashtagged title that defines what the section includes. If a line in your script includes a hashtag, it is treated as text and not code. Following the initial title, you may want to include "##" followed by a description.

```
# List
## Create a list of characters
sites <- c("a", "b", "c", "d")
## Create a list of numbers
areas <- c(5, 12, 10, 11)
```

3.7 Basic Script Example

Below is a basic script. Open your own R script and write this script in your new file. Next, run each line.

Activity Goals

- Write your first R script. Name this script "BasicScript.R".
- Practice running lines of code.
- Be able to identify classes of objects.

- Identify what each function does and add comments to your script to indicate what each line does.

```
# Basic Script
## This is a basic script example
## 2020-10-02
## Name

# Objects
## Define objects of different classes
weight <- 3
class(weight) # numeric
weight <- 3L # integer
weight <- 3.5 # double
weight <- 3+2i # complex
hair <- TRUE # logical
hair <- "yellow" # character
hair = "brown"

# List
## Create a list of character
sites <- c("a", "b", "c", "d")
## Create a list of numbers
areas <- c(5, 12, 10, 11)

# Slice
### "give me a part of something"
### practice selecting from the lists
sites[1:3]
areas[3]

# Combining list
## c or combined
combine <- c(sites, areas)
combine
```

```
## rbind or row bind
combine_rbind <-  rbind(sites, areas)
combine_rbind

## cbind or column bind
### Here you have a command surrounded by parentheses -
### What happens? Run this line to find out!
(combine_cbind <- cbind(sites, areas))
combine_cbind

# Dataframes
## making a data frame
(xy <- data.frame(sites, areas))
xy

## Explore a data frame
str(xy)
head(xy)
View(xy)
xy$areas
class(xy$areas)
length(xy$areas)
nrow(xy)
ncol(xy)
```

Test yourself: You now should be able to answer Question 2 - 4 in the assessment.

Chapter 4

Class Activity

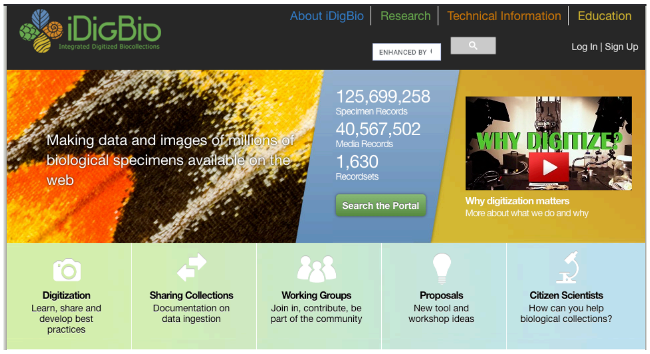
4.1 Learning Objective

- Demonstrate how to download biodiversity data through an Application Programming Interface.
- Plot occurrence data on a simple map.

4.2 Background information

4.2.1 iDigBio

iDigBio, or the Integrated Digitized Biocollections, is a biodiversity aggregator. It currently holds over 125 million specimen records and over 40 million media records.



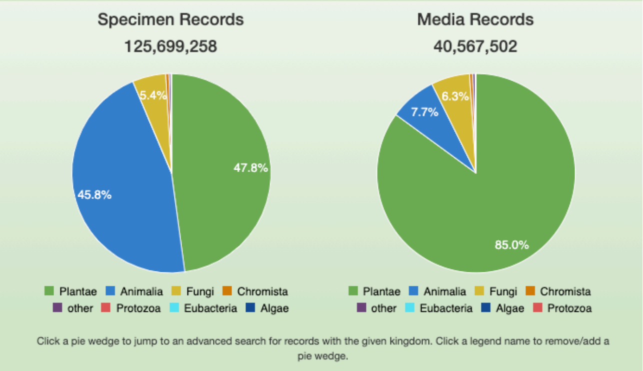
Making data and images of millions of biological specimens available on the web

125,699,258 Specimen Records
40,567,502 Media Records
1,630 Recordsets

Why digitization matters
More about what we do and why

Digitization: Learn, share and develop best practices
Sharing Collections: Documentation on data ingestion
Working Groups: Join in, contribute, be part of the community
Proposals: New tool and workshop ideas
Citizen Scientists: How can you help biological collections?

These specimen include mostly plants and animals. While media records include mostly plant specimen.

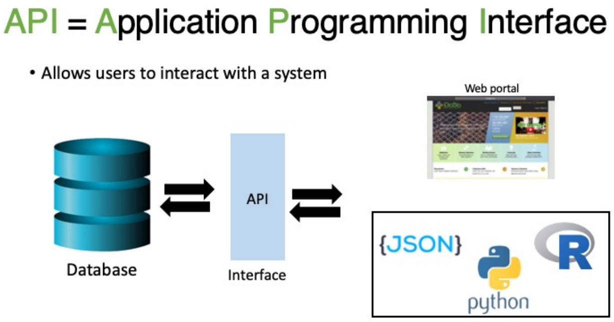


Click a pie wedge to jump to an advanced search for records with the given kingdom. Click a legend name to remove/add a pie wedge.

Here is an example of a plant media record:

4.2.1.1 Application Programming Interface

An API, or Application Programming Interface, allows a user to interact with a system that contains data. In this case, we are interacting with iDigBio, a biodiversity data aggregator. Even when we use the web portal for iDigBio, we are still interacting with the API. Here we will learn how to interact with the iDigBio API using R.


API = Application Programming Interface

- Allows users to interact with a system

4.2.1.2 *Spartina alterniflora*

Spartina alterniflora, smooth cordgrass, grows along shorelines throughout both the Atlantic and Gulf coasts of North America. In its native range, it is used in ecosystem restoration due to its extensive rooting capabilities.

4.3 Downloading Data

Set up your R project and R script

1. Open a new R project and name it "DataDownloading"
2. Within your R project, create an R script titled "DataDownloading.R"

DataDownloading.R

Install Packages

Do not include this in your R script! It is better to write this in the console than in our script for any package, as there's no need to re-install packages every time we run the scrip

```
install.packages(c("tidyverse", "ridigbio"))
```

The rest should be included in the script.

Load Packages

```
library(ridigbio)
library(ggplot2)
```

Data download from iDigBio

Here we use the idig_search_records function and the rq, or record query, indicates we want to download all the records where the scientificname is equal to *Spartina alterniflora*, or smooth cord grass.

```
iDigBio_SA <- idig_search_records(rq=list(scientificname="Spartina alterniflora"))
```

Inspect the downloaded records

How many observations are in this record set?
To determine this, we can use the nrow function to see the **n**umber of **row**s this data frame includes.

```
nrow(iDigBio_SA)
```

```
## [1] 1656
```

How many columns does this data frame include?
To determine this, we can use the ncol function to see the **n**umber of **col**umns this data frame includes.

```r
ncol(iDigBio_SA)
```

```
## [1] 13
```

What columns does this data frame include?
To print the **col**umns **names**, we can use the colname function.

```r
colnames(iDigBio_SA)
```

```
##  [1] "uuid"           "occurrenceid"   "catalognumber"  "family"
##  [5] "genus"          "scientificname" "country"        "stateprovince"
##  [9] "geopoint.lon"   "geopoint.lat"   "datecollected"  "collector"
## [13] "recordset"
```

Based on Darwin Core Standards, we know what each of these columns means.

Column	Description
uuid	Universally Unique IDentifier, https://tools.ietf.org/html/rfc4122
occurrenceid	identifier for the occurrence, http://rs.tdwg.org/dwc/terms/occurrenceID
catalognumber	identifier for the record within the collection, http://rs.tdwg.org/dwc/terms/catalogNumber
family	scientific name of the family, http://rs.tdwg.org/dwc/terms/family
genus	scientific name of the genus, http://rs.tdwg.org/dwc/terms/genus
scientificname	scientific name, http://rs.tdwg.org/dwc/terms/scientificName
country	country, http://rs.tdwg.org/dwc/terms/country
stateprovince	name of the next smaller administrative region than country, http://rs.tdwg.org/dwc/terms/stateProvince
geopoint.lon	equivalent to decimalLongitude, http://rs.tdwg.org/dwc/terms/decimalLongitude
geopoint.lat	equivalent to decimalLatitude,http://rs.tdwg.org/dwc/terms/decimalLatitude

Column	Description
datecollected	equivalent to eventDate, https://dwc.tdwg.org/terms/#dwc:eventDate)
collector	equivalent to recordedBy, http://rs.tdwg.org/dwc/terms/recordedBy
recordset	indicates the iDigBio recordset the observation belongs too

Plot the records

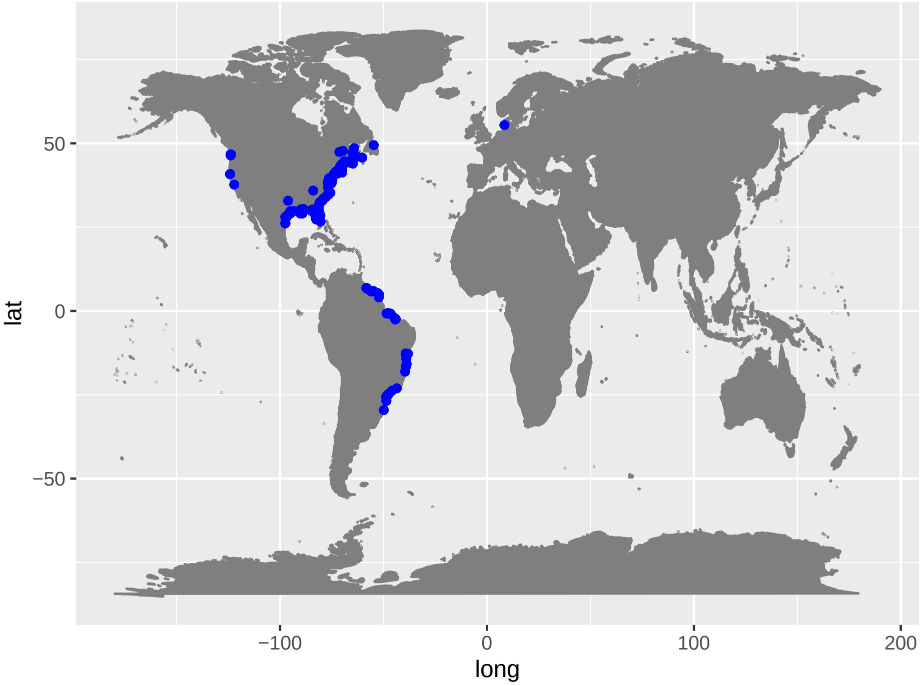
Now that we have inspected the data frame, let's plot the records. We will use ggplot2 to create a simple plot.

First, we need to download a basemap to plot our points on. Here I download a world map that is outlined and filled with the shade "gray50" using the function borders which is from the package ggplot2.

```
world <- borders(database = "world", colour = "gray50", fill = "gray50")
```

Next, we are going to create a plot using the base map. We will add the points from the iDigBio_SA data frame using the geom_point function. For ggplot2, you have to indicate mapping aesthetics or mapping = aes(). Within this statement we include x = geopoint.lon, y = geopoint.lat) to indicate that the x coordinate should be the column containig longitude (geopoint.lon), and the y coordinate should be the column containing latitude (geopoint.lat). Finally, we indicate we want these points to be blue, or col = "Blue" .

```
all_plot <- ggplot() +
            world +
            geom_point(iDigBio_SA,
                       mapping = aes(x = geopoint.lon, y = geopoint.lat),
                       col = "Blue")
all_plot
```



When you plot the occurrence records downloaded, you will notice points outside of the native range! *Spartina alterniflora* is very invasive in California and other parts of the world.

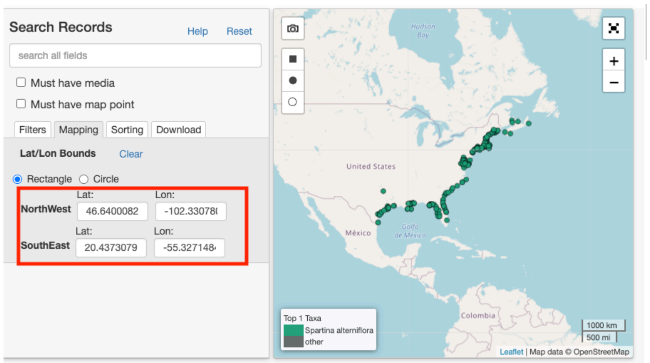
Save as a jpg: To save your map as an image file, you can easily run the following lines.

```
jpeg(file="map_spartina.jpeg")
all_plot
dev.off()
```

4.4 Limiting the extent

What if you want to only download points for the species within the native extent? Then we would need to define a bounding box to restrict our query (as coded by rq) too.

Hint: Use the iDigBio portal to determine the coordinates for the corners of the bounding box of your region of interest.



☐ Must have media

☐ Must have map point

Lat/Lon Bounds

● Rectangle ○ Circle

	Lat:	Lon:
NorthWest	46.6400082	-102.33078(
SouthEast	20.4373079	-55.327148

Set the search criteria

Here we will redefine the rq, or record query, to include a geo_bounding_box based on the iDigBio webportal - I limited the extent only to the native range.

```
rq_input <- list("scientificname"="Spartina alterniflora",
                 geopoint=list( type="geo_bounding_box",
                                top_left=list(lon = -102.33, lat = 46.64),
                                bottom_right=list(lon = -55.33, lat = 20.43) ) )
```

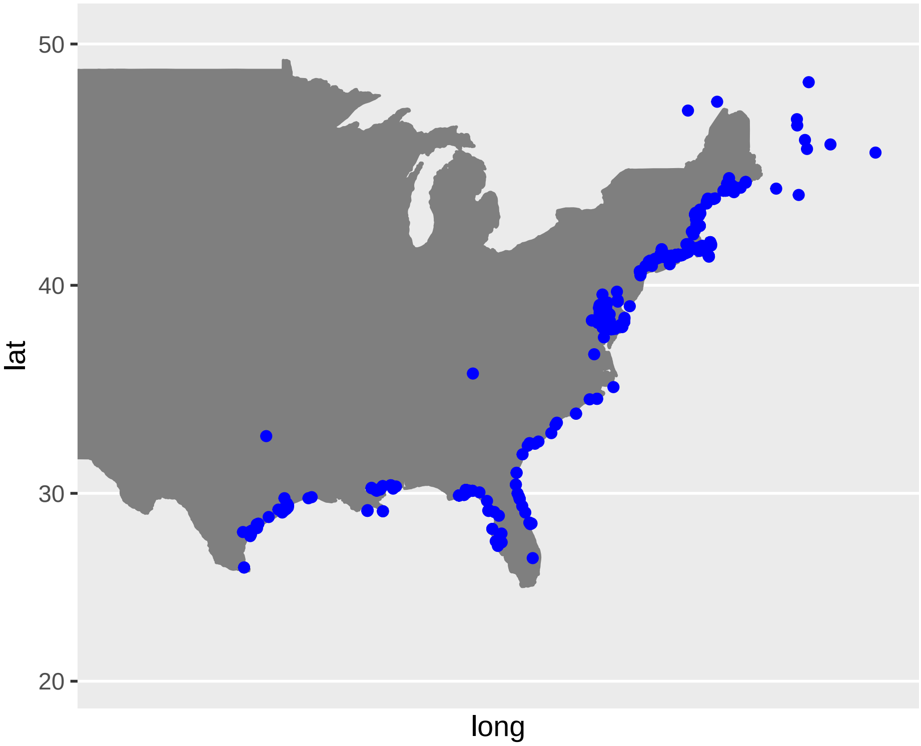
```
iDigBio_SA_limited <- idig_search_records(rq=rq_input)
```

Plot the records

Similar to before, we will use ggplot2 to plot our points. These records only include points within the desired extent!

Since the native range is only in the USA, I downloaded a USA map that is outlined and filled with the shade "gray50" using the function borders which is from the package ggplot2.

```
USA <- borders(database = "usa", colour = "gray50", fill = "gray50")
limited_plot <- ggplot() +
                USA +
                geom_point(iDigBio_SA,
                           mapping = aes(x = geopoint.lon, y = geopoint.lat),
                           col = "Blue") +
                coord_map(xlim = c(-60,-105), ylim = c(20, 50))

limited_plot
```



Congrats! You have made it through all the activities, you should now be able to answer all the questions in the assessment!

Chapter 5

Assessement

1. True or False: The best way to keep your files organized for analysis is to indicate and set the working directory in each R script.

```
# New Analysis Script
setwd("Desktop/CURE/Analysis/Data")
```

2. Fill in the operator used to create an object named "age" which contains the values 18, 19, 20, and 17:

```
age [_____] c(18, 19, 20, 17)
```

3. I have a vector that includes many different types of berries in a specific order (see below). How do I subset just "strawberries" from the object?

```
berries <- c("strawberries", "blueberries", "raspberries", "blackberries", "cranberries")
```

A. berries[1]
B. berries(1)
C. berries[,1]
D. berries[c(2)]

4. After reading a data frame into R using the following code, I want to explore my data.
mydata <- read.csv("data/mydata.csv").

Match the following functions with questions about the data:

Functions	Questions
str(mydata)	What is the first row in my data frame?
head(mydata)	What is the class of the row named row1?
view(mydata)	What are the first few rows in the data frame?
mydata[,1]	What is in the row named row1?
"mydata$row1"	What is the data frames structure?
"class(mydata$row1)"	What does my data frame look like?

5. **Activity**:
   Download specimen data for a species of interest using the iDigBio API in R following the steps learned in section 4.

a. What species did you download records for?
b. How many observations were in this record set?
c. Plot a map of these records and save this plot as a jpg.