ALA4R

The Atlas of Living Australia (ALA) provides tools to enable users of biodiversity information to find, access, combine and visualise data on Australian plants and animals; these have been made available from http://ala.org.au/ (http://ala.org.au/). Here we provide a subset of the tools to be directly used within R.

ALA4R enables the R community to directly access data and resources hosted by the ALA. Our goal is to enable outputs (e.g. observations of species) to be queried and output in a range of standard formats.

Installing ALA4R

Windows

In R:

```
install.packages("devtools")
library(devtools)
install_github("AtlasOfLivingAustralia/ALA4R")
```

You may see a warning about the `Rtools` package: you don't need to install this. You may also be asked about a location for the `R.cache` directory — choose whatever you prefer here, ALA4R does not use `R.cache`.

If you wish to use the `data.table` package for potentially faster loading of data matrices (optional), also do:

```
install.packages("data.table")
```

Linux

First, ensure that `libcurl` is installed on your system — e.g. on Ubuntu, open a terminal and do:

```
sudo apt-get install libcurl4-openssl-dev
```

or install `libcurl4-openssl-dev` via the Software Centre.

Then follow the instructions for Windows, above.

Using ALA4R

The ALA4R package must be loaded for each new R session:

```
library(ALA4R)
```

Customizing

Various aspects of the ALA4R package can be customized.

Caching

ALA4R can cache most results to local files. This means that if the same code is run multiple times, the second and subsequent iterations will be faster. This will also reduce load on the ALA servers.

By default, this caching is session-based, meaning that the local files are stored in a temporary directory that is automatically deleted when the R session is ended. This behaviour can be altered so that caching is permanent, by setting the caching directory to a non-temporary location. For example, under Windows, use something like:

```
ala_config(cache_directory=file.path("c:","mydata","ala_cache")) ## Windows
```

or for Linux:

```
ala_config(cache_directory=file.path("~","mydata","ala_cache")) ## Linux
```

Note that this directory must exist (you need to create it yourself).

All results will be stored in that cache directory and will be used from one session to the next. They won't be re-downloaded from the server unless the user specifically deletes those files or changes the caching setting to "refresh".

If you change the cache_directory to a permanent location, you may wish to add something like this to your .Rprofile file, so that it happens automatically each time the ALA4R package is loaded:

```
setHook(packageEvent("ALA4R", "attach"), function(...) ala_config(cache_directory=file.pa
th("~","mydata","ala_cache")))
```

Caching can also be turned off entirely by:

```
ala_config(caching="off")
```

or set to "refresh", meaning that the cached results will re-downloaded from the ALA servers and the cache updated. (This will happen for as long as caching is set to "refresh" — so you may wish to switch back to normal "on" caching behaviour once you have updated your cache with the data you are working on).

User-agent string

Each request to the ALA servers is accompanied by a "user-agent" string that identifies the software making the request. This is a standard behaviour used by web browsers as well. The user-agent identifies the user requests to the ALA, helping the ALA to adapt and enhance the services that it provides. By default, the ALA4R user-agent string is set to "ALA4R" plus the ALA4R version number, R version, and operating system (e.g. "ALA4R 0.16 (R version 3.0.2 (2013-09-25)/x86_64-pc-linux-gnu)").

NO personal identification information is sent. You can see all configuration settings, including the the user-agent string that is being used, with the command:

```
ala_config()
```

Debugging

If things aren't working as expected, more detail (particularly about web requests and caching behaviour) can be obtained by setting the `verbose` configuration option:

```
ala_config(verbose=TRUE)
```

Setting the download reason

ALA requires that you provide a reason when downloading occurrence data (via the ALA4R `occurrences()` function). You can provide this as a parameter directly to each call of `occurrences()`, or you can set it once per session using:

```
ala_config(download_reason_id=your_reason_id)
```

(See `ala_reasons()` for valid download reasons)

Other options

If you make a request that returns an empty result set (e.g. an un-matched name), by default you will simply get an empty data structure returned to you without any special notification. If you would like to be warned about empty result sets, you can use:

```
ala_config(warn_on_empty=TRUE)
```

Example usage

First, check that we have some additional packages that we'll use in the examples, and install them if necessary.

```
to_install=c("plyr","jpeg","phytools","ape","leafletR","vegan","mgcv","geosphere","maps",
"mapdata","maptools")
to_install=setdiff(to_install,installed.packages()[,"Package"])
if(length(to_install)) install.packages(to_install)
```

We'll use the `plyr` package throughout these examples, so load that now:

```
library(plyr)
```

Example 1: Name searching and taxonomic trees

```
library(ape)
library(phytools)
```

We want to look at the taxonomic tree of penguins, but we don't know what the correct scientific name is, so let's search for it:

```
sx=search_fulltext("penguins")
(sx$data[,c("name","rank","score","commonName")])
```

```
##                                    name       rank     score
## 1                          SPHENISCIDAE     family 7.799e+00
## 2                       Eudyptula minor    species 6.826e-01
## 3                    Eudyptes chrysocome    species 2.968e-09
## 4       Eudyptes pachyrhynchus robustus subspecies 2.570e-09
## 5                          Pteria penguin    species 1.260e-09
## 6                 Eudyptes pachyrhynchus    species 8.409e-10
## 7                  Eudyptes chrysolophus    species 8.409e-10
## 8  Eudyptes pachyrhynchus pachyrhynchus subspecies 8.403e-10
## 9                       Eudyptes sclateri    species 6.308e-10
## 10               Aptenodytes patagonicus    species 2.615e-10
##
```

```
                                                                                commonName
## 1
```

```
                                                                                  Penguins
## 2   Blue Penguin,  Fairy Penguin, Fairy Penguin, Little Penguin, Little Penguin In The
 Manly Point Area (being The Area On And Near The Shoreline From Cannae Point Generally N
orthward To The Point Near The Intersection Of Stuart Street And Oyama Cove Avenue, And E
xtending 100 Metres Offshore From That Shoreline), Little Penguin, Little Blue Penguin
## 3
```

```
                                                                                Crested Pen
guin, Jackass Penguin, Tufted Penguin, Rockhopper Penguin, Southern Rockhopper Penguin
## 4
```

```
                      Snares Crested Penguin,  Snares Islands Penguin, Snares Penguin
## 5
```

```
                                  Black Banded Winged Pearl Shell, Penguin Wing Oyster
## 6
```

```
                                         Fiordland Crested Penguin, Fiordland Penguin
## 7
```

```
                                                  Macaroni Penguin, Royal Penguin
## 8
```

```
                                         Fiordland Crested Penguin, Fiordland Penguin
## 9
```

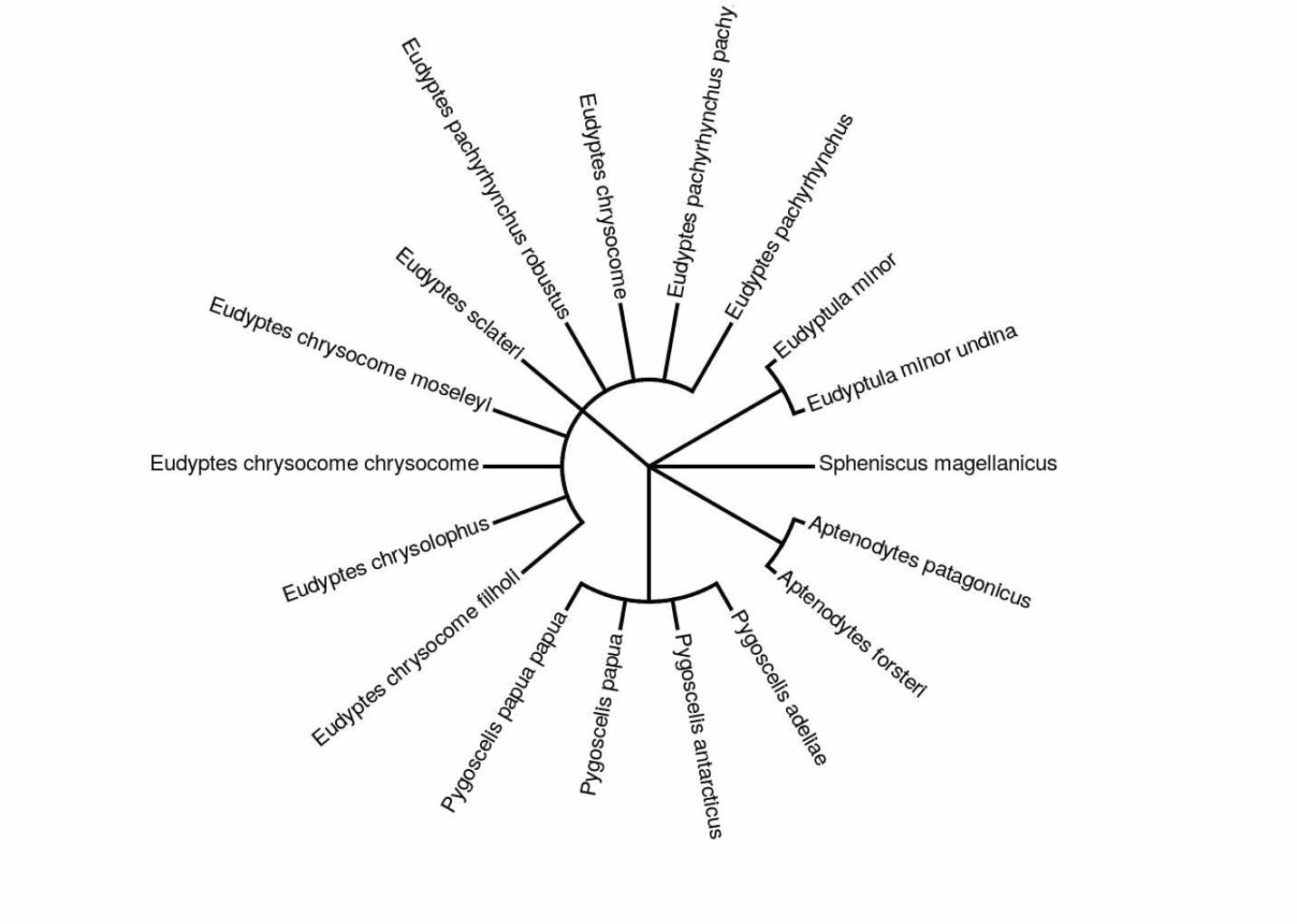
```
                                                  Big-crested Penguin, Erect-crested Penguin
## 10


                                                                                King Penguin
```

And we can see from the first result that penguins correspond to the family Spheniscidae. Now we can download the taxonomic data (note that the search is case-sensitive, so “SPHENISCIDAE” must appear as it does in the search results above):

```
tx=taxinfo_download("family:SPHENISCIDAE",fields=c("guid","genus","nameComplete","rank"))
tx=tx[tx$rank %in% c("species","subspecies"),] ## restrict to species and subspecies
```

We can make a taxonomic tree plot using the `phytools` package:

```
## as.phylo requires the taxonomic columns to be factors
temp=colwise(factor, c("genus","scientificName"))(tx)
## create phylo object of Scientific.Name nested within Genus
ax=as.phylo(~genus/scientificName,data=temp)
tr=plotTree(ax,type="fan",fsize=0.7) ## plot it
```



We can also plot the tree with images of the different penguin species. We’ll first extract a species profile for each species identifier (guid) in our results:

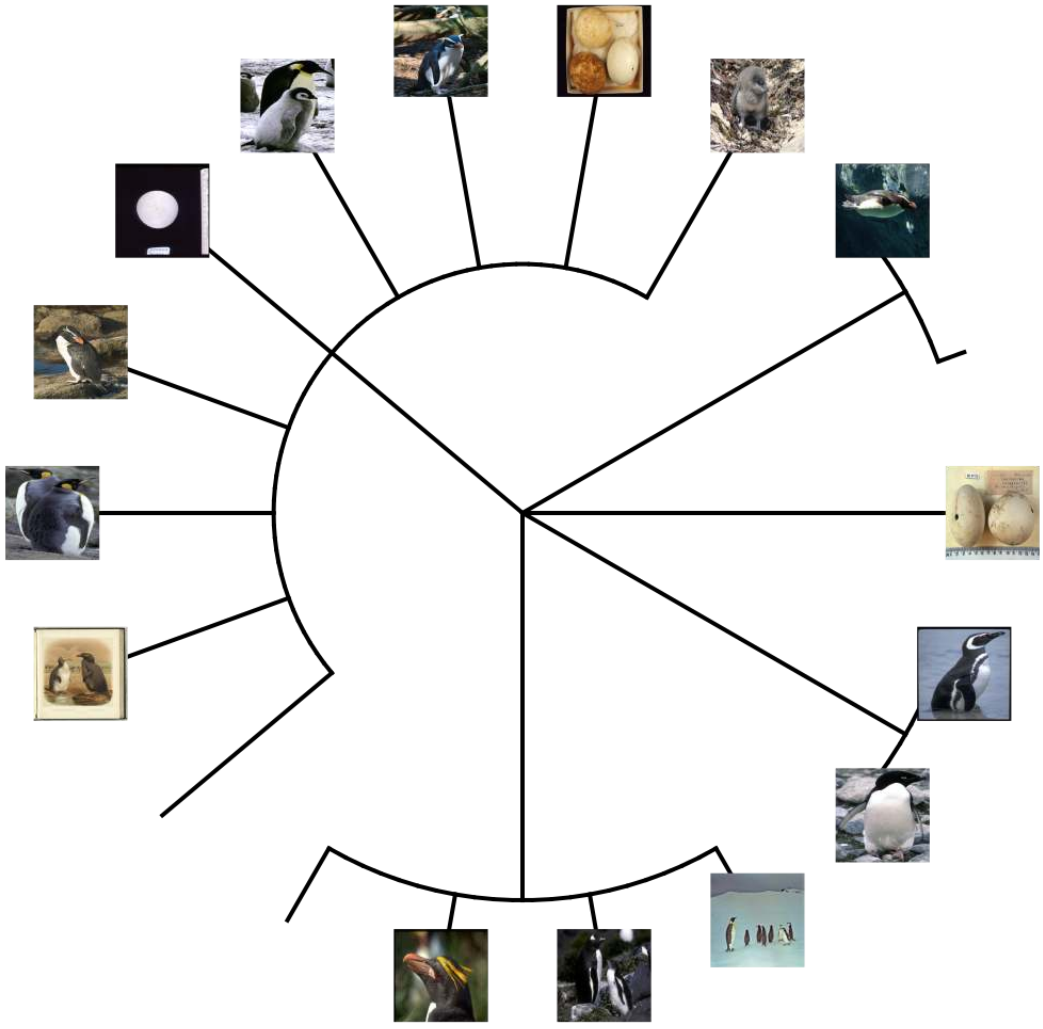
```
s=lapply(tx$guid,function(z){ species_info(guid=z) })
```

And for each of those species profiles, find the first jpeg image, download it and store it in our data cache:

```
imfiles=sapply(s,function(z){
  ifelse(any(grepl("(\\.jpg)",z$images$smallImageUrl,ignore.case=TRUE)),
  ALA4R:::cached_get(z$images$smallImageUrl[grepl("(\\.jpg)",z$images$smallImageUrl,
  ignore.case=TRUE)][1],type="binary_filename"),"")
})
```

And finally, plot the tree:

```
tr=plotTree(ax,type="fan",ftype="off") ## plot tree without labels
## add each image
library(jpeg)
for (k in which(nchar(imfiles)>0)) {
    tryCatch({
           im=readJPEG(imfiles[k]);
           rasterImage(im,tr$xx[k]-1/10,tr$yy[k]-1/10,tr$xx[k]+1/10,tr$yy[k]+1/10) },
           error=function(e){invisible(1)})
}
```



Example 2: Area report: what listed species exist in a given area?

First download an example shapefile of South Australian conservation reserve boundaries: see http://data.sa.gov.au/dataset/conservation-reserve-boundaries (http://data.sa.gov.au/dataset/conservation-reserve-boundaries). We use the ALA4R's caching mechanism here, but you could equally download this file directly.

```
library(maptools)
shape_filename=ALA4R:::cached_get("http://www.naturemaps.sa.gov.au/files/CONSERVATION_Npwsa_Reserves_shp.zip", type="binary_filename")
unzip(shape_filename,exdir=ala_config()$cache_directory) ## unzip this file
shape=readShapePoly(file.path(ala_config()$cache_directory, "CONSERVATION_NpwsaReserves.shp"))
## extract just the Morialta Conservation Park polygon
shape=shape[shape$RESNAME=="Morialta",]
```

We could create the WKT string using the `rgeos` library:

```
library(rgeos)
wkt=writeWKT(shape)
```

Unfortunately, in this instance this gives a WKT string that is too long and won't be accepted by the ALA web service. Instead, let's construct the WKT string directly, which gives us a little more control over its format:

```
lonlat=shape@polygons[[1]]@Polygons[[1]]@coords ## extract the polygon coordinates
## extract the convex hull of the polygon to reduce the length of the WKT string
temp=chull(lonlat)
lonlat=lonlat[c(temp,temp[1]),]
## create WKT string
wkt=paste("POLYGON((",paste(apply(lonlat,1,function(z) paste(z,collapse=" ")),collapse=",
"),"))",sep="")
```

Now extract the species list in this polygon:

```
x=specieslist(wkt=wkt,fq="state_conservation:*")
(head(arrange(x,desc(occurrenceCount)),20))
```

```
##                                                                taxonConceptLsid
## 1  urn:lsid:biodiversity.org.au:afd.taxon:e7873288-a90c-4f20-8be1-e8ec69a074a5
## 2                                urn:lsid:biodiversity.org.au:apni.taxon:379015
## 3                                urn:lsid:biodiversity.org.au:apni.taxon:664437
## 4                                urn:lsid:biodiversity.org.au:apni.taxon:305417
## 5                                urn:lsid:biodiversity.org.au:apni.taxon:287504
## 6                                urn:lsid:biodiversity.org.au:apni.taxon:667365
## 7                                urn:lsid:biodiversity.org.au:apni.taxon:546061
## 8                                urn:lsid:biodiversity.org.au:apni.taxon:322448
## 9                                urn:lsid:biodiversity.org.au:apni.taxon:693424
## 10 urn:lsid:biodiversity.org.au:afd.taxon:6246cb3a-04c4-4ae5-995a-5ecef8250d6c
## 11                               urn:lsid:biodiversity.org.au:apni.taxon:368705
## 12                               urn:lsid:biodiversity.org.au:apni.taxon:702500
## 13                               urn:lsid:biodiversity.org.au:apni.taxon:126854
```

```
## 14 urn:lsid:biodiversity.org.au:afd.taxon:00b1b9a2-70c9-45be-8019-9c7fd755afc8
## 15                            urn:lsid:biodiversity.org.au:apni.taxon:717120
## 16 urn:lsid:biodiversity.org.au:afd.taxon:c3e68140-6469-4e00-a33e-de700d1f16f3
## 17 urn:lsid:biodiversity.org.au:afd.taxon:ff789c7f-19b5-4205-9ef3-05ab294ec195
## 18                            urn:lsid:biodiversity.org.au:apni.taxon:717217
## 19 urn:lsid:biodiversity.org.au:afd.taxon:4273bd9a-f874-4a33-b1f0-3633abfdc5c8
## 20                            urn:lsid:biodiversity.org.au:apni.taxon:285221
##    speciesName scientificNameAuthorship rank kingdom phylum class order
## 1           NA                       NA   NA      NA     NA    NA    NA
## 2           NA                       NA   NA      NA     NA    NA    NA
## 3           NA                       NA   NA      NA     NA    NA    NA
## 4           NA                       NA   NA      NA     NA    NA    NA
## 5           NA                       NA   NA      NA     NA    NA    NA
## 6           NA                       NA   NA      NA     NA    NA    NA
## 7           NA                       NA   NA      NA     NA    NA    NA
## 8           NA                       NA   NA      NA     NA    NA    NA
## 9           NA                       NA   NA      NA     NA    NA    NA
## 10          NA                       NA   NA      NA     NA    NA    NA
## 11          NA                       NA   NA      NA     NA    NA    NA
## 12          NA                       NA   NA      NA     NA    NA    NA
## 13          NA                       NA   NA      NA     NA    NA    NA
## 14          NA                       NA   NA      NA     NA    NA    NA
## 15          NA                       NA   NA      NA     NA    NA    NA
## 16          NA                       NA   NA      NA     NA    NA    NA
## 17          NA                       NA   NA      NA     NA    NA    NA
## 18          NA                       NA   NA      NA     NA    NA    NA
## 19          NA                       NA   NA      NA     NA    NA    NA
## 20          NA                       NA   NA      NA     NA    NA    NA
##    family genus commonName occurrenceCount
## 1      NA    NA         NA             289
## 2      NA    NA         NA             289
## 3      NA    NA         NA             289
## 4      NA    NA         NA              50
## 5      NA    NA         NA              50
## 6      NA    NA         NA              50
## 7      NA    NA         NA              35
## 8      NA    NA         NA              35
## 9      NA    NA         NA              35
## 10     NA    NA         NA              27
## 11     NA    NA         NA              27
## 12     NA    NA         NA              27
## 13     NA    NA         NA              25
## 14     NA    NA         NA              25
## 15     NA    NA         NA              25
## 16     NA    NA         NA              23
## 17     NA    NA         NA              23
## 18     NA    NA         NA              23
## 19     NA    NA         NA              20
## 20     NA    NA         NA              20
```

Example 3: Quality assertions

Download occurrence data for the golden bowerbird:

```
x=occurrences(taxon="Amblyornis newtonianus", download_reason_id=10)
summary(x)
```

```
## number of names: 7
## number of taxonomically corrected names: 1
## number of observation records: 882
## number of assertions listed: 19  -- ones with flagged issues are listed below
##  invalidCollectionDate: 119 records
##  incompleteCollectionDate: 159 records
##  firstOfCentury: 4 records
##  detectedOutlier: 14 records -- considered fatal
##  uncertaintyRangeMismatch: 32 records
##  firstOfYear: 26 records
##  altitudeNonNumeric: 37 records
##  altitudeInFeet: 2 records
##  geodeticDatumAssumedWgs84: 620 records
##  speciesOutsideExpertRange: 14 records -- considered fatal
##  decimalLatLongConverted: 5 records
##  coordinatePrecisionMismatch: 18 records
##  countryInferredByCoordinates: 456 records
##  invalidImageUrl: 1 records
##  unrecognizedGeodeticDatum: 105 records
##  inferredDuplicateRecord: 134 records
##  stateCoordinateMismatch: 1 records
##  habitatMismatch: 14 records -- considered fatal
##  firstOfMonth: 65 records
```

You can see that some of the points have assertions that are considered “fatal” (i.e. the occurrence record in question is unlikely to be suitable for subsequent analysis). We can use the `occurrences_plot` function to create a PDF file with a plot of this data, showing the points with fatal assertions (this will create an “Rplots.pdf” file in your working directory; not run here):

```
occurrences_plot(x,qa="fatal")
```

There are many other ways of producing spatial plots in R. The `leafletR` package provides a simple method of producing browser-based maps iwth panning, zooming, and background layers:

```
library(leafletR)
## drop any records with missing lat/lon values: leaflet does not like them
x$data=x$data[!is.na(x$data$longitude) & !is.na(x$data$latitude),]
xa=check_assertions(x)
## columns of x corresponding to a fatal assertion
x_afcols=names(x$data) %in% xa$occurColnames[xa$fatal]
## rows of x that have a fatal assertion
x_afrows=apply(x$data[,x_afcols],1,any)
## which fatal assertions are present in this data?
these_assertions=names(x$data)[x_afcols]
## start with the "clean" data (data rows without fatal assertions)
datlist=list(toGeoJSON(data=x$data[!x_afrows,c("latitude","longitude")],name="Am0",dest=t
empdir()))
## now for each assertion, create a geojson formatted-file of the associated data
for (k in 1:length(these_assertions)) {
    idx=x$data[,which(x_afcols)[k]]
    datlist[k+1]=toGeoJSON(data=x$data[idx,c("latitude","longitude")],name=paste("Am",k,s
ep=""),dest=tempdir())
}
## create styles
sty0=styleSingle(col="white",fill="black",fill.alpha=1)
sty1=styleSingle(col="red",fill="red",fill.alpha=1)
sty2=styleSingle(col="yellow",fill="yellow",fill.alpha=1)
sty3=styleSingle(col="blue",fill="blue",fill.alpha=1)
## create the leaflet map
alamap=leaflet(data=datlist,title="Amblyornis newtonianus",base.map="mqsat",
  popup="mag",style=list(sty0,sty1,sty2,sty3),dest=tempdir())
```

And now you can open the map in your browser with:

```
browseURL(alamap)
```

Note: this would more elegantly be mapped as a single data set with categorical styling (marker colours by assertion) — but for unknown reasons this didn't seem to work properly.

Example 4: Community composition and turnover

Some extra packages needed here:

```
library(vegan)
library(mgcv)
library(geosphere)
```

Define our area of interest as a transect running westwards from the Sydney region, and download the occurrences of legumes (Fabaceae; a large family of flowering plants) in this area:

```
wkt="POLYGON((152.5 -35,152.5 -32,140 -32,140 -35,152.5 -35))"
x=occurrences(taxon="family:Fabaceae",wkt=wkt,qa="none",download_reason_id=10)
x=x$data ## just take the data component
```

Bin the locations into 0.5-degree grid cells:

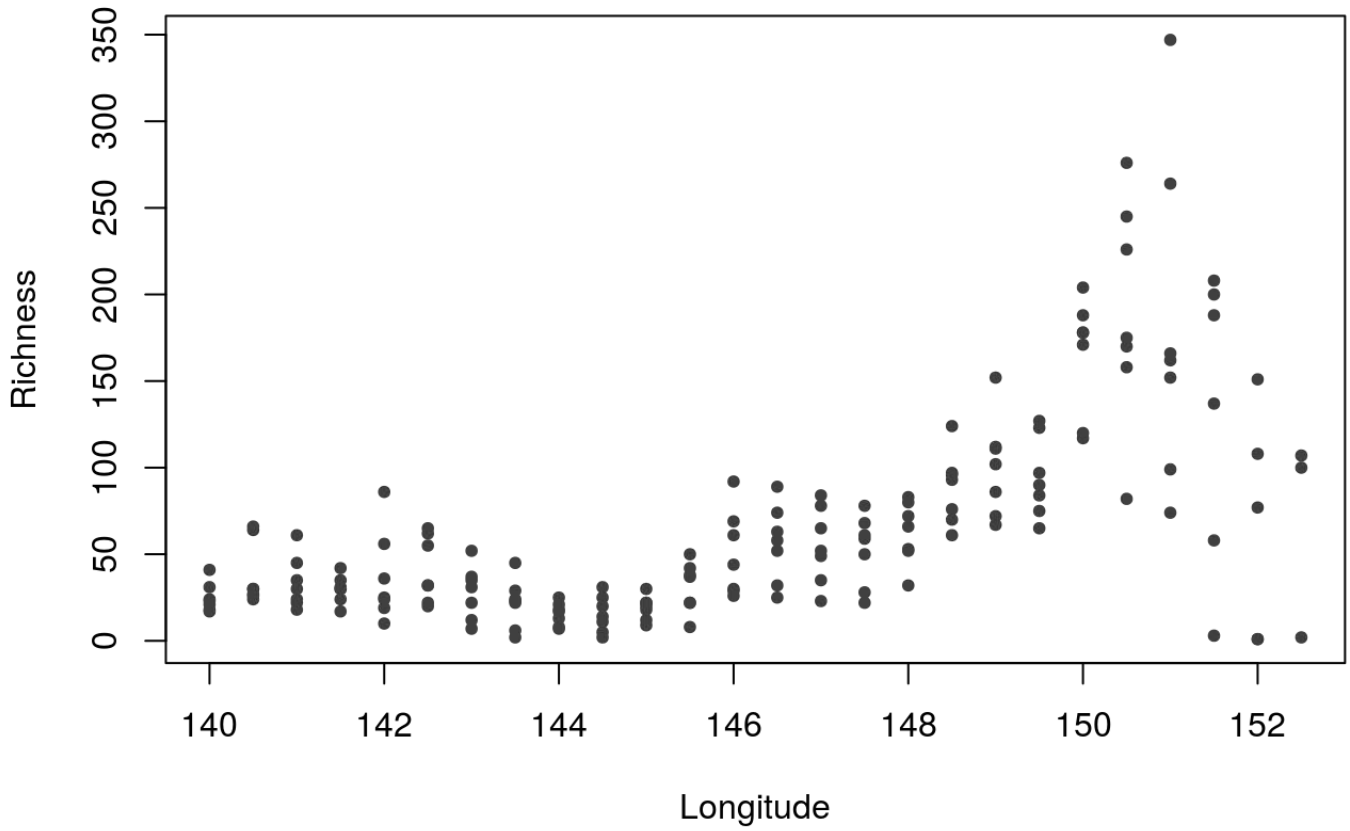
```
x$longitude=round(x$longitude*2)/2
x$latitude=round(x$latitude*2)/2
```

Create a sites-by-species data frame. This could also be done with e.g. the reshape library or the table() function, or indeed directly from ALA4R's `species_by_site` function. Note: this process inherently makes some strong assumptions about *absences* in the data.

```
## discard genus- and higher-level records
xsub=x$rank %in% c("species","subspecies","variety","form","cultivar")
unames=unique(x[xsub,]$scientificName) ## unique names
ull=unique(x[xsub,c("longitude","latitude")])
xgridded=matrix(NA,nrow=nrow(ull),ncol=length(unames))
for (uli in 1:nrow(ull)) {
    lidx=xsub & x$longitude==ull[uli,]$longitude & x$latitude==ull[uli,]$latitude
    xgridded[uli,]=as.numeric(unames %in% x[lidx,]$scientificName)
}
xgridded=as.data.frame(xgridded)
names(xgridded)=unames
xgridded=cbind(ull,xgridded)
```

Now we can start to examine the patterns in the data. Let's plot richness as a function of longitude:

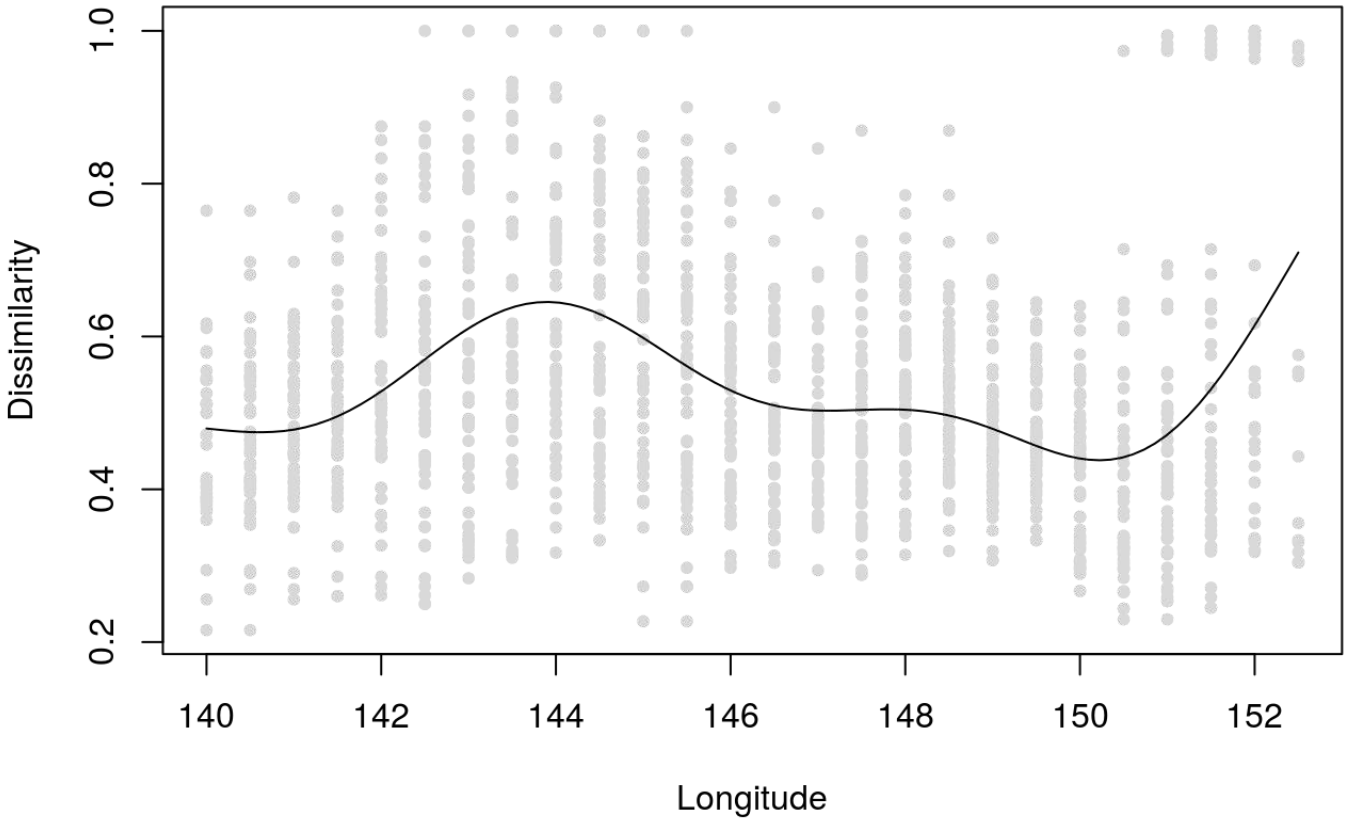
```
plot(xgridded$longitude,apply(xgridded[,-c(1:2)],1,sum),ylab="Richness",
  xlab="Longitude",pch=20,col="grey25")
```



The number of species is highest at the eastern end of the transect (the Sydney/Blue Mountains area). This probably reflects both higher species richness as well as greater sampling effort in this area compared to the western end of the transect.

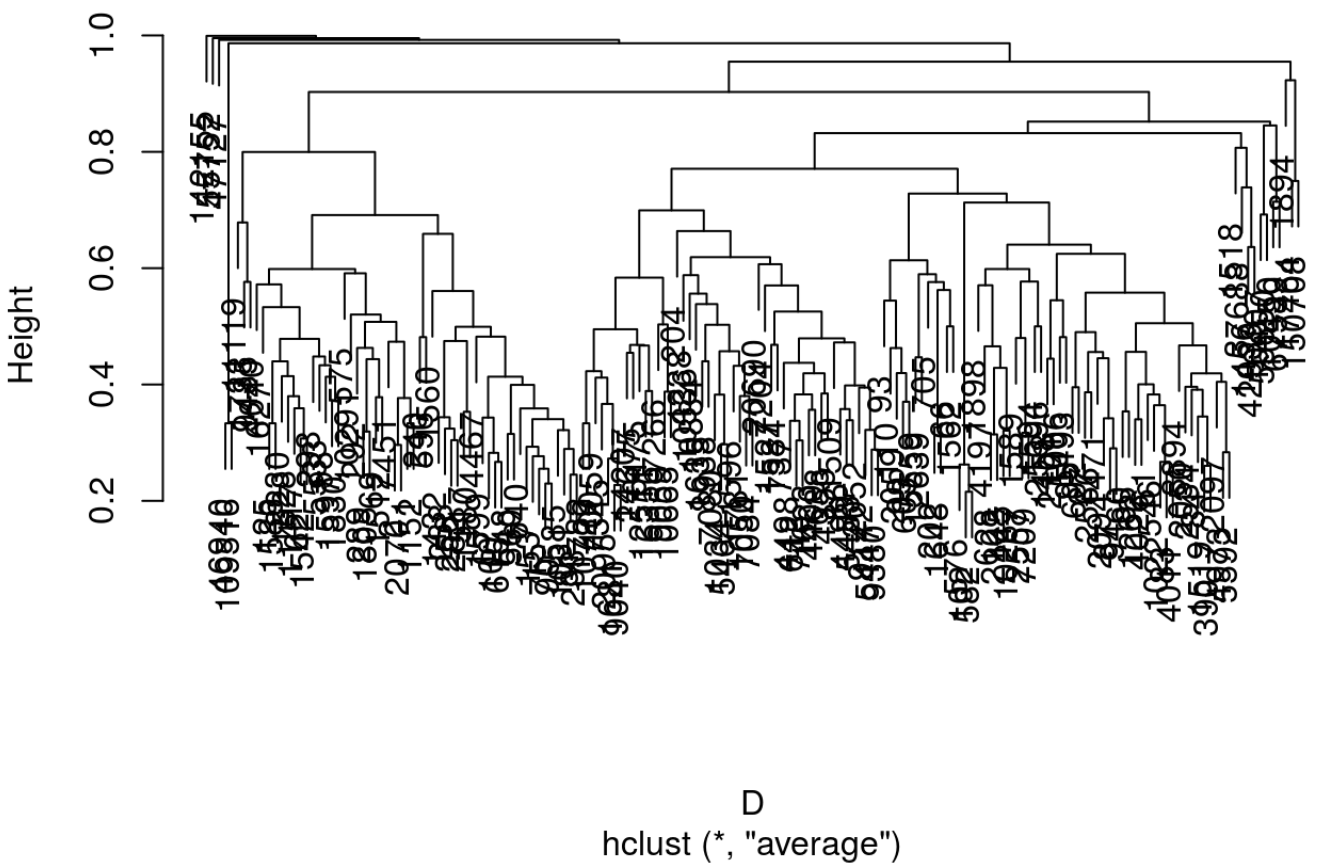
How does the community composition change along the transect? Calculate the dissimilarity between nearby grid cells as a function of along-transect position:

```
D=vegdist(xgridded[,-c(1:2)],'bray') ## Bray-Curtis dissimilarity
Dm=as.matrix(D) ## convert to a matrix object
## calculate geographic distance from longitude and latitude
Dll=apply(xgridded[,1:2],1,function(z){distVincentySphere(z,xgridded[,1:2])})
closeidx=Dll>0 & Dll<100e3 ## find grid cells within 100km of each other
## create a matrix of longitudes that matches the size of the pairwise-D matrices
temp=matrix(xgridded$longitude,nrow=nrow(xgridded),ncol=nrow(xgridded))
## plot dissimilarity as a function of transect position
plot(temp[closeidx],Dm[closeidx],xlab="Longitude",ylab="Dissimilarity",pch=20,col="grey85
")
## add smooth fit via gam()
fit=gam(d~s(tp,k=7),data=data.frame(tp=temp[closeidx],d=Dm[closeidx]))
tpp=seq(from=min(xgridded$longitude),to=max(xgridded$longitude),length.out=100)
fitp=predict(fit,newdata=data.frame(tp=tpp))
lines(tpp,fitp,col=1)
```

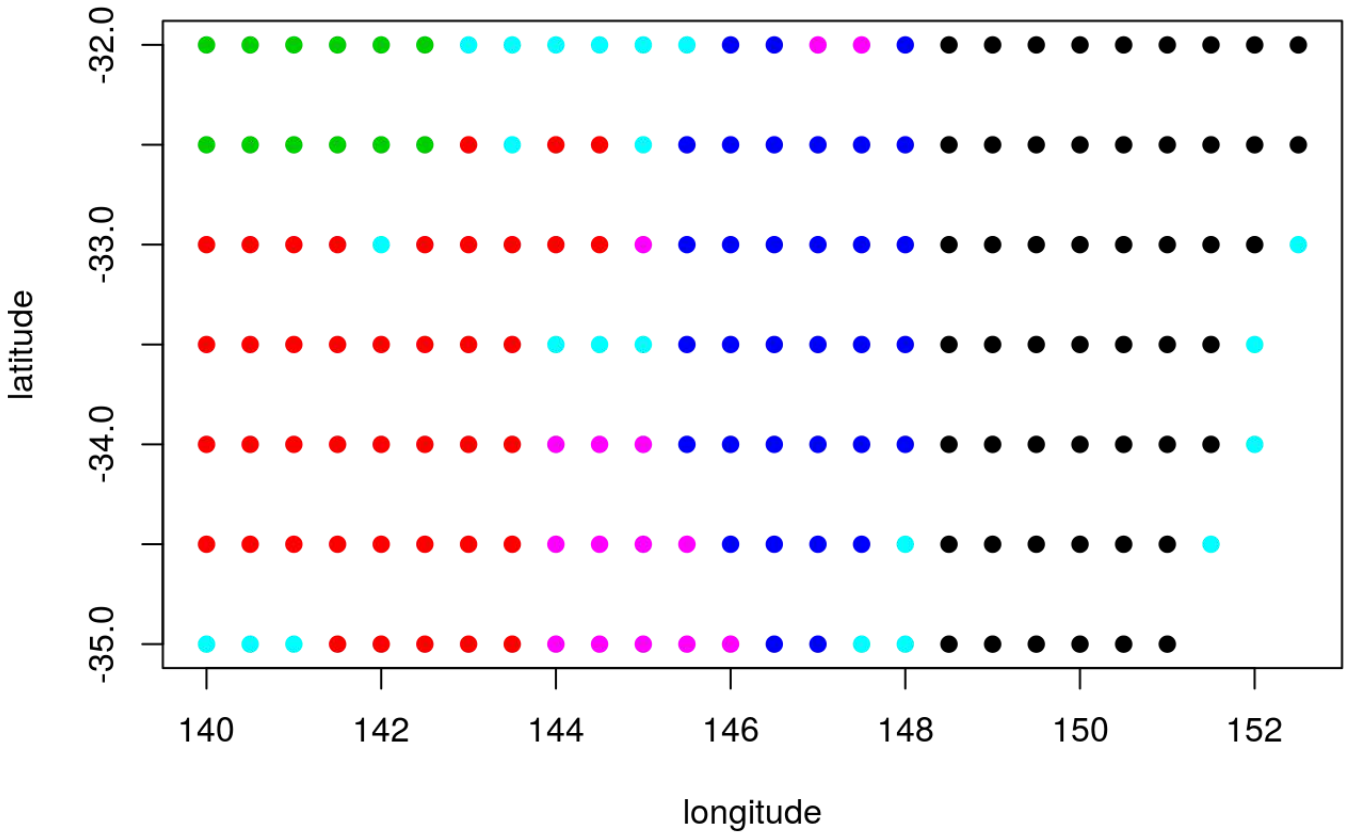


Clustering:

```
cl=hclust(D,method="ave") ## UPGMA clustering
plot(cl) ## plot dendrogram
```




```
grp=cutree(cl,20) ## extract group labels at the 20-group level
## coalesce small (outlier) groups into a single catch-all group
sing=which(table(grp)<5)
grp[grp %in% sing]=21 ## put these in a new combined group
grp=sapply(grp,function(z)which(unique(grp)==z)) ## renumber groups
## plot
with(xgridded,plot(longitude,latitude,pch=21,col=grp,bg=grp))
```




```
## or slightly nicer map plot
library(maps)
library(mapdata)
map("worldHires","Australia", xlim=c(105,155), ylim=c(-45,-10), col="gray90", fill=TRUE)
thiscol=c("#1f77b4","#ff7f0e","#2ca02c","#d62728","#9467bd","#8c564b","#e377c2","#7f7f7f"
,"#bcbd22","#17becf") ## colours for clusters
with(xgridded,points(longitude,latitude,pch=21,col=thiscol[grp],bg=thiscol[grp],cex=0.75)
)
```