Analysis of camera trap data

MSc course Global Ecology and Biodiversity

29 November, 2023

```r
#List of packages
requiredPackages=c("sf","maptiles","terra","iNEXT","corrplot","e1071","MASS","car")

#This code will install all required packages
newPackages = requiredPackages[!(requiredPackages %in% installed.packages()[,"Package"])]
#if some are not installed, install them
if(length(newPackages)) install.packages(newPackages)
```

Content

This practical deals with the analysis of data derived from camera traps. We'll use cameras deployed in the TEAM initiative: Tropical Ecology Assessment and Monitoring Network: more info here. TEAM monitors tropical ecosystems globally, in Africa, Asia and Latin America.

1 Camera trap data exploration

1.1 Reading in the data

The data we will be using today stems from a network of 60 cameras in and around the Parque Nacional Braulio Carrillo in Costa Rica. The Folder containing the data can be found on Canvas. Once you've downloaded the zip, move the data folders to your working directory.

You'll see the 'cameratrap_data' folder contains two .csv files: the images.csv file contains all observations while the deployments.csv contains details on every deployment: i.e. a camera recording in a specific location (or with specific settings), for a period of time.

```r
#load deployments.csv from the cameratrap_data folder
deploymentpath <- file.path("cameratrap_data","deployments.csv")
deployments <- read.csv(deploymentpath)
```

Use similar code to create an `observations` dataframe from `images.csv`

We will start by using the `head` function to explore the `deployments` dataframe, where each row represents a single deployment.

The first few variables are of particular interest.`project_id` denotes which project this deployment was attached to. All these deployments are from the same project. `deployment_id` is the unique identifier of a deployment. placename is the unique identifier of a location. Each location has a latitude and longitude. start_date and end_date record when a deployment.

Question: How many unique locations are there, and how many unique deployments?

We should convert the start and end dates to R date objects to use later. You can use the as.Date function to do this.

We can use these to add a variable for the length of time each deployment lasted. Create a new column in deployments called time_active to hold this.

If you've done this correctly, this column will be a difftime object. If you look at it, it'll state that the time difference is in days. Use the as.numeric function to convert it into a normal numeric column. This will make things easier later.

Lets also look at the variables in the observations dataframe. Here each row represents an observation of a single species from an image. A single image can have multiple observations of different species.

The first 6 columns contain information about the image. This dataframe also has a project_id variable showing that all these observations came from the same project. The deployment_id variable indicates which deployment this observation came from. This will be important later to let us link observations to environmental data. Following this is image_id, the individual ID of the image as well as the original filename of the image and its current location on the internet. After these variables come data on the individual observations, which we'll investigate more shortly.

Question: How many unique images are there, and how many observations?

1.2 Plotting the locations

It would be useful to plot the unique locations of deployments on a map. There are several packages in R that allow us to do this. In order to use these, we need to create a dataframe with only a single row for each placename and then convert this to a simple features objects with coordinates. We can later use this dataframe to ask questions about specific locations.

```r
#load the sf library to handle spatial data
library(sf)

# Create a boolean vector, which is TRUE if a value of deployments$placename
# has already appeared once.

dup_locations <- duplicated(deployments$placename)

# Subset the deployment using the dup_locations boolean.
# Using the ! inverts the boolean
# Inverted, TRUE is the first appearance of a deployments$placename
# Subsetting the rows by this results in a dataframe with one row per placename.
# We also subset columns by column name,
# We keep only the placename, lat and long variables.
locations <- deployments[!dup_locations,c("placename","longitude", "latitude")]

#Convert our dataframe to a "simple features" object.
#Specify the object to be converted, the columns
locations_sf <- st_as_sf(locations,
                         coords = c("longitude", "latitude"),
                         crs = 4326)
```

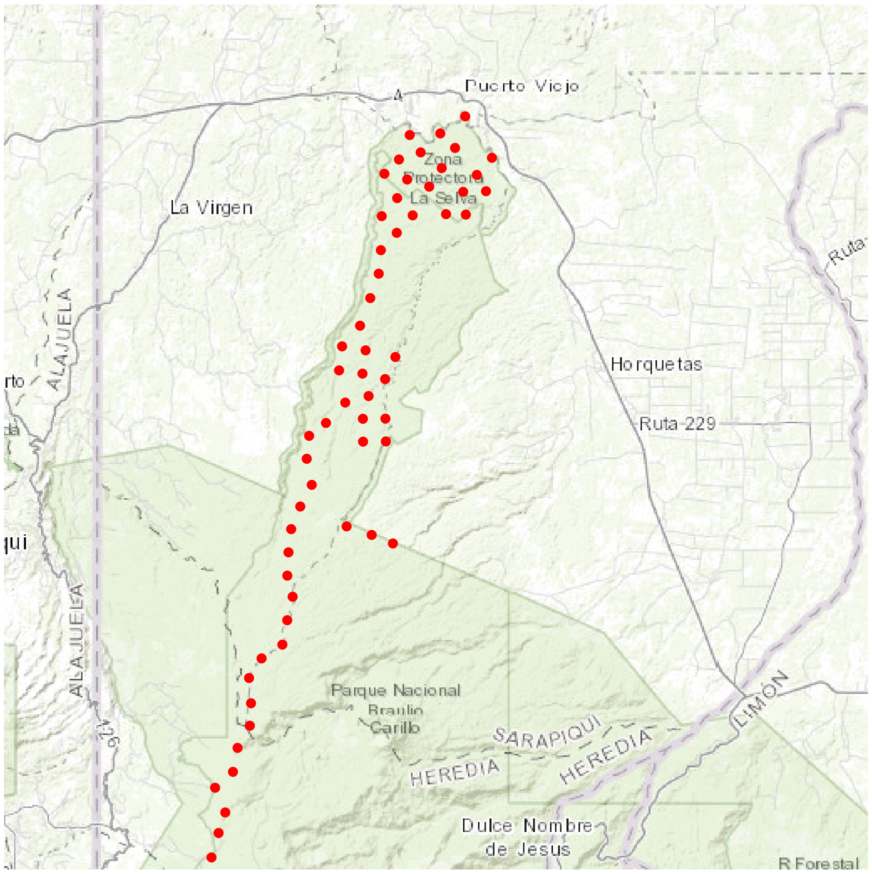
As you can see, R now reports that this is a simple feature collection, with one row per `placename`. We can now plot these locations on a map. Just typing `plot(locations_sf)` will plot the points, though this is not especially useful as R will try to show the `placename` variable. We can make a nicer map with a few additional commands

Load the `terra` and `maptiles` libraries to help make this map. Once you have loaded these you can use the `get_tiles` command from maptiles to get create a basemap object called `locations_basemap` This takes a spatial object and returns a map tile. The default basemap is open streetmap. For this map, we reccomend setting your `provider` to `"Esri.WorldTopoMap"`

You can see a list of available basemap providers by typing `?get_tiles`

Once you have your basemap, it is very simple to create a map.

```r
#Start a map plot using your basemap
plot(locations_basemap)
# add the locations as points, set the colour to be red
points(locations_sf,col="red")
```



Question: Based on this map, which environmental variable do you think these camera placements are trying to capture?

A common metric to report in camera trapping papers is a summary of the distances between camera trap placements. Use the `st_distance` function with your `locations_sf` to calculate a distance matrix. The diagonal of this matrix will be 0, as this is the distance between each camera and itself. You can use the `diag` function to set it to be NA so as not to bias your answers.

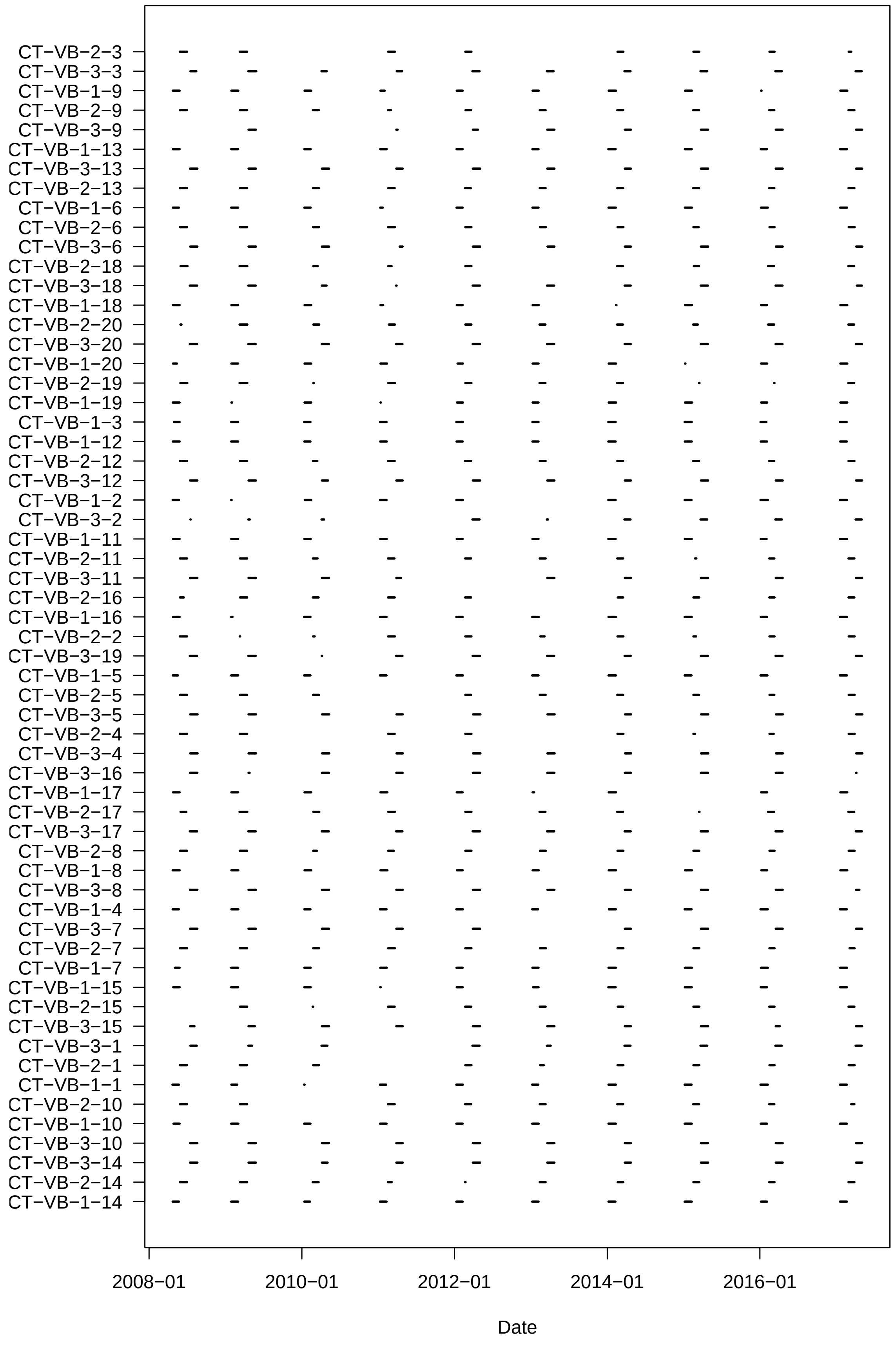
```r
# We can then show a summary of camera distances
# We flatten the matrix with as.vector, otherwise we will get a value per column
summary(as.vector(distmat))
```

Question: What is the minimum, mean and maximum distance between cameras?

Using these locations we can also look at when a camera was active at each location.

This code is optional, you don't need to run it yourself.

```r
#set the margins (bottom, left, top, right) so we can see the labels
par(mar=c(4,6,1,1))

#set up an empty plot
plot(0,#this will be ignored as we set type="n"
     type="n",#empty plot
     #set the full date range as our x axis
     xlim=c(min(deployments$start_date),max(deployments$end_date)),
     #set the number of locations as our x axis
     ylim=c(1,nrow(locations)),
     #the default labels on the axis will be wrong, so we suppress them
     # we do this by setting their type to "n"
     yaxt="n",
     xaxt="n",
     #set axis labels
     xlab="Date",
     ylab=""
     )
#set up a date axis on the x-axis with a year-month format
axis.Date(1,format="%Y-%m")
#add an axis of location labels
axis(2,at=c(1:nrow(locations)),labels=locations$placename,las=1)

#for every row of deployments
for(i in 1:nrow(deployments)){
  #select a deployment by row
  curr_deployment = deployments[i,]
  #get the numerical index of that deployments placename in our location df
  location_ind = which(locations$placename==curr_deployment$placename)
  #add a line from this deployments start to end date, at the loction index
  lines(x=c(curr_deployment$start_date,curr_deployment$end_date),
        y=c(location_ind,location_ind),
        lwd=2)
}
```


CT−VB−2−3
CT−VB−3−3
CT−VB−1−9
CT−VB−2−9
CT−VB−3−9
CT−VB−1−13
CT−VB−3−13
CT−VB−2−13
CT−VB−1−6
CT−VB−2−6
CT−VB−3−6
CT−VB−2−18
CT−VB−3−18
CT−VB−1−18
CT−VB−2−20
CT−VB−3−20
CT−VB−1−20
CT−VB−2−19
CT−VB−1−19
CT−VB−1−3
CT−VB−1−12
CT−VB−2−12
CT−VB−3−12
CT−VB−1−2
CT−VB−3−2
CT−VB−1−11
CT−VB−2−11
CT−VB−3−11
CT−VB−2−16
CT−VB−1−16
CT−VB−2−2
CT−VB−3−19
CT−VB−1−5
CT−VB−2−5
CT−VB−3−5
CT−VB−2−4
CT−VB−3−4
CT−VB−3−16
CT−VB−1−17
CT−VB−2−17
CT−VB−3−17
CT−VB−2−8
CT−VB−1−8
CT−VB−3−8
CT−VB−1−4
CT−VB−3−7
CT−VB−2−7
CT−VB−1−7
CT−VB−1−15
CT−VB−2−15
CT−VB−3−15
CT−VB−3−1
CT−VB−2−1
CT−VB−1−1
CT−VB−2−10
CT−VB−1−10
CT−VB−3−10
CT−VB−3−14
CT−VB−2−14
CT−VB−1−14
2008−01
2010−01
2012−01
2014−01
2016−01
Date

Question: What does this figure show? What could we do to use this to check camera coverage within a year?

We should also convert the timestamp of observations to an R date time object. For convenience we can also add a date variable.

```
#convert to POSIX continuous time, stating timezone and time format
observations$timestamp <- as.POSIXct(observations$timestamp, tz="UTC", format = "%Y-%m-%d %H:%M:%S")
#
observations$date <- as.Date(observations$timestamp)
```

Finally, we should link each observation to a location, via its deployment. This will later allow us to easily relate the environmental properties of that location

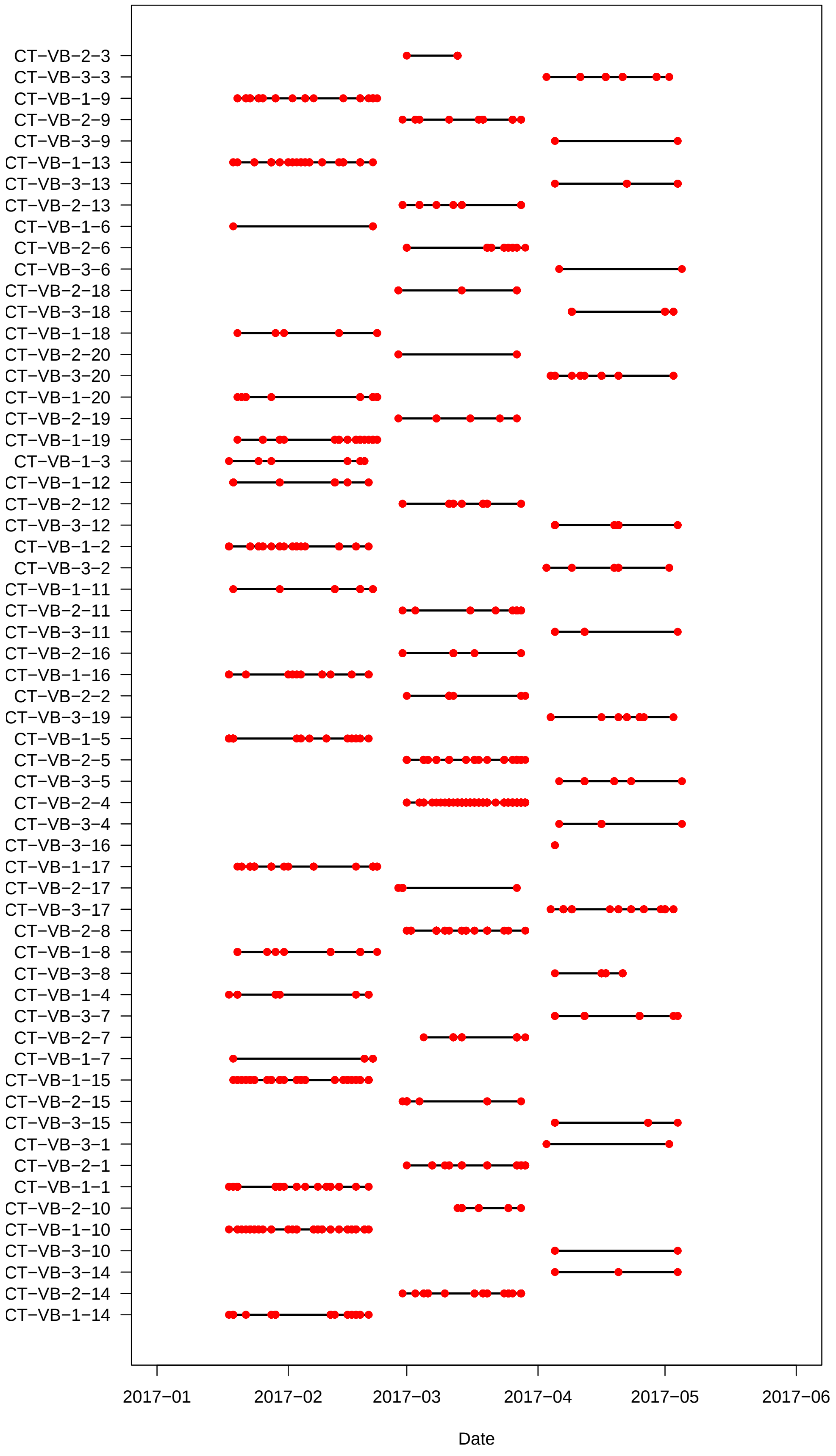
```
# We need to link the observation to the deployments, via the deployment_id
# First we get the numerical index of each observations deployment
# from the deployment dataframe
# To do this we use the match command to get indexes of deployments$deployment_id
# which match the observations$deployment_id
obs_deploy_index <- match(observations$deployment_id,deployments$deployment_id)

# Using this, we can get each observations placename.
# We can attach this to our observation dataframe
observations$placename <- deployments$placename[obs_deploy_index]
```

Doing this can also allow us to plot observations on our plot of deployment times. In order to better see the results, we focus on a single field season.

This code is optional, you don't need to run it yourself.

```
#This code is mostly the same to when we first plotted the deployments
#Only the x axis range is tweaked to restrict it to a single field season

#set the margins (bottom, left, top, right) so we can see the labels
par(mar=c(4,6,1,1))

#set up an empty plot
plot(0,#this will be ignored as we set type="n"
     type="n",#empty plot
     #set a limited date range as our axis
     xlim=as.Date(c("2017-01-01","2017-06-01")),
     #set the number of locations as our x axis
     ylim=c(1,nrow(locations)),
     #the default labels on the axis will be wrong, so we suppress them
     yaxt="n",
     xaxt="n",
     #set axis labels
     xlab="Date",
     ylab=""
     )
#set up a date axis on the x-axis with a year-month format
axis.Date(1,format="%Y-%m")
#add an axis of location labels
axis(2,at=c(1:nrow(locations)),labels=locations$placename,las=1)
```

```r
#for every row of deployments
for(i in 1:nrow(deployments)){
  #select a deployment by row
  curr_deployment = deployments[i,]
  #get the numerical index of that deployments placename in our location df
  location_ind = which(locations$placename==curr_deployment$placename)
  #add a line from this deployments start to end date, at the loction index
  lines(x=c(curr_deployment$start_date,curr_deployment$end_date),
        y=c(location_ind,location_ind),
        lwd=2)
}

#NEW CODE STARTS HERE

# as before, we convert placename to a numerical index so we can use it to plot
# the points by the correct location.

#This time we use the match command to get all the indexes at once
obs_place_ind = match(observations$placename,locations$placename)

#we can then add each observations as a point on the plot
points(observations$date,obs_place_ind,col="red",pch=16)
```



CT−VB−2−3
CT−VB−3−3
CT−VB−1−9
CT−VB−2−9
CT−VB−3−9
CT−VB−1−13
CT−VB−3−13
CT−VB−2−13
CT−VB−1−6
CT−VB−2−6
CT−VB−3−6
CT−VB−2−18
CT−VB−3−18
CT−VB−1−18
CT−VB−2−20
CT−VB−3−20
CT−VB−1−20
CT−VB−2−19
CT−VB−1−19
CT−VB−1−3
CT−VB−1−12
CT−VB−2−12
CT−VB−3−12
CT−VB−1−2
CT−VB−3−2
CT−VB−1−11
CT−VB−2−11
CT−VB−3−11
CT−VB−2−16
CT−VB−1−16
CT−VB−2−2
CT−VB−3−19
CT−VB−1−5
CT−VB−2−5
CT−VB−3−5
CT−VB−2−4
CT−VB−3−4
CT−VB−3−16
CT−VB−1−17
CT−VB−2−17
CT−VB−3−17
CT−VB−2−8
CT−VB−1−8
CT−VB−3−8
CT−VB−1−4
CT−VB−3−7
CT−VB−2−7
CT−VB−1−7
CT−VB−1−15
CT−VB−2−15
CT−VB−3−15
CT−VB−3−1
CT−VB−2−1
CT−VB−1−1
CT−VB−2−10
CT−VB−1−10
CT−VB−3−10
CT−VB−3−14
CT−VB−2−14
CT−VB−1−14
2017−01
2017−02
2017−03
2017−04
2017−05
2017−06
Date

This is an extremely useful plot when checking your cameras worked properly and your metadata is properly entered.

Question: What would a dot not on a line indicate?

1.3 Species data

Lets take a look at the species data. This is more or less everything beyond the 6th variable of observations.

These columns show the full taxonomy of the identified species from `class`, through to `species` along with who identified them. Note that `species` is not the full binomial species name, but rather just the "specific name".

Question: Why might this be a problem?

It will be useful to build a new column which combines `genus` and `species` into the full scientific name. Use the paste command to create a new column called `fullspecies`.

Question: How many unique fullspecies are there? How does this differ from the number derived from `species`? Which species should we remove?

For today's analysis we will remove unknowns unknowns, blanks and humans. For blanks you will also see that there is a full species called " ". This actually consists of a few different type of images, which are described in the `common_name` and `is_blank` column.

```
#subset dataset to get only the blank full species and the relevant columns
allblanks <- observations[observations$fullspecies==" ",c("is_blank","common_name")]
#restrict this to only each unique common name, using an inverted duplicated function
allblanks[!duplicated(allblanks$common_name),]
```

```
##     is_blank  common_name
## 22         1        Blank
## 93         0 Setup_Pickup
## 391        0      Misfire
```

You will see that there are three types of blank image. Firstly the truly `Blank` images, as indicated by having a `1` in the `is_blank` column. This may be caused by an animal triggering the camera, but not appearing the image. Secondly the `Setup_Pickup`. These are images of the researchers setting up and collecting the camera at the beginning and end of a deployment. Finally, there are misfires where the camera is triggers due to some internal error.

For more details see here

For today's exercise, all of these can be excluded. Create a boolean of which observations are `%in%` (hint) the following list: " ","Unknown Unknown" and "Homo sapiens".

```
#subset observations by an inverted (!) version of this boolean
observations <- observations[!badobs_bool,]
```

Question: If you make the plot of observations per location now, how has it changed?

1.4 Aggregating species observations by location

We can calculate the number of detections per deployment

```r
det_per_deployment <- aggregate(
  number_of_objects~deployment_id,#number of objects (in a picture) BY placename
  FUN=sum,#we will add these
  data=observations)

head(det_per_deployment)
```

```
##        deployment_id number_of_objects
## 1  2007.01-CT-VB-1-1               502
## 2 2007.01-CT-VB-1-10               833
## 3 2007.01-CT-VB-1-11               139
## 4 2007.01-CT-VB-1-12               601
## 5 2007.01-CT-VB-1-13                79
## 6 2007.01-CT-VB-1-14                13
```

Question: Is number of detections the same as number of individuals?

We can also tweak this to give us the number of images per deployment

```r
pic_per_location<-aggregate(
  number_of_objects~deployment_id,#number of objects (in a picture) BY deployment
  FUN=length,#instead of adding, we cound the number of rows
  data=observations)

head(det_per_deployment)
```

```
##        deployment_id number_of_objects
## 1  2007.01-CT-VB-1-1               502
## 2 2007.01-CT-VB-1-10               833
## 3 2007.01-CT-VB-1-11               139
## 4 2007.01-CT-VB-1-12               601
## 5 2007.01-CT-VB-1-13                79
## 6 2007.01-CT-VB-1-14                13
```

It is easy to tweak this code to look at these results by location. You can then break this down by placename and species by adding `fullspecies` to your formula.

Question: Which location has the most pictures? At which LOCATION were Oncilla detected most often?

2 Calculating species richness

Having explored our data, we want to examine how many species are in the survey area. Exploring patterns in species richness can also show us how sampling effort affects the numbers of species seen and whether sufficient sampling effort has been used to get an accurate indication.

2.1 Observed species richness.

The simplest way to quantify species richness is counting the number of species observed by a camera - 'observed richness'.

We'll do this initially at the deployment level, as it is easy to later sum them if we want per location richness.

```r
#for every deployment ID
for(deployment in deployments$deployment_id){
  #get observations from that deployment
  deployment_obs <- observations$fullspecies[observations$deployment_id==deployment]
  #get the unique values of these
  uni_obs <- unique(deployment_obs)
  #count them
  n_uni_obs <- length(uni_obs)
  #add them to the deployments dataframe, at the corect deployment
  deployments$n_species[deployments$deployment_id==deployment] <- n_uni_obs
}
```

Question: Which deployment has the greatest species richness?

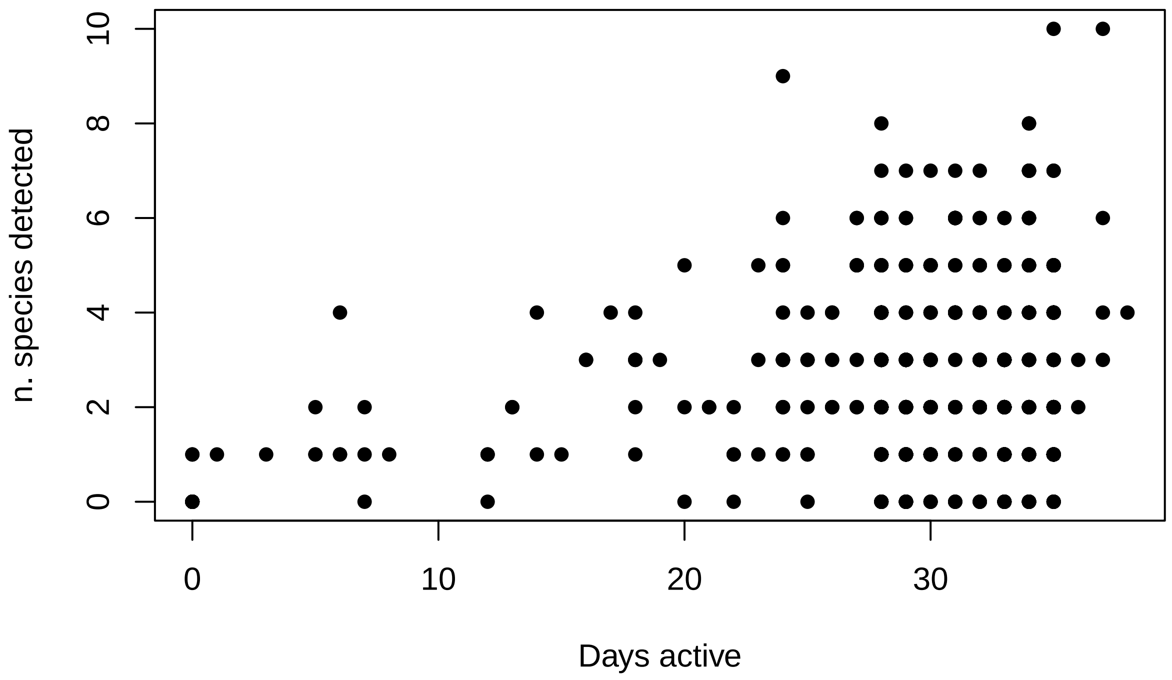
```r
maxspecies = max(deployments$n_species)
deploy_most_species = deployments$deployment_id[deployments$n_species==maxspecies]
```

Bonus question: Which *location* has the greatest species richness

If we wanted to compare between deployments, we theoretically do so. However ideally the survey effort (the length of time the cameras were deployed) would be equal between deployments. This is not often possible in camera trap studies where cameras break, run out of battery or are deployed for different lengths of time. These differences can lead to differences in the observed species richness

To illustrate this, we can plot species richness against the length of time a deployment lasted.

```r
#plot n_species BY time_active
plot(n_species~time_active,
     #data comes from the deployments dataframe
     data=deployments,
     #solid point symbol
     pch=16,
     #axis labels
     xlab="Days active",
     ylab="n. species detected")
```



As you can see, the number of species detected is definitely influenced by the length of time a camera is active. This means that observed richness will generally underestimate true richness. Consequently, We need ways of comparing species richness which accounts in some way for survey effort.

2.2 Estimated richness

One of the common ways to account for survey effort when estimating species richness from camera trap studies is to 'correct' observed richness using the incidence of rare species.

We can use the iNext package to estimate species richness. The methods used in this package are based off of Chao, Anne, et al. "Rarefaction and extrapolation with Hill numbers: a framework for sampling and estimation in species diversity studies." Ecological monographs 84.1 (2014): 45-67

We will also want to check if the sampling effort is sufficient capture the species present at a location. To do this we can compute a species accumulation curves. Species accumulation curves plot the increase in species richness as we add survey units (deployment duration in our case). If the curve plateaus (flattens), then that suggests we have sampled the majority of the species in the survey area.

2.2.1 Data formatting

In order to use iNEXT we need to reshape our data. There are a few different ways we can provide data, but the easiest is to create an "indidence matrix". This will be a SxN matrix where S is the number of species and N is the number of sampling events. To begin with, we will create such a matrix for a single deployment.

```
#this will be our deployment
deployment <- "2008.01-CT-VB-2-8"
```

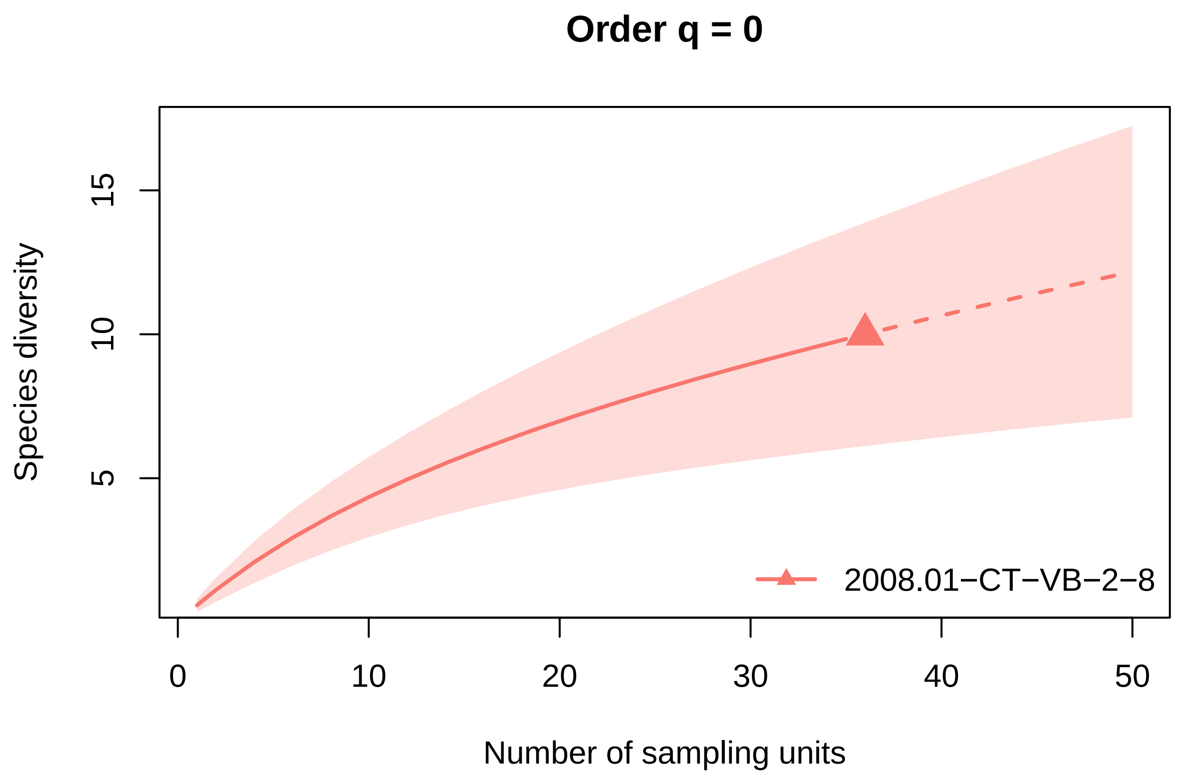
```r
deploy_start <- deployments$start_date[deployments$deployment_id==deployment]
deploy_end <- deployments$end_date[deployments$deployment_id==deployment]
#get all observations for that deployment
deploy_obs <- observations[observations$deployment_id==deployment,]
#get unique species for that deployment
deploy_spec <- unique(deploy_obs$fullspecies)
#get sequence of dates for that deployment from start, to end
deploy_days <- seq.Date(deploy_start,deploy_end,by="day")
#create a matrix of 0s, n. rows is number of species, n. cols is number of days
inc_mat <- matrix(0,length(deploy_spec),length(deploy_days))
#set row names to species
row.names(inc_mat) <- deploy_spec

#for every value of i from 1 to the number of days
for(i in 1:length(deploy_days)){
  #get the actual date
  currday <- deploy_days[i]
  #get the species on that day (there might be none)
  curr_spec <- deploy_obs$fullspecies[deploy_obs$date==currday]
  #get a boolean where TRUE indicates the species was observed that day
  present_bool <- deploy_spec%in%curr_spec
  #convert this to a number (1 or 0), assign it to column i in our matrix
  inc_mat[,i]=as.numeric(present_bool)
}

#to use this matrix in iNEXT, it needs to go into a list
#create an empty list
deployments_inc_mats = list()
# add the incidence matrix we just created
deployments_inc_mats[[1]] = inc_mat
#give this the name of our deployment
names(deployments_inc_mats)[[1]] = deployment
```

We put this matrix into a list because iNEXT is set up to compare multiple different incidence matrices. We will look at this later.

2.2.2 species accumulation curve

Now, we can calculate our estimated richness. You can ignore the warning

```r
library(iNEXT)
# run iNEXT. We tell it the data it is receiving are raw incidences.
# We set the endpoint to 50 days
iNET_out <- iNEXT(deployments_inc_mats, datatype="incidence_raw",endpoint=50)
#plot it!
plot(iNET_out)
```

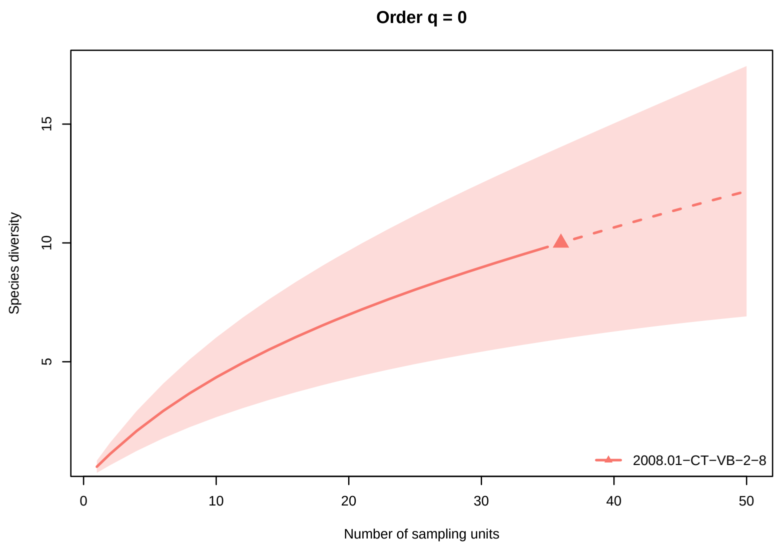


iNEXT comes with some handy functions to easily plot our species accumulation curve. In this plot, the number of sampling units (days) are on the x axis while the species diversity (n. species) is on the y axis. The Triangle represents the end of our provided data and the start of extrapolated data (predicted change in richness if the camera was left for longer).

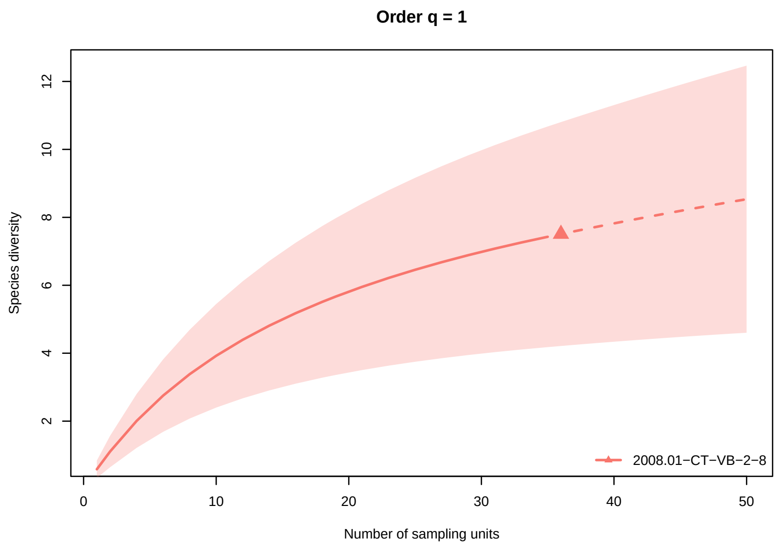
it looks like the number of species would be expected to increase if the camera was in the field longer! But this is one of the cameras that was in the field longest. This is likely because by default, iNEXT weights all species equally, whether they are rare or common. As mentioned above, we might want to control this. iNEXT can use different diversity indices by setting the `q` argument, where q=0 weights all species equally, q=1 is the traditional Shannon diversity indices and q=2 is the Simpson diversity index. These differ in how they calculate and weight rare species.

We can show all three of these in one plot.

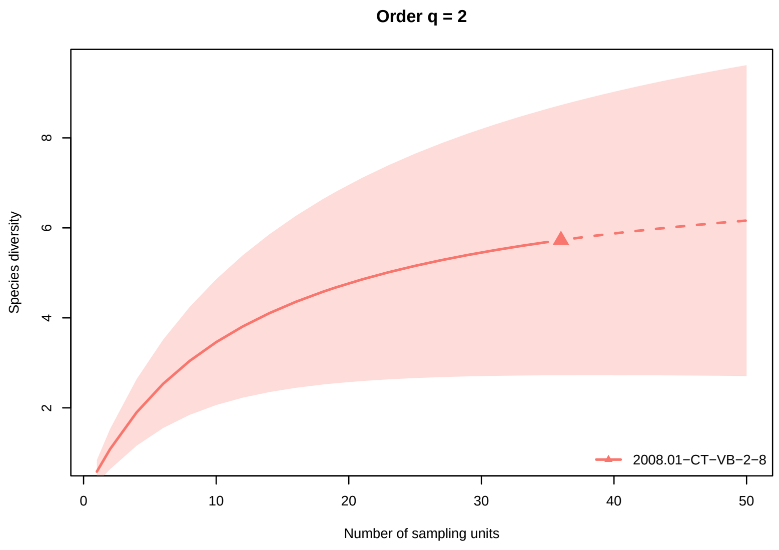
```r
library(iNEXT)
#set graphical parameters for 3 rows of plots, one columns
par(mfrow=c(3,1))
# run iNEXT. We tell it the data it is receiving are raw incidences.
iNET_out <- iNEXT(deployments_inc_mats,
                  q=c(0,1,2),
                  datatype="incidence_raw",
                  endpoint=50)
#plot it!
plot(iNET_out)
```



Order q = 0
Species diversity
5
10
15
0
10
20
30
40
50
Number of sampling units
2008.01−CT−VB−2−8



Order q = 1
Species diversity
2
4
6
8
10
12
0
10
20
30
40
50
Number of sampling units
2008.01−CT−VB−2−8



Order q = 2
Species diversity
2
4
6
8
0
10
20
30
40
50
Number of sampling units
2008.01−CT−VB−2−8

As you can see, using these diversity indices to weight rare species differently results in the curve starting to asymptote (the line stops rising). This may suggest that the length of time this camera was in the field was sufficient to capture most of the common species in the area.

2.2.3 comparing deployments

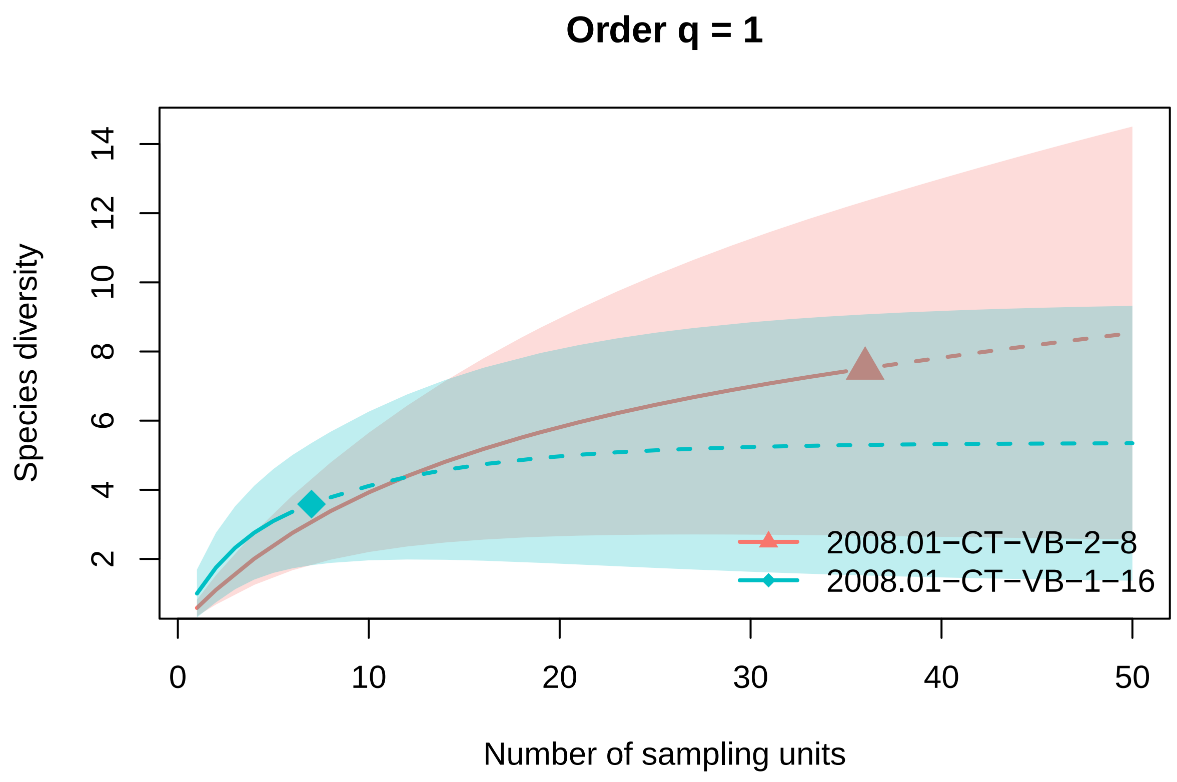
Lets generate another incidence matrix for a different deployment and add it to our list

```r
#this will be our deployment
deployment <- "2008.01-CT-VB-1-16"

#code that follows is the same as previous
deploy_start <- deployments$start_date[deployments$deployment_id==deployment]
deploy_end <- deployments$end_date[deployments$deployment_id==deployment]
#get all observations for that deployment
deploy_obs <- observations[observations$deployment_id==deployment,]
#get unique species for that deployment
deploy_spec <- unique(deploy_obs$fullspecies)
#get sequence of dates for that deployment from start, to end
deploy_days <- seq.Date(deploy_start,deploy_end,by="day")
#create a matrix of 0s, n. rows is number of species, n. cols is number of days
inc_mat <- matrix(0,length(deploy_spec),length(deploy_days))
#set row names to species
row.names(inc_mat) <- deploy_spec

#for every value of i from 1 to the number of days
for(i in 1:length(deploy_days)){
  #get the actual date
  currday <- deploy_days[i]
  #get the species on that day (there might be none)
  curr_spec <- deploy_obs$fullspecies[deploy_obs$date==currday]
  #get a boolean where TRUE indicates the species was observed that day
  present_bool <- deploy_spec%in%curr_spec
  #convert this to a number (1 or 0), assign it to column i in our matrix
  inc_mat[,i]=as.numeric(present_bool)
}

#CODE IS SLIGHTLY DIFFERENT HERE

# add the incidence matrix we just created
deployments_inc_mats[[2]] = inc_mat
#give this the name of our deployment
names(deployments_inc_mats)[[2]] = deployment
```

We can now create a new species accumulation curve with both deployments on.

```r
# run iNEXT. We tell it the data it is receiving are raw incidences.
# Using Shannon diversity
iNET_out <- iNEXT(deployments_inc_mats,
                  q=1,
                  datatype="incidence_raw",
                  endpoint=50)
```

```
#plot it!
plot(iNET_out)
```



The results can be explored using the following code

First, the data we provided

```
# Information about the data we provided
iNET_out$DataInfo
```

Here you can see the two deployments (Assemblages) we provided, followed by the number of sampling units `T` and the total number of observations `U` and the number of observed species `S.obs`.

```
# Estimates and extrapolation
iNET_out$iNextEst
```

This output includes two data frames: `size_based` and `coverage_based`. These give diversity estimates for the data we provided (Rarefaction to observed) and the extrapolated diversity estimates over different number of sampling units. You can for example, look at what the estimated diversity (`qD`) is at the timestep 36 in both deployments. In our first deployment, this was the observed diversity. In the second deployment, this is an extrapolated diversity. Note the pretty huge confidence intervals (`qd.LCL,  qD.UCL`).

```
#asymptotic estimates
iNET_out$AsyEst
```

This gives the asymptotic estimates and their related statistics. You will see that this shows the observed and estimated richness for our actual data, and the observed and estimated values of the two available diversity indexes. If we wanted, we could add these values to our deployment dataframe. We'd need to repeat the steps above with all our deployments, choose whether we use a diversity index and then extract the estimated richness/diversity for each deployment. You could also calculate these measures at a location level.