SVP 2016

Jessica Blois, Edward Davis, Simon Goring, Jack Williams, Eric C. Grimm

Table of Contents

[Introduction 1](#_Toc464982909)

[Finding Data 2](#_Toc464982910)

[Explorer 2](#_Toc464982911)

[Getting Started 2](#_Toc464982912)

[Tips and tricks 2](#_Toc464982913)

[Search for Data 2](#_Toc464982914)

[View Data 4](#_Toc464982915)

[Download Data 5](#_Toc464982916)

[Web API- Application Programming Interface 5](#_Toc464982917)

[Example 1, text returns from the Paleobiology database. 5](#_Toc464982918)

[Example 2, JSON returns from iDigBio 6](#_Toc464982919)

[More about JSON 6](#_Toc464982920)

[More on the Neotoma DB API 9](#_Toc464982921)

[The neotoma Package 9](#_Toc464982922)

[Finding sites 9](#_Toc464982923)

[Getting Datasets 11](#_Toc464982924)

[Get\_Download 12](#_Toc464982925)

[Helper functions 16](#_Toc464982926)

[Age Models 18](#_Toc464982927)

[Multi-Site Analysis 19](#_Toc464982928)

[Conclusions 19](#_Toc464982929)

[References 19](#_Toc464982930)

# Introduction

This workshop will provide 1) guidance on best practices in the archiving and analysis of paleovertebrate data and 2) training in the use of the Neotoma Paleoecology Database (www.neotomadb.org) to archive, access, and analyze paleoecological data. Neotoma is a multiproxy paleodatabase that stores multiple kinds of paleoecological & paleoenvironmental data, including vertebrate faunal data. One of the strengths of Neotoma is the ability to compare faunal data with other proxy data such as fossil pollen, diatoms, ostracodes, insects, charcoal, and geochemical data. In addition, the database is structured to relate absolute dates to taxon occurrences and to allow the creation and storage of age models built on absolute dates from stratigraphic sections. Neotoma is a public-access, community-supported database that is emerging as the standard repository for Pliocene and Quaternary paleoecological data.

More teaching materials can be found [here](http://www.neotomadb.org/education/category/higher_ed/)

# Finding Data

## Explorer

### Getting Started

1. Go to <http://www.neotomadb.org/> and click on the 'Explorer' picture, or navigate directly to the [Explorer App](http://apps.neotomadb.org/Explorer/)
2. Pan (by dragging), or change the zoom so that your window is centered on North America, including all of the lower 48 states and the southern half of Canada.

### Tips and tricks

1. Show/Hide Search Results
   1. You often accumulate many search layers - it may be getting a bit confusing. Find the icon that lets you show/hide/combine search layers and use it to hide or delete some of your searches.
2. Rename searches
   1. Searches from the "Basic" search window are automatically named, but not "Advanced" searches. Add names for your searches as you go at the bottom of the Search dialog

### Search for Data

#### Finding sites

1. **Find a known site**
   1. Using the Search dialog window (Advanced tab, Metadata subtab, Site Name field), find the classic Guilday site "New Paris #4".
   * *Note: Search doesn't like the # symbol, so search for "New Paris", which should return two sites: New Paris #2, and New Paris #4*
   1. Once you've performed the search, click on the point that appears. A window will pop up with some information about this record.

* QUESTION 1: What is the latitude and longitude of New Paris #4? What is the Site ID?

**2. Explore sites by geography** a. Using the Search dialog window (Advanced tab), first choose "dataset type = vertebrate fauna" at the top b. Then, in the Space subtab, click "Search by extent" c. Click the "Extent" dropdown menu and search by shape, select the rectangle, then draw a rectangle on the map in your chosen region of North America

> QUESTION 2: How many total sites are found in that region?

**3. Find all sites produced by a researcher** a. Search Window, Advanced Tab, Metadata subtab, Person Name field b. Find all sites produced by Ernie Lundelius

> QUESTION 3: In which states has Ernie worked?

#### Find a Taxon

**1. Search for a single taxon.** a. Search Window, Basic Tab, use Taxon field b. Find all sites with *Antilocapra* records\*\*

> QUESTION 4: How many sites have \*Antilocapra\* records?

**2. Search for multiple taxa** a. The basic *Antilocapra* search indicates that we might need to build up the taxonomy. Let's re-search for *Antilocapra*, but make sure we've included all relevant records. b. Search Window, Advanced tab, Taxa subtab + *Note 1:* To do this, we will search for a "Taxa group". To the right of the "Taxon name" field, click on the gear symbol. Click on "Mammals" for the Taxa group, then search for *Antilocapra*. Then, click on all taxa you want to include here. For example: Antilocapra americana, Antilocapra sp., Antilocapra, cf. Antilocapra americana, cf Antilocapra sp., Antilocapra cf A. americana, Antilocapridae ?Antilocapra sp., and ?Antilocapra sp.

> QUESTION 5: What state has the easternmost location of \*Antilocapra\* in this search? (For comparison, the eastern range limit of \*Antilocapra\* today is in western Iowa).  
  
c. Add the modern species range onto the search.  
 + Click on the red polygon at the top, search for and select \*Antilocapra americana\*

#### Find all vertebrate records in North America

1. Search window, Advanced Tab, Dataset type = "Vertebrate fauna"

QUESTION 6: The generation of fossil records is labor intensive and hence expensive - *e.g.* the costs of fieldwork, the money spent on radiocarbon dates, the time required for a trained analysis to identify specimens, etc. A rough time/cost estimate for a single vertebrate fossil record is on the order of two years and $30,000. Given this, give an order-of-magnitude estimate of the number of person-years and dollars it took to generate these fossil vertebrate records now stored in Neotoma. (Order-of-magnitude = 10 person-years? 100 person-years? 1000? etc.)

#### Multi-Taxon Search

1. Find all sites with at least 20% *Picea* pollen between 15,000 and 12,000 years ago.
   * For *Picea*, first click on 'Dataset Type" = "pollen". Then use the 'Advanced Taxon Selection' which you can use by clicking on the gears icon to the right of the 'Taxon' field in the 'Search' window. Enter 'Taxa group' = Vascular Plant, search for Picea, click the box next to Taxon to check all taxa, then uncheck taxa like "Picea/Pinus" or "Picea/Abies".Then click the 'Abundance/density' box and select >20%. Fill in the appropriate age range, and choose records that 'intersects result age range'.
2. Find all sites with *Mammut* (mastodon) between 15,000 and 12,000 years ago.
   * For *Mammut*, use the 'Advanced Taxon Selection' which you can use by clicking on the gears icon to the right of the 'Taxon' field in the 'Search' window.
   1. In the ATS window, choose 'Mammals' for Taxa Group and then enter \*Mammut\* into the 'Search for' window. Then click 'Go'
   2. Note that the search returned taxon names for both \*Mammut\* (mastodon) and \*Mammuthus\* (mammoth). Click all boxes for all variants of \*Mammut\* but do not click the \*Mammuthus\* boxes
   3. Enter a search name (e.g. '\*Mammut\* - all') and click Save
   4. In the general search window, click 'Search'
3. QUESTION 6: Take a screenshot of this map and include it in your homework. Does mastodon tend to live in places with spruce, or without spruce? Suggest two hypotheses that might explain the observed association (or lack of one).

#### Multi-Time Search

1. Hide your previous searches.
2. Find all sites with \*Mammuthus\* between 21,000 and 18,000 years ago.
3. Find all sites with \*Mammuthus\* between 18,000 and 15,000 years ago.
4. Find all sites with \*Mammuthus\* between 15,000 and 13,000 years ago.
5. Find all sites with \*Mammuthus\* between 13,000 and 11,000 years ago.
6. Find all sites with *Mammuthus* between 11,000 and 8,000 years ago.
7. QUESTION 7: Take a screenshot of this map and include it in your homework. Describe the history of \*Mammuthus\* distributions in North America over the last 21,000 years.

### View Data

1. Using the last *Mammuthus* search (between 11,000 - 8,000 years ago), click on the single site that appears (Fetterman Mammoth Locality).
2. In the popup window with metadata for Fetterman, note that the bottom includes a list of datasets available at the site. (V = vertebrate; clock = geochronological data)
3. Hold the mouse over the vertebrate dataset. Note that an eyeball and '+' appear to the right.
4. Click on the eyeball to view the Fetterman Mammoth Locality vertebrate dataset. This opens up a new window with more detail about your dataset.
   * Explore the specimens found at the site. There is no associated geochronological dataset, but the chronology tab indicates an age for the mammoth. What other information would you want about this date to determine authenticity?
5. Samples: A data table. Each row is a different variable, the first few columns show metadata associated with each specimen, and each subsequent column is a stratigraphic depth. In this case, this is an assemblage so there is only one depth column.
6. Site: General information about the site
7. QUESTION 8: What publications are listed for Fetterman?
8. Now search for the site "Samwell Cave".
   * Notice that this site has multiple localities within it, and each locality has both a vertebrate dataset and a geochronological dataset.
   * Explore the Samwell Cave Popcorn Dome datasets further.
   * QUESTION 9: What is the difference between information in the "Chronology" tab of the Vertebrate dataset versus the information within the Geochronology dataset?

### Download Data

1. In the popup window with metadata for Samwell Cave Popcorn Dome, hold the mouse over the vertebrate dataset. Note that an eyeball and '+' appear to the right. Click on the '+'. This adds the dataset to a 'Datasets' tray.
2. Find the icon for the 'Saved Datasets Tray' and click on it.
3. Hover over the dataset then click on the Save icon to the right. The dataset will be saved as a text file in CSV (comma separated value) format.
4. Open the downloaded CSV file in Excel or a text editor (e.g. Notepad, Wordpad) to look at it.
5. QUESTION 10: When was the last time *Aplodontia rufa* was seen at the site?

# Web API- Application Programming Interface

What is a Web API?

* Set of protocols for building tools and applications that use a specific web service
* Often a structure for URL/URI formulation to make a query on a web database, such as Neotoma, iDigBio, or the Paleobiology Database.
  + URL = Uniform Resource Locator, or web address.
  + URI = Uniform Resource Identifier, or web address of a service, like an API.
* Calls are usually sent as URL “get” statements: values appended to a URL after a “?” that are processed as queries by a SQL or other web database.

For example, Google has APIs for most of its products. You could write an R (or Python, etc.) script that would connect to the Google Calendar API to allow you to automatically change events on your calendar or report agenda items back to you in your own custom environment.

APIs exist for many important biological and paleobiological databases. We will work through examples from GeoLocate, the georeferencing service, iDigBio, the Paleobiology Database, and Neotoma. Different databases/services produce different kinds of API returns. The simplest kinds are comma delimited text files, but many APIs are now returning JSON documents. We will begin with text and move on to JSON.

## Example 1, text returns from the Paleobiology database.

The full documentation for the Paelobiology Database API is located at <https://paleobiodb.org/data1.2/>

The PaleioBioDB API uses calls to different URLs to return different kinds of data. We’ll try some calls to the occurrence APIs.

In your web browser, try typing: <https://paleobiodb.org/data1.2/occs/list.txt?base_name=Camelidae&interval=Pleistocene&show=loc,class>

Note that this call is telling the API to search the base\_name, “Camelidae”; the interval, “Pleistocene”; and to return the attributes “loc,class”. In this API, “base\_name” searches for a taxonomic name at any level of the Linnaean hierarchy and returns any occurrences with that name, its synonyms, or its subtaxa. You should try changing to a name of interest to you and viewing the results.

Similarly, interval returns occurrences that fall within the specific geologic interval. These intervals include North American Land Mammal Ages as well as the conventional Geologic Timescale. Try “Blancan” and “Burdigalian” to see what happens. Try an interval of interest to you.

You can search by taxonomy, age, environment, even by the specific locality or occurrence ID in the database.

The list of potential calls is here: <https://paleobiodb.org/data1.2/occs/list_doc.html>

You can narrow your results geographically by invoking “lngmin” and “lngmax” as well as “latmin” and “latmax”. Try for a few minutes to explore the data with different calls, using the call list.

## Example 2, JSON returns from iDigBio

iDigBio (Integrated Digitized Biocollections) is the central resource for searching digital specimen data liberated by the NSF’s Advancing the Digitization of Biological Collections (ADBC) program. This program funds networks of collections centered on research themes (Thematic Collection Networks, TCNs) to enter specimen data into databases, to georeference collection locations, and to photograph those specimens. Most importantly, these data have to be made available online and connected to the iDigBio portal. iDigBio cleans up the data and provides several ways to access them, including the API we will be investigating.

First, try running this query:

<https://search.idigbio.org/v2/search/records/?rq=%7B%22scientificname%22%3A+%22camelus+bactrianus%22%2C+%22hasImage%22%3A+true%7D&limit=5>

There are several URL encoding tags in this statement. The %7B means {, %22 means “, the %3A means :, the %2C is a comma, and the %7D is }. The + means a space. So this statement translates as ?rq={“scientificname”: “camelus bactrianus”, “hasImage”: true}&limit=5. Try changing some of the values and seeing what you get.

This encoding formats the URL into what is called JSON, or JavaScript Object Notation. The output from this call is also in JSON. It is an important way to transmit the sort of complicated, nested data saved in relational databases when you have to send a flat file over the web.

The documentation for the iDigBio API is here: <https://www.idigbio.org/wiki/index.php/IDigBio_API>

## More about JSON

At its most basic level, JSON transmits data objects in attribute-value pairs. It has come to replace XML, which was the previous standard for this sort of data transmission.

JSON is composed of objects, enclosed by curly brackets, which may have any number of attributes named in quotes, with values after a colon, separated by commas. You may also present an array, or an ordered collection of values, enclosed in square brackets.

As an example, you could present an occurrence like this example from the Neotoma API:

{

"SiteLongitudeWest": -103.31666666666666,

"SiteLatitudeSouth": 34.283333333333339,

"TaxonName": "Smilodon fatalis",

"VariableElement": "bone/tooth",

"Value": 1.0,

"VariableContext": null,

"TaxaGroup": "MAM",

"SampleAgeYounger": 15332.0,

"SampleAgeOlder": 30041.0,

"SiteLongitudeEast": -103.31666666666666,

"SiteAltitude": 1280.0,

"VariableUnits": "present/absent",

"DatasetID": 4564,

"SampleAge": null,

"SiteLatitudeNorth": 34.283333333333339

}

Can you tell what kind of occurrence this JSON object is describing? How old is it? Where is it located?

This record comes from searching the Neotoma DB API. Here is the example API call:

[http://api.neotomadb.org/v1/data/sampledata?taxonname=Smilodon\*](http://api.neotomadb.org/v1/data/sampledata?taxonname=Smilodon*)

Notice when you make this call that the JSON is computer-friendly but not human-friendly. Try it again with the ‘pretty’ format tag:

[http://api.neotomadb.org/v1/data/sampledata?taxonname=Smilodon\*&format=pretty](http://api.neotomadb.org/v1/data/sampledata?taxonname=Smilodon*&format=pretty)

You should be able to see the nested set of JSON objects, including the occurrences returned as comma-separated objects within the array “data”.

Try experimenting with the search, substituting different names. We’ll do more with the Neotoma API in a bit.

JSON is becoming the standard for data transfer in web services. R has several packages for dealing with JSON-formatted data. We will use some examples from the package RJSONIO. We will also use the package RCurl, which has functions to let you query APIs from within the R environment.

Go ahead and install and load these two packages in your preferred R environment.

You will also want to make sure you have the latest version of R installed, or else the secure connection (hhtp***s***) won’t work in the following queries.

Now create a query for the PaleoBioDB API using the example from above:

library(RCurl)

library(RJSONIO)

q <- "https://paleobiodb.org/data1.2/occs/list.txt?base\_name=Camelidae&interval=Pleistocene&show=loc,class"

Create an object to receive the results:

a <- basicTextGatherer()

And execute that query:

curlPerform(url = q, writefunction = a$update)

Finally, view the data:

a$value()

You can see that the data have come in as a character vector; a long list of text strings with no clear structure. Luckily, the PaleoBioDB API also has a JSON interface.

Change your query to refer to list.json:

q <- "https://paleobiodb.org/data1.2/occs/list.json?base\_name=Camelidae&interval=Pleistocene&show=loc,class"

Rerun your query and look at a$value() again. How does it look now?

You can convert it to a data frame:

tmp <- fromJSON(a$value())

records <- tmp$records

results <- data.frame(records[1], stringsAsFactors = FALSE)

for(x in records[-1]){

x<-data.frame(x, stringsAsFactors = FALSE)

results <- merge(results, x, by = intersect(names(results),names(x)), all = TRUE)

}

When you have gotten this code to work, go through and make comments to describe what each section is doing. Remember, you can make comments in your R code by placing a hashtag (#) at the beginning of a line.

Try playing with your queries to see what you can pull from the PaleoBioDB, or even extending to the other two APIs we have explored.

## More on the Neotoma DB API

If you think back to the Neotoma API example, you can see that the example API call is reporting data from only part of the distributed database schema of Neotoma. In fact, the Neotoma API is designed around a set of different URLs, each of which allows a user to search a portion of the database. So, if you want the full Site information for locations with *Smilodon* present, you would have to search on *Smilodon* in the SampleData URI (as we did in the example), pull the DatasetID values from those returns, then search on the Dataset URI (api.neotomadb.org/v1/data/datasets) for those DatasetIDs, which would, in turn, produce the SiteIDs, which you would then search on the Sites URI (api.neotomadb.org/v1/data/sites). This sort of searching would be cumbersome if you were to do it by hand, but fortunately you can script a computer to do it for you. In fact, you don’t have to write the scripts to do it, because they have already been constructed and provided to the community as the R neotoma package.

Using this package will the the subject of our next module, but bear in mind that the other APIs also have wrapper packages to simplify data calls in R. The PaleoBioDB has a package, paleiobioDB, and iDigBio has a package, ridigbio. Currently, a large group of collaborators is working on a single API and wrapping R package to access both PaleoBioDB and Neotoma at the same time, as well as linking to many online museum databases and iDigBio. We wanted to introduce you to the underlying architecture here so you would understand what these packages are doing, and would know that you can crack them open and hack your own solutions if you cannot get them to give you the data or format of data that you need for your work.

# The neotoma Package

# Uncomment this line if you haven't already installed any of these packages:  
# install.packages(c("neotoma", "analogue"))  
  
#Add the neotoma package to your programming environment (we'll add analogue later)  
library(neotoma)

neotoma has three fundamental commands: get\_site, get\_dataset, get\_download. The first two return metadata for sites and datasets; the latter returns data. See Goring et al. (Goring et al. 2015) for a full description of the package and example code. This exercise is partially based on those examples.

## Finding sites

We'll start with get\_site. get\_site returns a data.frame with metadata about sites. You can use this to find the spatial coverage of data in a region (using get\_site with a bounding box), or to get explicit site information easily from more complex data objects. Use the command ?get\_site to see all the options available.

You can easily search by site name, for example, finding "Marion Lake". By default the search string is explicit, but because older sites, entered as part of COHMAP, often had appended textual information (for example (CA:British Columbia)), it's often good practice to first search using a wildcard character:

marion\_site <- get\_site(sitename = 'Marion%')

## The API call was successful, you have returned 3 records.

print(marion\_site)

## site.name long lat elev  
## Marion Lake (CA:British Columbia) -122.54722 49.30833 305  
## Marion Landfill -83.18611 40.59167 NA  
## Marion Lake -121.86241 44.55770 NA  
## A site object containing 3 sites and 8 parameters.

While marion\_site is a data.frame it also has class site, that's why the print output looks a little different than a standard data.frame. That also allows you to use some of the other neotoma functions more easily.

#Search by lat/lon bounding box. This one roughly corresponds to Florida.  
FL\_sites <- get\_site(loc = c(-88, -79, 25, 30))

## The API call was successful, you have returned 383 records.

You can also search by geopolitical name or geopolitical IDs (gpid) stored in Neotoma. For a list of names and gpids, go to **Error! Hyperlink reference not valid.**, or use the get\_table(table.name = "GeoPoliticalUnits") command. This command works either with an explicit numeric ID, or with a text string:

#get all sites in New Mexico (gpid=7956)  
NM\_sites <- get\_site(gpid = 7956)

## The API call was successful, you have returned 116 records.

#get all sites in Wisconsin  
WI\_sites <- get\_site(gpid = "Wisconsin")

## The API call was successful, you have returned 360 records.

data.frames store vectors of equal length. The nice thing about data.frames is that each vector can be of a different type (character, numeric values, *etc*.). In RStudio, you can use the Environment panel in upper right to explore variables.

We pointed out before that the object returned from get\_site is both a data.frame and a site object. Because it has a special print method some of the information from the full object is obscured when printed. You can see all the data in the data.frame using str (short for *structure*):

str(marion\_site)

## Classes 'site' and 'data.frame': 3 obs. of 8 variables:  
## $ site.id : int 1649 5873 13416  
## $ site.name : chr "Marion Lake (CA:British Columbia)" "Marion Landfill" "Marion Lake"  
## $ long : num -122.5 -83.2 -121.9  
## $ lat : num 49.3 40.6 44.6  
## $ elev : num 305 NA NA  
## $ description: chr "Lake in valley depression. Physiography: hilly and mountainous. Surrounding vegetation: western hemlock forest." NA NA  
## $ long.acc : num 0 0 0.0239  
## $ lat.acc : num 0 0 0.016

Let's look at the description field:

marion\_site$description

## [1] "Lake in valley depression. Physiography: hilly and mountainous. Surrounding vegetation: western hemlock forest."  
## [2] NA   
## [3] NA

## Getting Datasets

The structure of the Neotoma data model, as expressed through the API is roughly: "counts within downloads, downloads within datasets, datasets within sites". So a dataset contains more information than a site, about a particular dataset from that site. A site may have a single associated dataset, or multiple. For example:

get\_dataset(marion\_site[1,])

## A dataset\_list containing 2 objects:  
## Accessed from 2016-10-12 12:07h to 2016-10-12 12:07h.   
## Datasets:  
## dataset.id site.name long lat  
## 1705 Marion Lake (CA:British Columbia) -122.5472 49.30833  
## 8377 Marion Lake (CA:British Columbia) -122.5472 49.30833  
## type  
## pollen   
## geochronologic

#get\_dataset returns a list of datasets containing the metadata for each dataset  
#We can pass output from get\_site to get\_dataset  
marion.meta.dataset <- get\_dataset(marion.meta.site)  
#Let's look at the metadata returned for Marion Lake and Marion Landfill. Both  
#have a geochronology dataset, while one has a pollen dataset and the other a   
#vertebrate fauna dataset  
marion.meta.dataset

## Get\_Download

get\_download returns a list which stores a list of download objects - one for each retrieved dataset. Each download object contains a suite of data for the samples in that dataset. Get all datasets for both Marion Site and Marion Landfill. get\_download will accept an object of class dataset:

marion\_all <- get\_download(marion\_site)

## Fetching datasets for the site(s)

## Getting downloads:

## API call was successful. Returned record for Marion Lake(CA:British Columbia)  
## API call was successful. Returned record for Marion Lake(CA:British Columbia)

## The dataset ID 8377 is associated with a geochronology object, not count data.

## API call was successful. Returned record for Marion Landfill  
## API call was successful. Returned record for Marion Landfill

## The dataset ID 10248 is associated with a geochronology object, not count data.

## API call was successful. Returned record for Marion Lake  
## API call was successful. Returned record for Marion Lake

## Warning in get\_download.default(datasetid, verbose = verbose): Some datasetids returned empty downloads, be aware that length(datasetid) is longer than the download\_list.

print(marion\_all)

## A download\_list containing 4 objects:  
## Accessed from 2016-10-12 12:07h to 2016-10-12 12:07h.   
## Datasets:  
## dataset.id site.name long lat  
## 1705 Marion Lake (CA:British Columbia) -122.54722 49.30833  
## 7203 Marion Landfill -83.18611 40.59167  
## 20103 Marion Lake -121.86241 44.55770  
## 20104 Marion Lake -121.86241 44.55770  
## age.younger age.older type  
## 58 13051 pollen   
## 10340 10340 vertebrate fauna  
## -48 -48 water chemistry   
## -48 -48 diatom

## NULL

There are a number of messages that appear. These should be suppressed with the flag verbose = FALSE in the function call. One thing you'll note is that not all of the datasets can be downloaded directly to a download objct. This is because geochronologic datasets have a different data structure than other data, requiring different fields, and as such, they can be obtained using the get\_geochron function:

marion\_geochron <- get\_geochron(marion\_site)

## This dataset contains records that are not geochronological datasets. Only geochronological datasets will be returned.

## API call was successful.  
## API call was successful.

print(marion\_geochron)

## A geochronology\_list containing 2 objects:  
## Accessed from 2016-10-12 12:07h to 2016-10-12 12:07h.   
## Geochronologies:  
## id site.name long lat ages min  
## 8377 Marion Lake (CA:British Columbia) -122.54722 49.30833 7 520  
## 10248 Marion Landfill -83.18611 40.59167 1 10340  
## max interval  
## 12350 1690  
## 10340 0

## NULL

The result is effectively the inverse of the first.

#Get all datasets for just Marion Lake (BC):  
marion\_bc <- get\_download(marion\_site[1,])

## Fetching datasets for the site(s)

## Getting downloads:

## API call was successful. Returned record for Marion Lake(CA:British Columbia)  
## API call was successful. Returned record for Marion Lake(CA:British Columbia)

## The dataset ID 8377 is associated with a geochronology object, not count data.

## Warning in get\_download.default(datasetid, verbose = verbose): Some datasetids returned empty downloads, be aware that length(datasetid) is longer than the download\_list.

Within the download object, sample.meta stores the core depth and age information for that dataset. We just want to look at the first few lines, so are using the head function.

head(marion\_bc[[1]]$sample.meta)

## depth thickness age.older age age.younger chronology.name  
## 1 2.5 1 NA 58 NA NAPD 1  
## 2 7.5 1 NA 173 NA NAPD 1  
## 3 12.5 1 NA 289 NA NAPD 1  
## 4 17.5 1 NA 404 NA NAPD 1  
## 5 22.5 1 NA 520 NA NAPD 1  
## 6 27.5 1 NA 587 NA NAPD 1  
## age.type chronology.id sample.id dataset.id unit.name  
## 1 Radiocarbon years BP 791 33760 1705 NA  
## 2 Radiocarbon years BP 791 33761 1705 NA  
## 3 Radiocarbon years BP 791 33762 1705 NA  
## 4 Radiocarbon years BP 791 33763 1705 NA  
## 5 Radiocarbon years BP 791 33764 1705 NA  
## 6 Radiocarbon years BP 791 33765 1705 NA

#taxon.list stores a list of taxa found in the dataset  
head(marion\_bc[[1]]$taxon.list)

## taxon.name variable.units variable.element variable.context  
## 1 Abies NISP pollen NA  
## 2 Acer circinatum NISP pollen NA  
## 3 Alnus NISP pollen NA  
## 4 Amaranthaceae NISP pollen NA  
## 5 Asteraceae undiff. NISP pollen NA  
## 6 Betula NISP pollen NA  
## taxon.group ecological.group  
## 1 Vascular plants TRSH  
## 2 Vascular plants TRSH  
## 3 Vascular plants TRSH  
## 4 Vascular plants UPHE  
## 5 Vascular plants UPHE  
## 6 Vascular plants TRSH

#counts stores the the counts, presence/absence data, or percentage data for each taxon for each sample  
head(marion\_bc[[1]]$counts)

## Abies Acer circinatum Alnus Amaranthaceae Asteraceae undiff. Betula  
## 33760 6 5 390 2 3 23  
## 33761 5 1 197 1 4 2  
## 33762 9 0 165 0 3 13  
## 33763 4 0 138 3 0 3  
## 33764 10 0 196 0 4 4  
## 33765 13 0 244 0 2 6  
## Corylus Cupressaceae Ericaceae Lycopodiaceae Lysichiton Myrica  
## 33760 5 157 5 1 11 6  
## 33761 1 168 1 2 3 2  
## 33762 3 99 2 3 7 4  
## 33763 1 145 1 3 8 8  
## 33764 4 339 7 3 9 10  
## 33765 8 568 4 4 7 10  
## Nuphar Other plants Picea Pinus contorta-type Pinus monticola-type  
## 33760 1 30 5 28 18  
## 33761 1 3 1 12 8  
## 33762 1 6 2 12 8  
## 33763 1 3 4 11 8  
## 33764 0 1 18 87 9  
## 33765 0 4 14 88 7  
## Poaceae Potamogetonaceae Pseudotsuga Pteridium Rosaceae Salix  
## 33760 32 8 5 10 23 4  
## 33761 10 2 8 8 14 4  
## 33762 2 0 12 3 8 1  
## 33763 0 3 13 1 9 0  
## 33764 1 4 29 5 3 1  
## 33765 3 1 35 5 5 2  
## Sphagnum Tsuga heterophylla Tsuga mertensiana Unknown  
## 33760 2 78 4 6  
## 33761 0 52 2 3  
## 33762 0 94 3 2  
## 33763 0 98 4 6  
## 33764 1 200 7 7  
## 33765 2 202 7 2  
## Unknown (monolete) Artemisia Acer macrophyllum Cyperaceae Apiaceae  
## 33760 67 0 0 0 0  
## 33761 49 1 0 0 0  
## 33762 51 3 0 0 0  
## 33763 56 2 0 0 0  
## 33764 111 4 1 15 0  
## 33765 128 1 0 6 0  
## Selaginella wallacei  
## 33760 0  
## 33761 0  
## 33762 0  
## 33763 0  
## 33764 0  
## 33765 0

#lab.data stores any associated laboratory measurements in the dataset  
#For Marion Lake, this returns the Microsphere suspension used as a spike to calculate  
#concentrations  
head(marion\_bc[[1]]$lab.data)

##   
## 33760  
## 33761  
## 33762  
## 33763  
## 33764  
## 33765

## Helper functions

#### compile\_taxa

The level of taxonomic resolution can vary among analysts. Often for multi-site analyses it is helpful to aggregate to a common taxonomic resolution. The compile\_taxa function in neotoma will do this. To help support rapid prototyping, neotoma includes a few pre-built taxonomic lists, **however**, the function also supports the use of a custom-built data.frame for aligning taxonomies. Because new taxa are added to Neotoma regularly (based on analyst identification), it is worthwhile to check the assignments performed by the compile\_taxa function, and to build your own explicit compilation table.

marion\_bc <- compile\_taxa(marion\_bc, list.name = "P25")

## Warning in FUN(X[[i]], ...):   
## The following taxa could not be found in the existing conversion table:  
## Amaranthaceae  
## The following taxa could not be found in the existing conversion table:  
## Ericaceae  
## The following taxa could not be found in the existing conversion table:  
## Lycopodiaceae  
## The following taxa could not be found in the existing conversion table:  
## Lysichiton  
## The following taxa could not be found in the existing conversion table:  
## Myrica  
## The following taxa could not be found in the existing conversion table:  
## Nuphar  
## The following taxa could not be found in the existing conversion table:  
## Other plants  
## The following taxa could not be found in the existing conversion table:  
## Potamogetonaceae  
## The following taxa could not be found in the existing conversion table:  
## Pteridium  
## The following taxa could not be found in the existing conversion table:  
## Rosaceae  
## The following taxa could not be found in the existing conversion table:  
## Sphagnum  
## The following taxa could not be found in the existing conversion table:  
## Unknown  
## The following taxa could not be found in the existing conversion table:  
## Unknown (monolete)  
## The following taxa could not be found in the existing conversion table:  
## Apiaceae  
## The following taxa could not be found in the existing conversion table:  
## Selaginella wallacei

You'll notice that warning messages return a number of taxa that cannot be onverted using the existing data table. Are these taxa important? They may be important for you. Check to see which taxa have been converted by looking at the new taxon table:

marion\_bc[[1]]$taxon.list[,c("compressed", "taxon.name")]

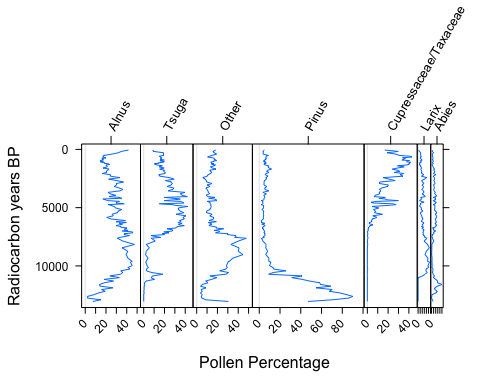
## compressed taxon.name  
## 1 Abies Abies  
## 2 Acer Acer circinatum  
## 3 Alnus Alnus  
## 4 Other Amaranthaceae  
## 5 Prairie Forbs Asteraceae undiff.  
## 6 Betula Betula  
## 7 Corylus Corylus  
## 8 Cupressaceae/Taxaceae Cupressaceae  
## 9 Other Ericaceae  
## 10 Other Lycopodiaceae  
## 11 Other Lysichiton  
## 12 Other Myrica  
## 13 Other Nuphar  
## 14 Other Other plants  
## 15 Picea Picea  
## 16 Pinus Pinus contorta-type  
## 17 Pinus Pinus monticola-type  
## 18 Poaceae Poaceae  
## 19 Other Potamogetonaceae  
## 20 Larix Pseudotsuga  
## 21 Other Pteridium  
## 22 Other Rosaceae  
## 23 Salix Salix  
## 24 Other Sphagnum  
## 25 Tsuga Tsuga heterophylla  
## 26 Tsuga Tsuga mertensiana  
## 27 Other Unknown  
## 28 Other Unknown (monolete)  
## 29 Prairie Forbs Artemisia  
## 30 Acer Acer macrophyllum  
## 31 Cyperaceae Cyperaceae  
## 32 Other Apiaceae  
## 33 Other Selaginella wallacei

And note that if you look at the names of the objects in the new download (using names(marion\_bc[[1]])), there is now a full.counts object. This allows you to continue using the original counts, while also retaining the new compiled counts.

#### Plotting

There are several options for plotting stratigraphic data in R. The rioja package (Juggins 2015) and analogue (Simpson and Oksanen 2016) each have methods, and other possibilities exist. Here we will show simple plotting using the analogue package. To make it clear which functions come from the analogue package I will use analogue:: before the function names. This is just an explicit way to state the function source. If you choose not to do this you will not encounter any problems unless multiple packages have similarly name functions.

library("analogue")  
  
# Convert the Marion Lake pollen data to percentages  
marion\_bc\_pct <- analogue::tran(x = marion\_bc[[1]]$counts, method = 'percent')  
  
# Drop rare taxa:  
marion\_bc\_pct <- marion\_bc\_pct[, colMeans(marion\_bc\_pct, na.rm = TRUE) > 2]  
  
analogue::Stratiplot(x = marion\_bc\_pct[ , order(colMeans(marion\_bc\_pct, na.rm = TRUE),   
 decreasing = TRUE)],   
 y = marion\_bc[[1]]$sample.meta$age,  
 ylab = marion\_bc[[1]]$sample.meta$age.type[1],  
 xlab = " Pollen Percentage")



# Age Models

We can use the Neotoma package, along with other tools to rebuild age models. Many of the age modeling tools require additional files to be installed. One such program is [Bacon](http://chrono.qub.ac.uk/blaauw/bacon.html) [@]. Here we will rebuild a Baysian age model for a site in Wisconsin.

We've looked in detail at Marion Lake, let's choose another site from our pool of Wisconsin sites earlier. First we need to get Bacon set up:

*Jessica Note: section deleted from this document. See template to recover*

# Multi-Site Analysis

# Conclusions

# References

Goring, S., A. Dawson, G. L. Simpson, K. Ram, R. W. Graham, E. C. Grimm, and J. W. Williams. 2015. neotoma: A programmatic interface to the neotoma paleoecological database. Open Quaternary 1:2.

Juggins, S. 2015. Rioja: Analysis of quaternary science data.

Simpson, G., and J. Oksanen. 2016. Analogue: Analogue matching and modern analogue technique transfer function models.