Seascape Ecology

Lab 05 - Analysing Animal Movement Data

Ryan Reisinger

2025-11-11

Introduction

This week we'll be trying our hand at some basic analyses of animal tracking data.

There are numerous methods for analysing animal movement data, and many R packages too. Joo et al. (2019) provide a recent overview: https://besjournals.onlinelibrary.wiley.com/doi/10.1111/1365-2656.13116#:~:text=https%3A//doi.org/10.1111/1365%2D2656.13116. You'll see in Table 1 of Joo et al. that for a given task several packages could be used. So, tools I've used in this tutorial are usually only one of the options. **I would encourage you to explore others.** For each step in our analysis, think about what what we're doing, rather than about the tools we're using. This will make it easier if you need or want to switch tools later.

We'll be using GPS tracking data for two species of giant petrel: the northern giant petrel (*Macronectes halli*) and the southern giant petrel (*Macronectes giganteus*). The data come from a paper where my colleagues and I looked at niche segregation in these sibling species - how do the two species (as well as males and females) avoid competing for the same resources?

All the scripts and data for the analyses in that paper are stored on a Github repository https://github.com/ryanreisinger/giantPetrels. We'll use some of the data from there.

Let's load the packages we'll be using.

```
library(dplyr) # for working with data
```

```
##
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
library(rnaturalearth) # for map data
library(rnaturalearthdata) # for map data
```

```
##
## Attaching package: 'rnaturalearthdata'
```

```
## The following object is masked from 'package:rnaturalearth':
##
##     countries110
```

```r
library(ggplot2) # for plotting
library(sf) # for working with spatial vector data
```

```
## Linking to GEOS 3.13.0, GDAL 3.8.5, PROJ 9.5.1; sf_use_s2() is TRUE
```

```r
library(geosphere) # distance calculations
library(adehabitatHR) # for calculating utilisation distributions
```

```
## Loading required package: sp
```

```
## Loading required package: ade4
```

```
## Loading required package: adehabitatMA
```

```
## Registered S3 methods overwritten by 'adehabitatMA':
##   method                      from
##   print.SpatialPixelsDataFrame sp
##   print.SpatialPixels          sp
```

```
## Loading required package: adehabitatLT
```

```
##
## Attaching package: 'adehabitatLT'
```

```
## The following object is masked from 'package:dplyr':
##
##     id
```

```r
library(terra) # for working with rasters
```

```
## terra 1.8.70
```

```
##
## Attaching package: 'terra'
```

```
## The following object is masked from 'package:adehabitatMA':
##
##     buffer
```

```r
library(tidyterra) # for plotting terra rasters in ggplot
```

```
##
## Attaching package: 'tidyterra'
```

```
## The following object is masked from 'package:stats':
##
##     filter
```

```r
library(EMbC) # for classifying behaviour along tracks
library(marmap) # for bathymetry data
```

```
##
## Attaching package: 'marmap'
```

```
## The following object is masked from 'package:terra':
##
##     as.raster
```

```
## The following object is masked from 'package:grDevices':
##
##     as.raster
```

First, let's read in the tracking data directly from a URL.

```
# Instead of reading in the data from a file on our computers, we can point read.csv() directly to the
tracks <- read.csv("https://github.com/ryanreisinger/giantPetrels/raw/master/Data/GP_tracks_2019-07-01.
```

Let's take a look at the top of the file.

```
# The head() function shows us the first 6 rows of a dataframe.
head(tracks)
```

```
##   sp_code       scientific_name     individual_id Culmen_length Culmen_depth
## 1      NG Northern Giant Petrel NGP01_KD_SEP_2015         100.1         40.4
## 2      NG Northern Giant Petrel NGP01_KD_SEP_2015         100.1         40.4
## 3      NG Northern Giant Petrel NGP01_KD_SEP_2015         100.1         40.4
## 4      NG Northern Giant Petrel NGP01_KD_SEP_2015         100.1         40.4
## 5      NG Northern Giant Petrel NGP01_KD_SEP_2015         100.1         40.4
## 6      NG Northern Giant Petrel NGP01_KD_SEP_2015         100.1         40.4
##   breeding_stage deployment_site deployment_decimal_latitude
## 1     Incubation       Kildalkey                  37.8553287
## 2     Incubation       Kildalkey                  37.8553287
## 3     Incubation       Kildalkey                  37.8553287
## 4     Incubation       Kildalkey                  37.8553287
## 5     Incubation       Kildalkey                  37.8553287
## 6     Incubation       Kildalkey                  37.8553287
##   deployment_decimal_longitude device_type device_id       date      time
## 1                    -46.95416         GPS      5084 2015/09/15  13:55:14
## 2                    -46.95416         GPS      5084 2015/09/15  13:55:15
## 3                    -46.95416         GPS      5084 2015/09/15  13:55:16
## 4                    -46.95416         GPS      5084 2015/09/15  13:55:17
## 5                    -46.95416         GPS      5084 2015/09/15  13:55:18
## 6                    -46.95416         GPS      5084 2015/09/15  13:55:19
##   decimal_latitude decimal_longitude location_quality
## 1        -46.95463          37.85330               NA
## 2        -46.95462          37.85330               NA
## 3        -46.95462          37.85330               NA
## 4        -46.95461          37.85329               NA
## 5        -46.95461          37.85329               NA
## 6        -46.95460          37.85329               NA
##   latitude_uncertainty_metres longitude_uncertainty_metres          track_id
## 1                          NA                           NA NGP01_KD_SEP_2015
## 2                          NA                           NA NGP01_KD_SEP_2015
## 3                          NA                           NA NGP01_KD_SEP_2015
## 4                          NA                           NA NGP01_KD_SEP_2015
## 5                          NA                           NA NGP01_KD_SEP_2015
## 6                          NA                           NA NGP01_KD_SEP_2015
##     datetime trip
## 1 1442318114    0
## 2 1442318115    0
## 3 1442318116    0
## 4 1442318117    0
## 5 1442318118    0
## 6 1442318119    0
```

Notice that the date and time are in two separate columns and they are also character vectors (notice the `<chr>` tag under the column names, and think back to your first lab, on data types).

R has some spatial classes for date-time data, and many tracking packages use these. So, let's first join those two columns (`date` and `time`) into a new column (`date_time`) and convert that new column to a column with class `POSIXlt`, (one of the classes used in R for dates and times). For more on working dates, look at this section from the R for Data Science book, or this section from R Programming for Data Science.

```
# 'date' and 'time' are currently character vectors
class(tracks$date)
```

```
## [1] "character"
```

```
class(tracks$time)
```

```
## [1] "character"
```

```
# Join these two columns in a new column, using the paste() function
tracks$date_time <- paste(tracks$date, tracks$time, sep = " ")

# And convert to a date using the 'strptime' function
# Notice that we set the timezone (argument 'tz') as 'UTC' (GMT)
tracks$date_time <- strptime(tracks$date_time, format = "%Y/%m/%d %H:%M:%S", tz = "UTC")

# Look at the class of the new column, to check
class(tracks$date_time)
```

```
## [1] "POSIXlt" "POSIXt"
```

```
# Also look at the first 6 entries of the column to check
# that the date has been created correctly (if not, you would see NAs or other # weird things
head(tracks$date_time)
```

```
## [1] "2015-09-15 13:55:14 UTC" "2015-09-15 13:55:15 UTC"
## [3] "2015-09-15 13:55:16 UTC" "2015-09-15 13:55:17 UTC"
## [5] "2015-09-15 13:55:18 UTC" "2015-09-15 13:55:19 UTC"
```

For this analysis, we're only interested in a few of the columns from the data frame, so let's select only those.

`track_id` is the unique id of each track, `date_time` is the date and time, UTC, in POSXIct format (which we just created), and `decimal_longitude` and `decimal_latitude` are the longitude and latitude, respectively, in the WGS 1984 coordinate reference system that GPSs use (remember the lab on coordinate reference systems).

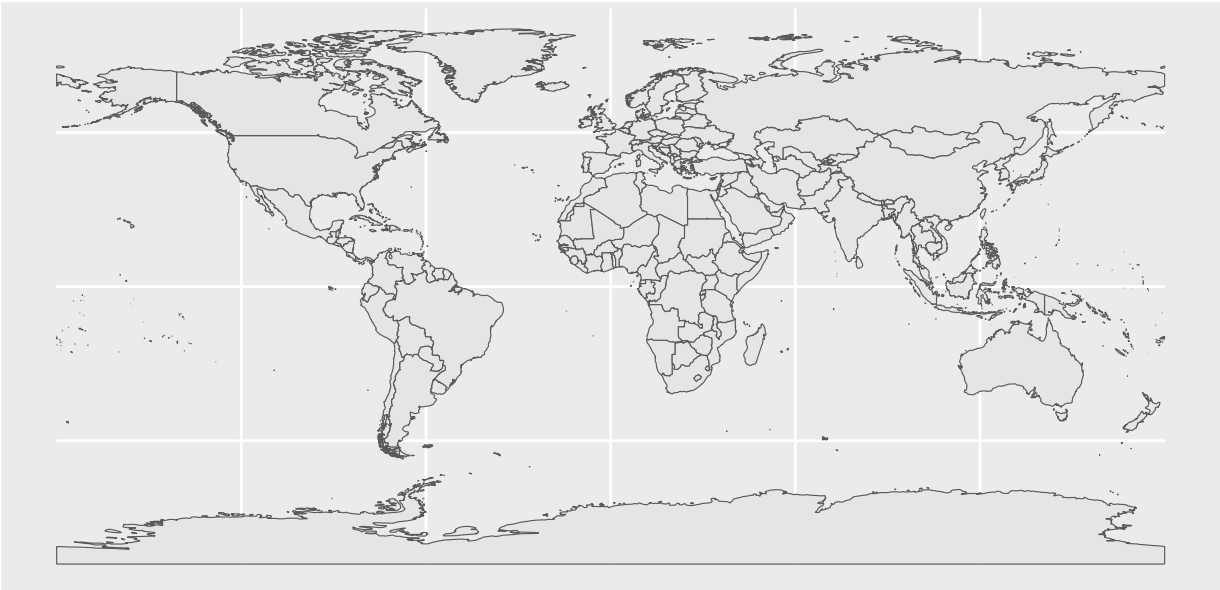
```
# Note that I've used dplyr::select() to force R to use the 'select' function from the 'dplyr' package.
tracks <- dplyr::select(tracks,
                        scientific_name,
                        individual_id,
                        date_time,
                        decimal_longitude,
                        decimal_latitude)

# Look at the first lines of our new data frame, 'tracks'
head(tracks)
```

```
##         scientific_name     individual_id           date_time decimal_longitude
## 1 Northern Giant Petrel NGP01_KD_SEP_2015 2015-09-15 13:55:14          37.85330
## 2 Northern Giant Petrel NGP01_KD_SEP_2015 2015-09-15 13:55:15          37.85330
## 3 Northern Giant Petrel NGP01_KD_SEP_2015 2015-09-15 13:55:16          37.85330
## 4 Northern Giant Petrel NGP01_KD_SEP_2015 2015-09-15 13:55:17          37.85329
## 5 Northern Giant Petrel NGP01_KD_SEP_2015 2015-09-15 13:55:18          37.85329
## 6 Northern Giant Petrel NGP01_KD_SEP_2015 2015-09-15 13:55:19          37.85329
```

```
##   decimal_latitude
## 1        -46.95463
## 2        -46.95462
## 3        -46.95462
## 4        -46.95461
## 5        -46.95461
## 6        -46.95460
```

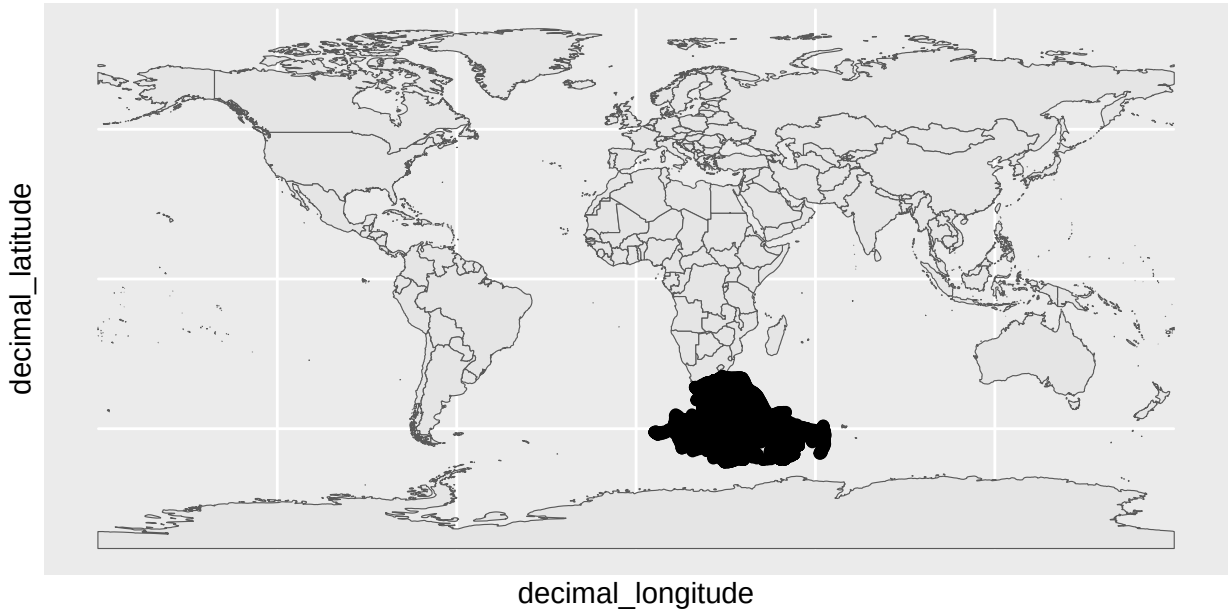
We can make a quick map of all the tracks.

```
# First let's get some map data from the 'rnaturalearth' package.
# Note that we ask the function to return the data in 'sf' class,
# which works with the 'sf' package.
world <- ne_countries(scale = "medium", returnclass = "sf")

# Map
ggplot(data = world) +
    geom_sf()
```



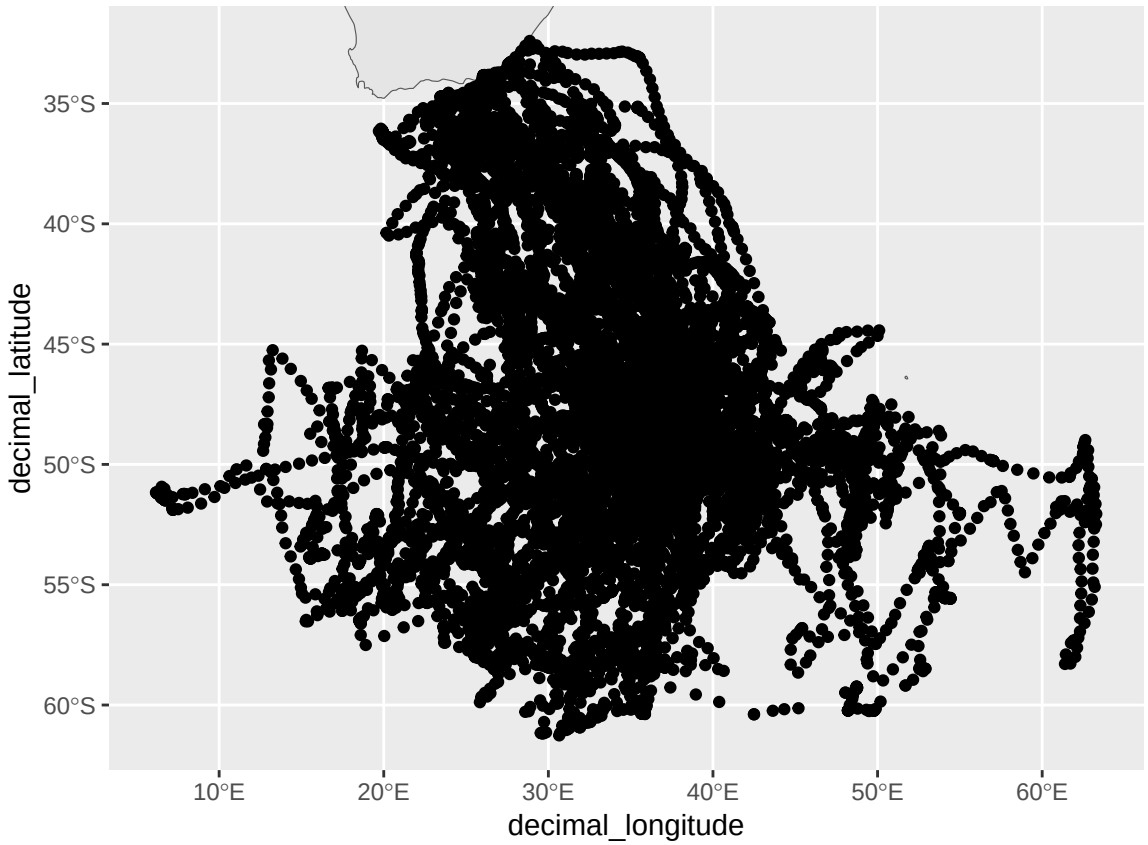
```
# And let's add our petrel tracks on top
ggplot(data = world) +
    geom_sf() +
  geom_point(data = tracks, aes(x = decimal_longitude,
                                y = decimal_latitude))
```

```
## Warning: Removed 53 rows containing missing values or values outside the scale range
## (`geom_point()`).
```




```
# Let's 'zoom in' on the tracks by limiting the spatial extent of the map
# according to the tracking data
# First, we set up the minimum and maximum longitudes (x) and latitudes (y)
min_x <- min(tracks$decimal_longitude, na.rm = T)
max_x <- max(tracks$decimal_longitude, na.rm = T)
min_y <- min(tracks$decimal_latitude, na.rm = T)
max_y <- max(tracks$decimal_latitude, na.rm = T)

# We add this spatial extent (these limits) using the 'coord_sf' function
ggplot(data = world) +
    geom_sf() +
  geom_point(data = tracks, aes(x = decimal_longitude,
                                y = decimal_latitude)) +
  coord_sf(xlim = c(min_x, max_x),
           ylim = c(min_y, max_y))
```

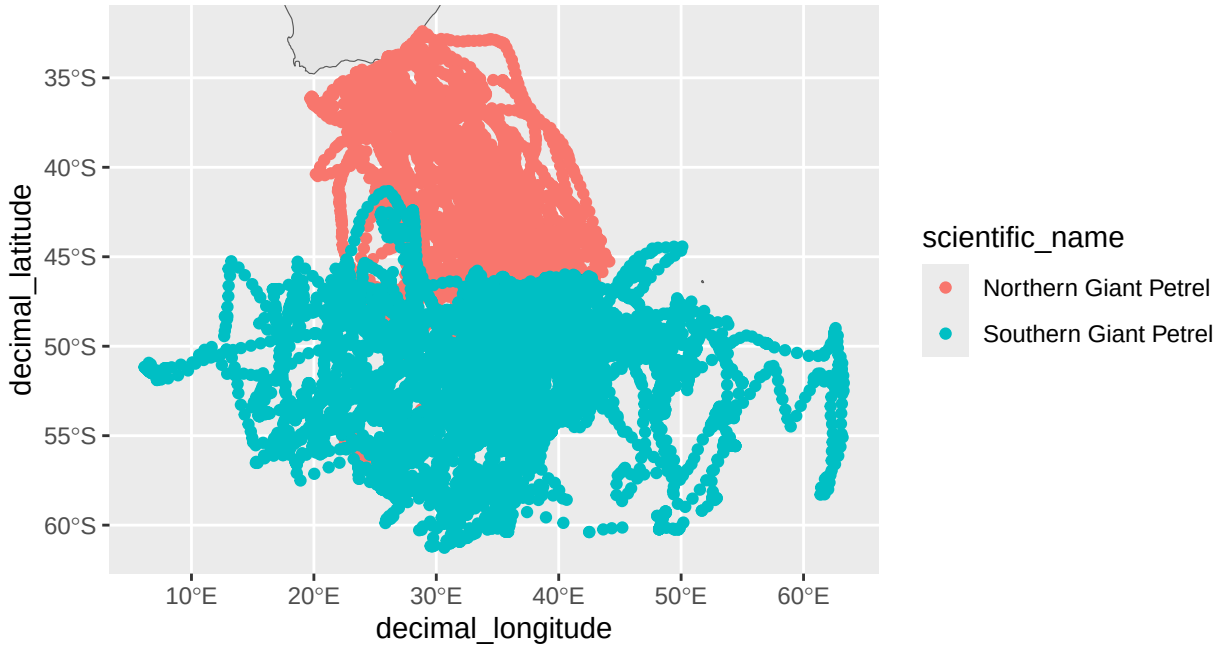
```
## Warning: Removed 53 rows containing missing values or values outside the scale range
## (`geom_point()`).
```



You can look through this tutorial for more ways to improve and customise your maps. But for the moment, let's customise one more thing on our map: we'll specify the tracks to be coloured according to the species, using arguments in the `aes()` function in `ggplot`. Notice how different the movements of northern and southern giant petrels are.

```
ggplot(data = world) +
    geom_sf() +
  geom_point(data = tracks, aes(x = decimal_longitude,
                                y = decimal_latitude,
                                colour = scientific_name)) +
  coord_sf(xlim = c(min_x, max_x),
           ylim = c(min_y, max_y))
```

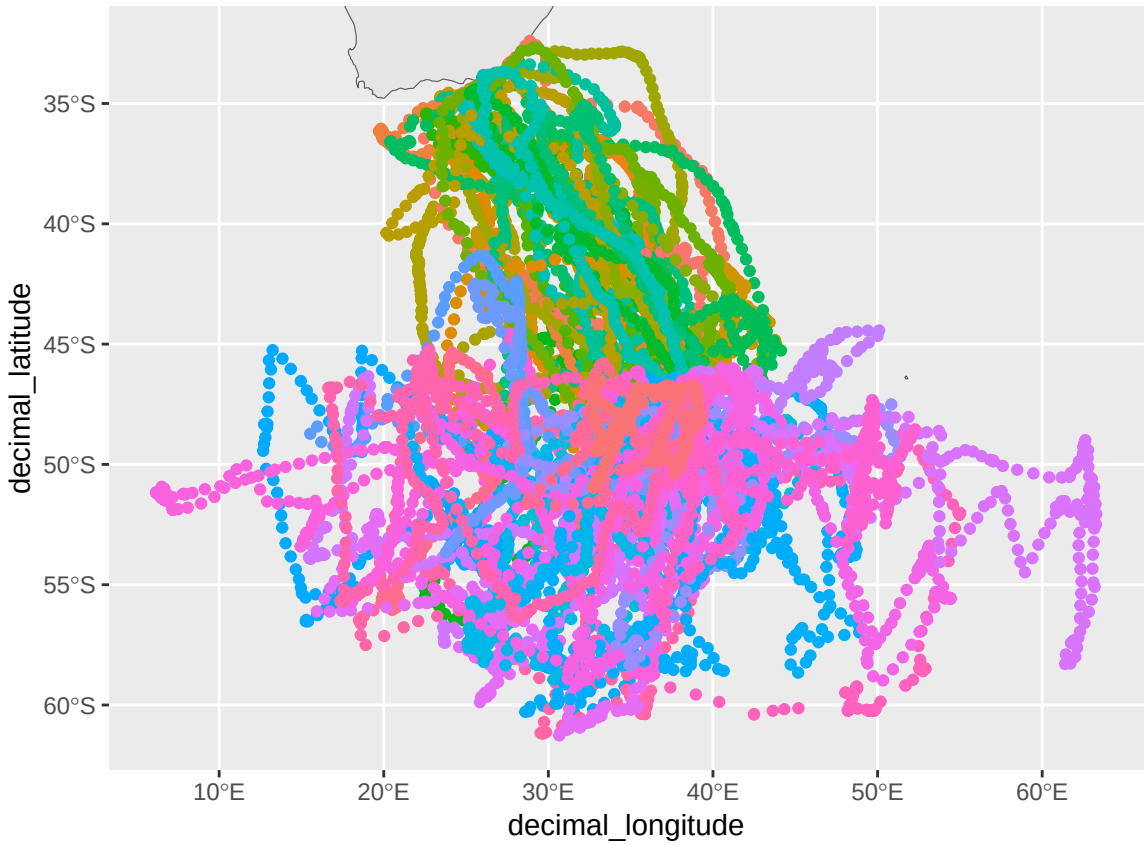
```
## Warning: Removed 53 rows containing missing values or values outside the scale range
## (`geom_point()`).
```



We could use the same approach to colour the tracks by individual. Notice that I switch off the legend in `theme`. There are so many individuals that the legend would take up too much space.

```
ggplot(data = world) +
    geom_sf() +
  geom_point(data = tracks, aes(x = decimal_longitude,
                                y = decimal_latitude,
                                colour = individual_id)) +
  coord_sf(xlim = c(min_x, max_x),
           ylim = c(min_y, max_y)) +
   theme(legend.position="none")
```

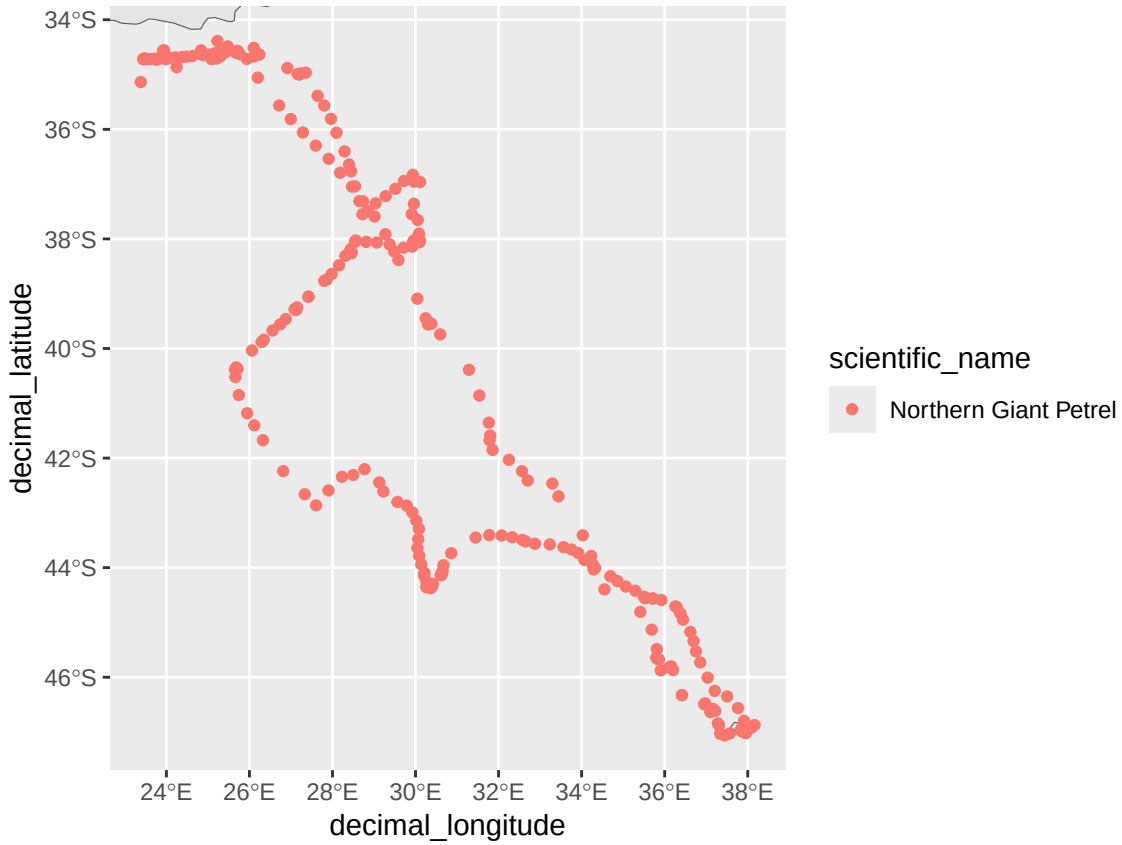
```
## Warning: Removed 53 rows containing missing values or values outside the scale range
## (`geom_point()`).
```



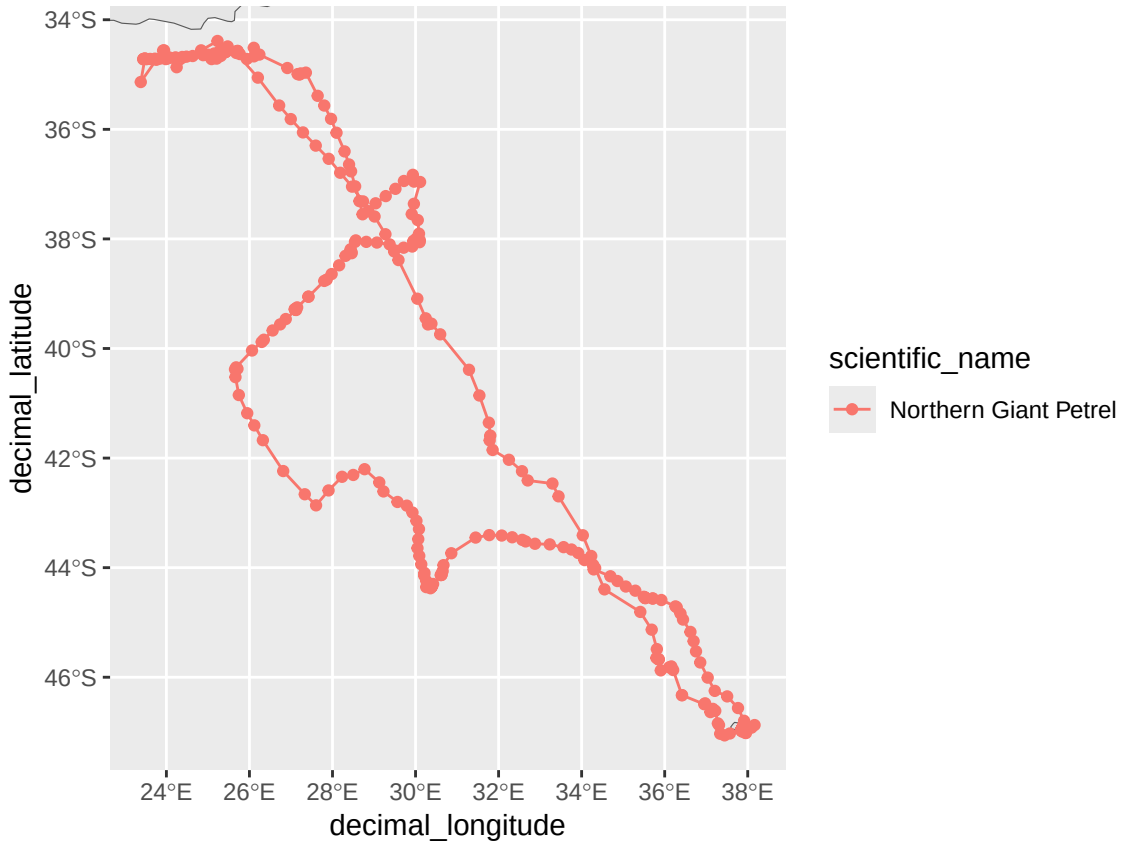
Basic analyses

For the following analyses, we'll select one individual to work with. In a 'real' analysis, you would work with all individuals. Let's use one of the northern giant petrels, with track id `NGP03_KD_SEP_2015`.

```
# Select one individual, creating a new dataframe called 'ngp'
# Again, we use a dplyr function called 'filter'
ngp <- dplyr::filter(tracks, individual_id == "NGP06_KD_SEP_2015")

# Instead of recalculating the x and y limits manually, we'll calculate
# them 'on the fly' inside the ggplot call, using the 'min' and 'max'
# functions.

ggplot(data = world) +
    geom_sf() +
  geom_point(data = ngp, aes(x = decimal_longitude,
                             y = decimal_latitude,
                             colour = scientific_name)) +
  coord_sf(xlim = c(min(ngp$decimal_longitude, na.rm = T),
                    max(ngp$decimal_longitude, na.rm = T)),
           ylim = c(min(ngp$decimal_latitude, na.rm = T),
                    max(ngp$decimal_latitude, na.rm = T)))
```




```
# We could also make a plot with lines under the points,
# using the geom_path() layer
ggplot(data = world) +
    geom_sf() +
    geom_path(data = ngp, aes(x = decimal_longitude,
                              y = decimal_latitude,
                              colour = scientific_name)) +
  geom_point(data = ngp, aes(x = decimal_longitude,
                              y = decimal_latitude,
                              colour = scientific_name)) +
  coord_sf(xlim = c(min(ngp$decimal_longitude, na.rm = T),
                    max(ngp$decimal_longitude, na.rm = T)),
           ylim = c(min(ngp$decimal_latitude, na.rm = T),
                    max(ngp$decimal_latitude, na.rm = T)))
```



We can calculate some basic parameters from the track, starting with the duration of the track and its spatial extent.

```
# Start and end date-time
start_date <- min(ngp$date_time)
end_date <- max(ngp$date_time)

# We can do algebra with dates
track_duration <- end_date - start_date
track_duration
```

```
## Time difference of 15.65057 days
```

```
# Note that track_duration is an object of class 'difftime' (time difference)
class(track_duration)
```

```
## [1] "difftime"
```

```
# We could also do the following, if we want to control the units
# of the output
track_duration <- difftime(time1 = end_date,
                           time2 = start_date,
                           units = "hours")

track_duration
```

```
## Time difference of 375.6136 hours
```

```
track_duration <- difftime(time1 = end_date,
                           time2 = start_date,
```

```
                            units = "days")

track_duration
```

```
## Time difference of 15.65057 days
```

Let's look at distance traveled, for each 'step' (that is, between each pair of locations) and in total (that is, the sum of all the steps). For this, we use the `geosphere` package.

```
# Create a new column called 'distance'.
ngp$distance <- distGeo(p1 = ngp[,c("decimal_longitude", "decimal_latitude")])

# Distance is in meters, divide by 1000 to get distance in km
ngp$distance <- ngp$distance/1000

# Total distance travelled in km, by summing all the steps
sum(ngp$distance, na.rm = T) # We need na.rm = TRUE to remove NAs - the last value is NA
```

```
## [1] 5570.265
```

Now let's look at the time steps, whereafter we can calculate speed (because we have just calculated distance, and speed is simply distance divided by time).

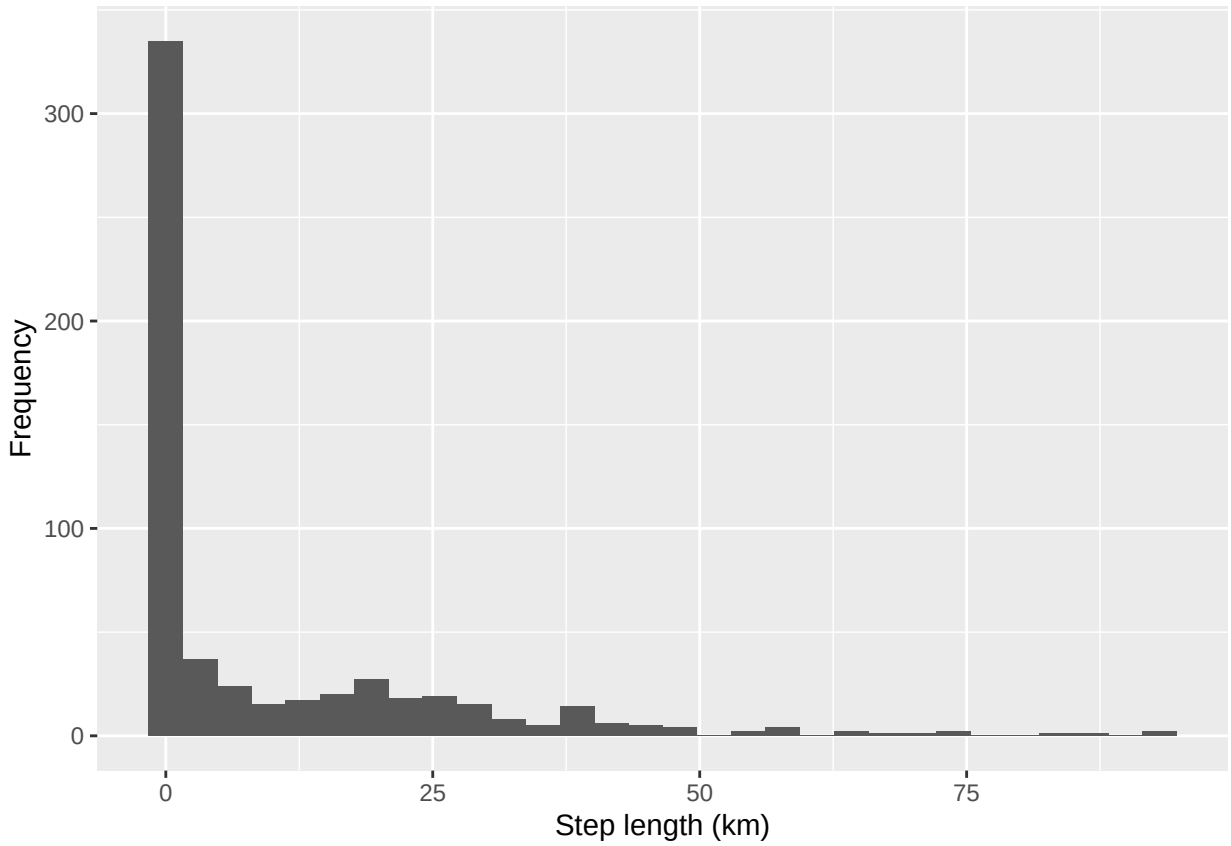
```
# First we calculate the time differenc between each pair of locations
# (that is, the step duration)
timediff <- diff(ngp$date_time)
units(timediff) <- "hours"
timediff <- as.numeric(timediff)
timediff <- c(timediff, NA)
ngp$timestep <- timediff

# We needed to do a few things here
# 1) We changed the output from the 'diff' function into hours (it was in seconds)
# 2) We coerced the result into a nummeric (it was a 'difftime' class)
# 3) We added an NA at the end, since the diff funciton is not as clever as
# distGeo, and doesn't automatically add it.
```

Let's plot the frequency distributions (histograms) of step lengths and step durations. You will see in the histogram for step duration, there is a big outlier (>15 hours) that we would normally deal with by filtering or some kind of inspection.

```
ggplot(data = ngp,
       aes(x = distance)) +
         geom_histogram() +
  labs(main = "Step length distribution",
       x = "Step length (km)",
       y = "Frequency")
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```

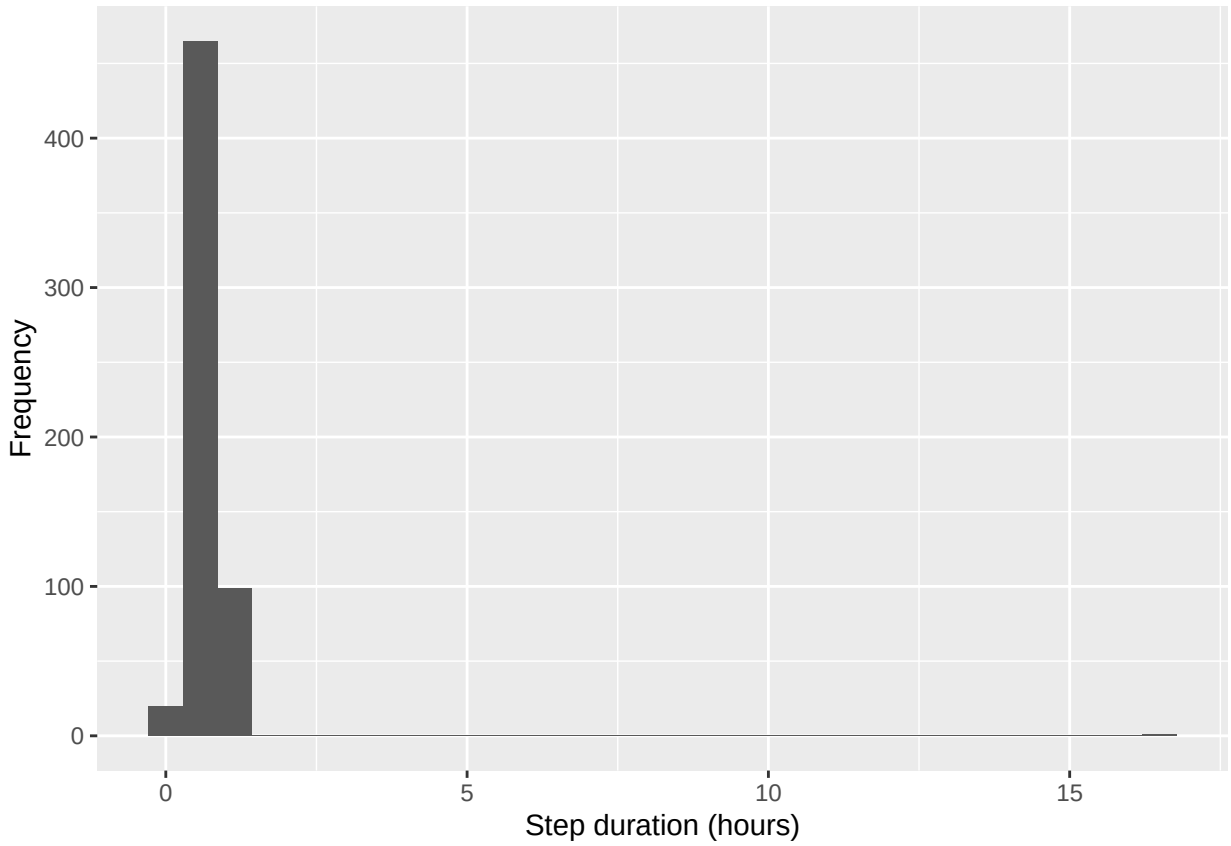
```
## Warning: Removed 1 row containing non-finite outside the scale range
## (`stat_bin()`).
```




```
ggplot(data = ngp,
       aes(x = timestep)) +
         geom_histogram() +
  labs(main = "Step duration distribution",
       x = "Step duration (hours)",
       y = "Frequency")
```

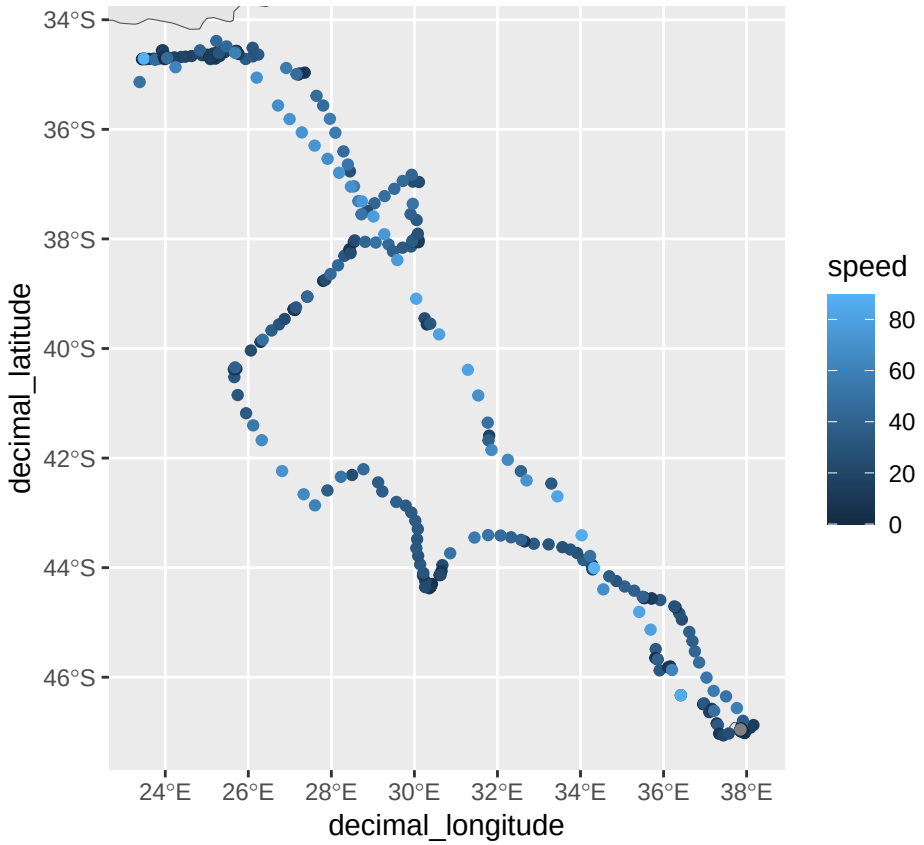
```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```

```
## Warning: Removed 1 row containing non-finite outside the scale range
## (`stat_bin()`).
```



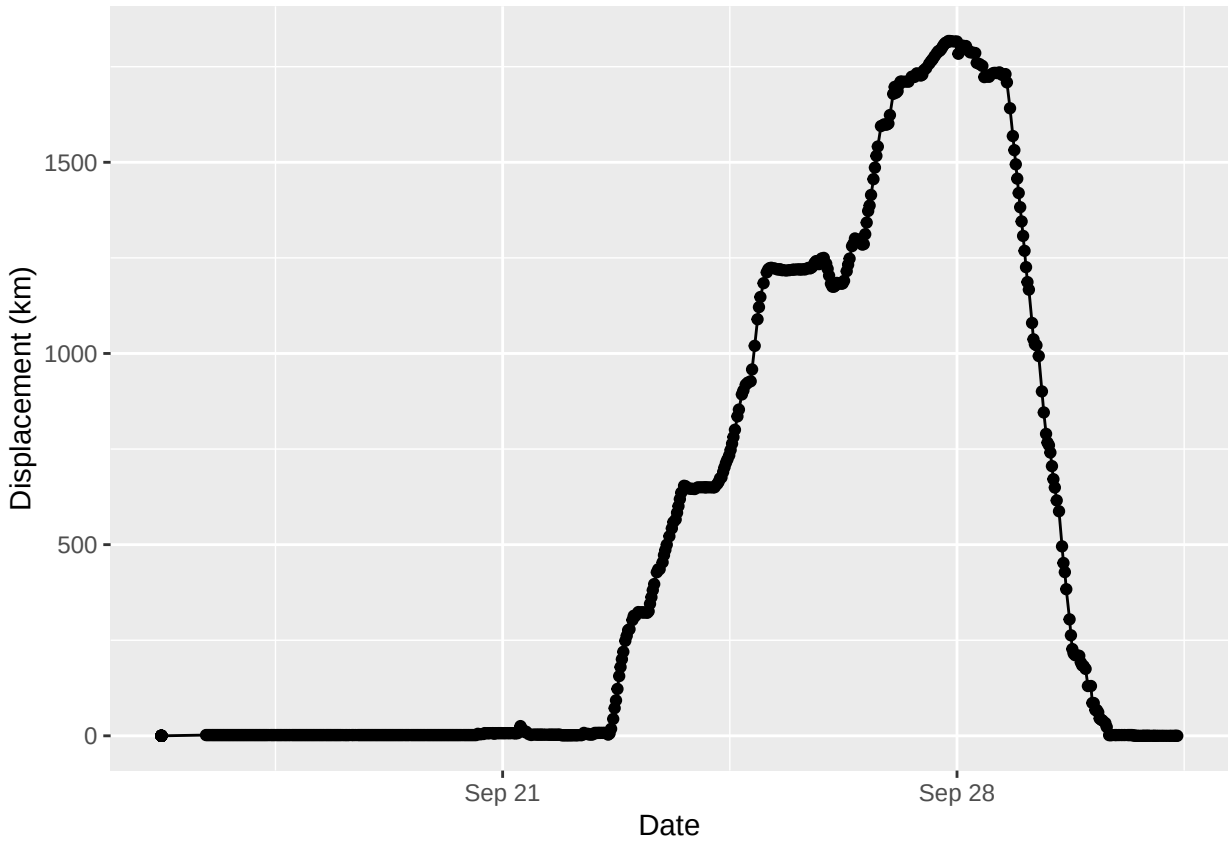
Since we calculated step length and duration, we can work out speed in km/h, and colour our map with that value. Note that in the call to ggplot, we replace the aesthetic (`aes`), `scientific_name` (which we previously used to colour by species) with `speed`, so we can colour each point by speed.

```
ngp$speed <- ngp$distance / ngp$timestep

ggplot(data = world) +
    geom_sf() +
  geom_point(data = ngp, aes(x = decimal_longitude,
                             y = decimal_latitude,
                             colour = speed)) +
  coord_sf(xlim = c(min(ngp$decimal_longitude, na.rm = T),
                    max(ngp$decimal_longitude, na.rm = T)),
           ylim = c(min(ngp$decimal_latitude, na.rm = T),
                    max(ngp$decimal_latitude, na.rm = T)))
```



Related to the step length (distance), is calculating the displacement. This is, at each location along the track, the distance of that point from the track's origin. Again, we use the `distGeo` function, but this time we calculate the distance to a pair of 'home' coordinates (the start of the track), since we can think of displacement as the distance from 'home' at each point.

```
# First we define 'home' as the coordinates at the start of the track
home <- ngp[1, c("decimal_longitude", "decimal_latitude")] # Select row 1 of the data

# Now we use the distGeo function again
# Here the function will look at each set of coordinates (p1) in the dataframe and calculate how far th
ngp$displacement <- distGeo(p1 = ngp[,c("decimal_longitude", "decimal_latitude")],
                            p2 = home)

# Remember to divide by 1000 to convert from distance in m to km
ngp$displacement <- ngp$displacement/1000

# And we can plot this over time to get a displacement plot
ggplot(data = ngp,
       aes(x = as.POSIXct(date_time), y = displacement)) +
       geom_path() +
         geom_point() +
  labs(main = "Displacement plot",
       x = "Date",
       y = "Displacement (km)")
```



In the plot we can see that the line is initially flat (near-zero displacement), so lots of the first locations are likely to be while the bird was still sitting on its nest, after the tag was deployed on it. In a thorough analysis we would trim this first part of the track, manually or programatically, to remove these locations on the nest (or on the beach or shore for seals). Techniques called 'net squared displacement' and 'mean squared displacement' are sometime used to determine the kind of migratory behavior that animals are displaying, from these plots of displacement. See Singh et al. (2016) for more details.

Calculating utilization distributions

We'll use the `adehabitatHR` package (https://cran.r-project.org/web/packages/adehabitatHR/index.html) to calculate utilisation distributions for our single track. Remember the methods I introduced to you in Lecture 07 (Figure 7.9 in Pittman Chapter 7). There are three 'sibling' adehabitat packages for different analyses (Calenge 2006). Unfortunately, they all still rely on the `sp` package, which you will recall is the older version of `sf`. Nonetheless, they remain a useful and comprehensive set of tools.

Minimum Convex Polygons

Minimum convex polygons (MCPs) are one of the simplest estimators of spatial utilisation. MCPs are 'the smallest polygon around points with all interior angles less than 180 degrees. MCPs are common estimators of home range, but can potentially include area not used by the animal and overestimate the home range' (from https://jamesepaterson.github.io/jamespatersonblog/03_trackingworkshop_homeranges). Remember from the animal movement lecture where I spoke about overestimation.

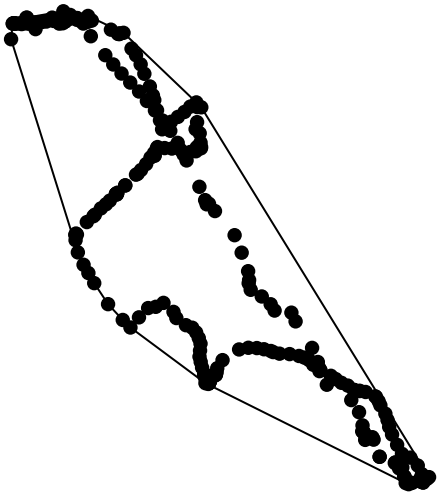
Calculating utilisation distributions with adehabitat takes a little bit of preparation. Let's calculate MCPs...

```
# First, we need to create an object of class 'spatial points' (a vector format) to use in the mcp func

# Load the sp library
```

```
library(sp)

# Make a copy of our tracks
ngp_sp <- ngp

# The mcp function only allows one extra column, the animal id, so we
# select only three columns from our dataframe. The latitude, the longitude, and the animal ID (remembe
ngp_sp <- dplyr::select(ngp_sp,
                        decimal_longitude,
                        decimal_latitude,
                        individual_id)

# Tell R the dataframe we just made has spatial coordinates
coordinates(ngp_sp) <- c("decimal_longitude", "decimal_latitude")

# And tell R what the coordinate reference system of the dataframe is
proj4string(ngp_sp) <- CRS("EPSG:4326")

# Now we see that the object has a new class: spatial points data frame
class(ngp_sp)
```

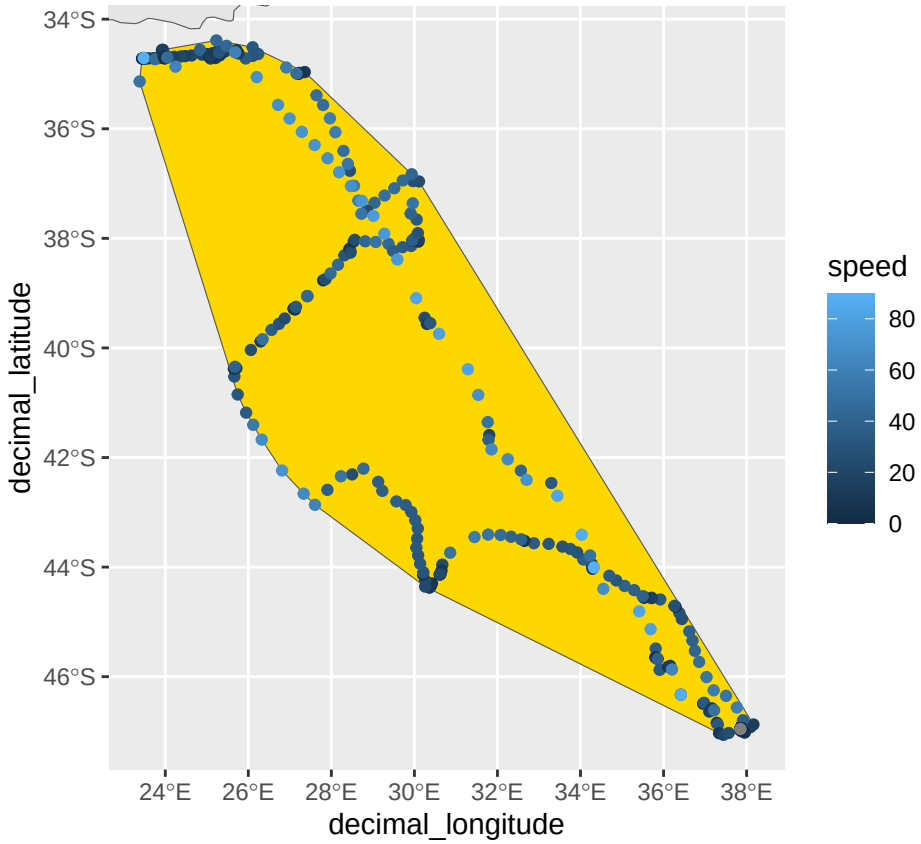
```
## [1] "SpatialPointsDataFrame"
## attr(,"package")
## [1] "sp"
```

```
# We can use this object as input for the MCP function in adehabitatHR
ngp_mcp <- mcp(ngp_sp, percent = 100)

# And we can plot the result, with the locations on top
plot(ngp_mcp)
plot(ngp_sp, add = TRUE, pch = 16)
```



```
# If we want to plot the mcp with ggplot, we need to convert
# it to an sf object
ngp_mcp_sf <- st_as_sf(ngp_mcp)

# And we plot it by adding a `geom_sf` layer to the
# kind of plot we made earlier
```

```
ggplot(data = world) +
    geom_sf() +
  geom_sf(data = ngp_mcp_sf, fill = "gold") +
    geom_point(data = ngp, aes(x = decimal_longitude,
                               y = decimal_latitude,
                               colour = speed)) +
    coord_sf(xlim = c(min(ngp$decimal_longitude, na.rm = T),
                      max(ngp$decimal_longitude, na.rm = T)),
             ylim = c(min(ngp$decimal_latitude, na.rm = T),
                      max(ngp$decimal_latitude, na.rm = T)))
```



Kernel density estimates

Next, we can calculate the kernel density estimate (kde).

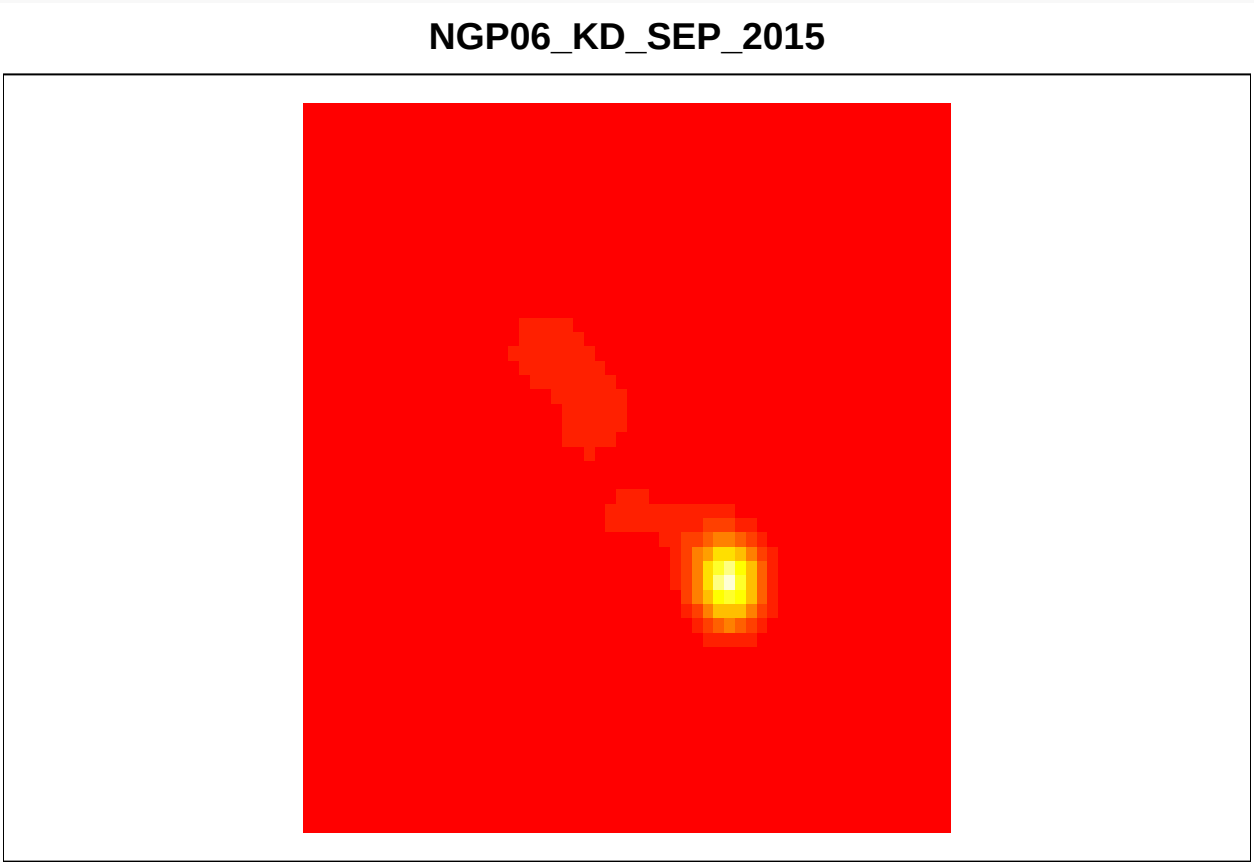
Calenge (2015) writes:

> The MCP has met a large success in the ecological literature. However, many authors have stressed that the definition of the home range which is commonly used in the literature was rather imprecise: "that area traversed by the animal during its normal activities of food gathering, mating and caring for young" (Burt, 1943). Although this definition corresponds well to the feeling of many ecologists concerning what is the home range, it lacks formalism: what is an area traversed? what is a normal activity? Several authors have therefore proposed to replace this definition by a more formal model: the utilization distribution (UD, van Winkle, 1975). Under this model, we consider that the animals use of space can be described by a bivariate probability density function, the UD, which gives the probability density to relocate the animal at any place according to the coordinates (x, y) of this place. The study of the space use by an animal could

consist in the study of the properties of the utilization distribution. The issue is therefore to estimate the utilization distribution from the relocation data. The seminal paper of Worton (1989) proposed the use of the kernel method (Silverman, 1986; Wand and Jones, 1995) to estimate the UD using the relocation data. The kernel method was probably the most frequently used function in the package adehabitat.

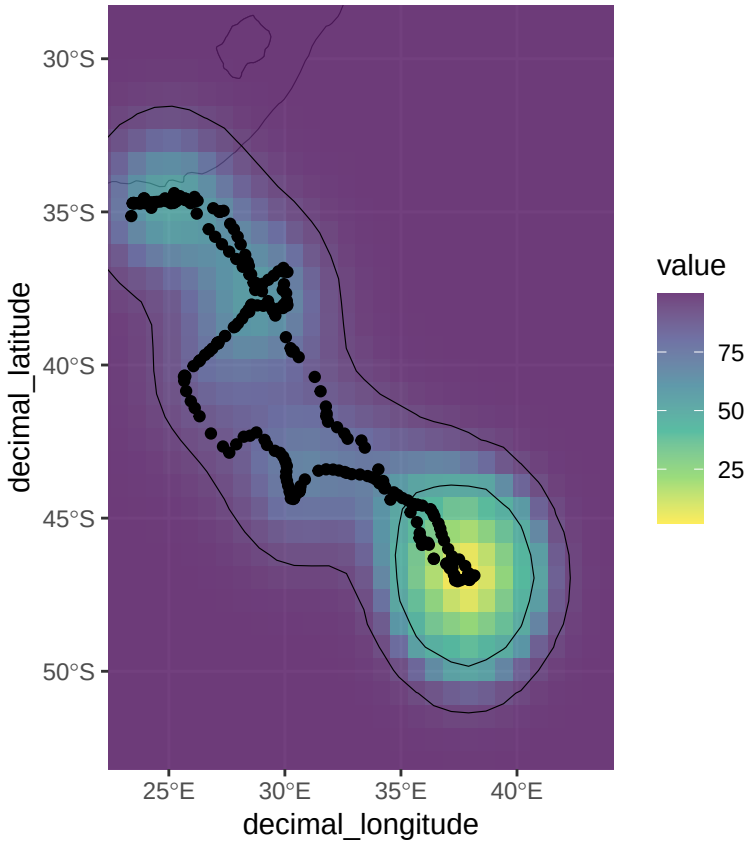
Let's go...

```
# We can calculate the kde using the same input we used for
# calculating the mcp above
ngp_kde <- kernelUD(ngp_sp, h = "href")

# The output has its own bespoke class, but we can plot it with
image(ngp_kde)
```




```
# We can create a raster-package raster with
ngp_vud <- getvolumeUD(ngp_kde)
ngp_kde_raster <- rast(as(ngp_vud$NGP06_KD_SEP_2015,"SpatialPixelsDataFrame"))

# Remember that the 95% kde is often designated the home range and the 50% kde
# the core range. To get a specific contour from
# the KDE we just caculated we do:
ngp_kde_95 <- getverticeshr(ngp_kde, percent = 95)
ngp_kde_50 <- getverticeshr(ngp_kde, percent = 50)

# And we can then convert these to sf objects for plotting,
# using the 'st_as_sf' function
ngp_kde_95 <- st_as_sf(ngp_kde_95)
```

```r
ngp_kde_50 <- st_as_sf(ngp_kde_50)

# Let's plot the kde and its contours in ggplot
# To plot the terra raster in ggplot we use the tidyterra package
# and its 'geom_spatraster'
# see https://www.r-bloggers.com/2022/05/introducing-tidyterra/ for more
ggplot(data = world) +
    geom_sf() +
  # first, the raster
  geom_spatraster(data = ngp_kde_raster) +
  scale_fill_viridis_c(direction = -1, alpha = 0.75) +
  # then, the kernel contours
  geom_sf(data = ngp_kde_50, colour = "black", fill = NA) +
    geom_sf(data = ngp_kde_95, colour = "black", fill = NA) +
  # then, the tracking data
    geom_point(data = ngp, aes(x = decimal_longitude,
                               y = decimal_latitude)) +
  # and then we 'zoom' the map -- note I expanded the area by 5 degrees
    coord_sf(xlim = c(min(ngp$decimal_longitude, na.rm = T)-5,
                      max(ngp$decimal_longitude, na.rm = T))+5,
             ylim = c(min(ngp$decimal_latitude, na.rm = T)-5,
                      max(ngp$decimal_latitude, na.rm = T)+5))
```



Remember that the **low kde value represents high density**. Notice how the kde overestimates the area (the mcp too), and how it ignores hard barriers like land (South Africa in the north-east of the plot). As I mentioned in lecture 7, and as stated in Pittman chapter 7, LoCoH (Getz et al. 2007) is an alternative to mcp, and movement-based kernel density estimation, or biased random bridge, (Benhamou & Cornelis 2010)

is an alternative to kde. We won't use these two methods in this tutorial, but you can calculate them in adehabitat. See `?LoCoH()` and `?BRB()` in adehabitatHR, for more information.

Behavioural classification

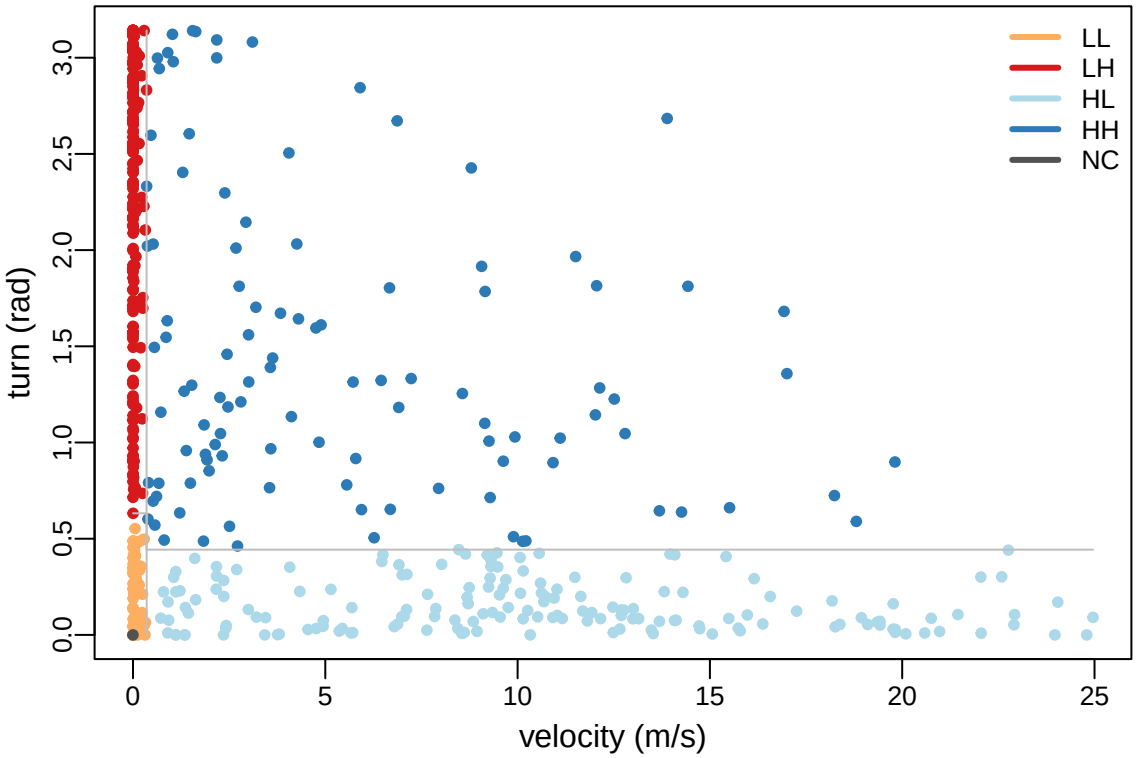
There are many ways of estimating behaviour from movement data. Figure 7.9 in Pittman lists several. We'll look at a method called Expectation Maximisation Binary Clustering (EMbC) (Garriga et al. 2016 https://doi.org/10.1371/journal.pone.0151984).

A quick start guide is at https://rdrr.io/cran/EMbC/f/vignettes/EMbC_qckref.Rmd

```
# EBbC expects the data in a specif format
# Look at the 'obj' argument in the help file ?stbc()
ngp_embc_data <- dplyr::select(ngp,
                               date_time,
                               decimal_longitude,
                               decimal_latitude)

# Run the function
mybcp <- stbc(ngp_embc_data, info=-1)
```

```
## [1]   0  -0.0000e+00       4       586
## [1] ... Stable clustering
```

```
# Inspect the clustering results
sctr(mybcp)
```

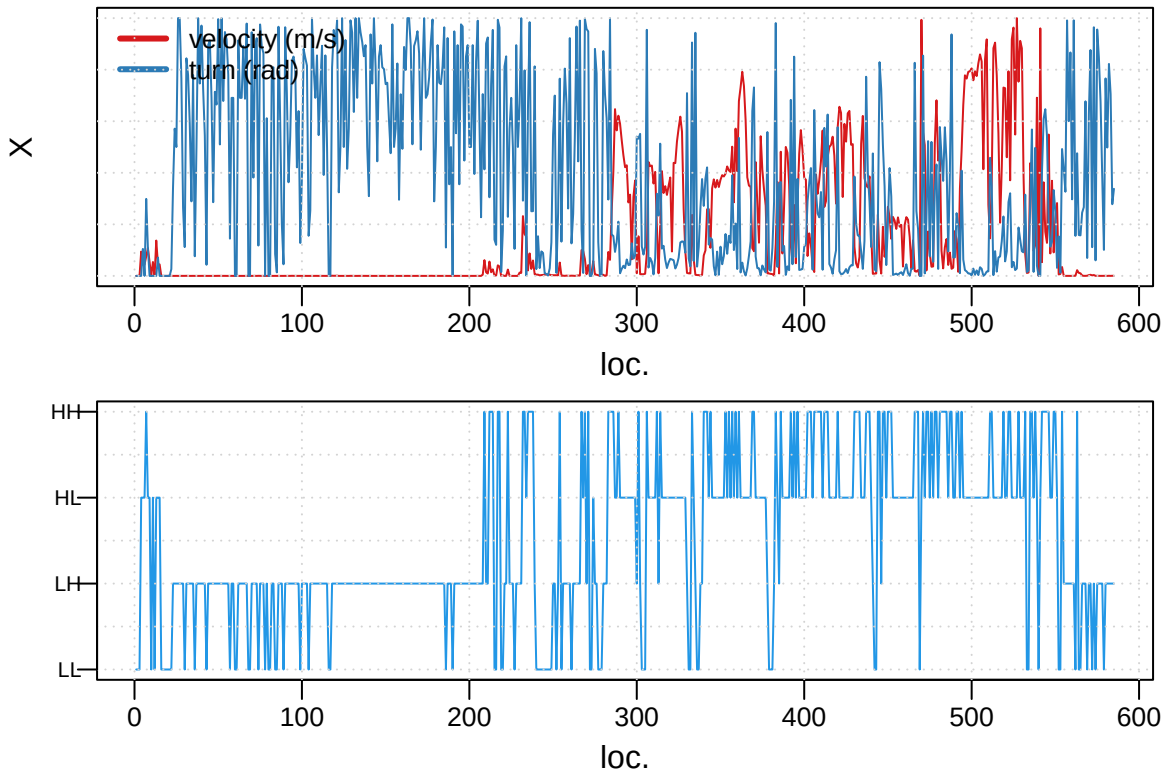


This plot shows us the each tracking location according to the turning angle (vertical axis) and velocity (horizontal axis) of the preceding step. The algorithm then performs bivariate (two variable - speed and angle) clustering, and clusters the locations into four categories according to low and high speed and turning angle. LL = low speed, low turning angle. LH = low speed, high turning angle. HL = high speed, low turning

angle, HH = high speed and high turning angle. NC = not classified.

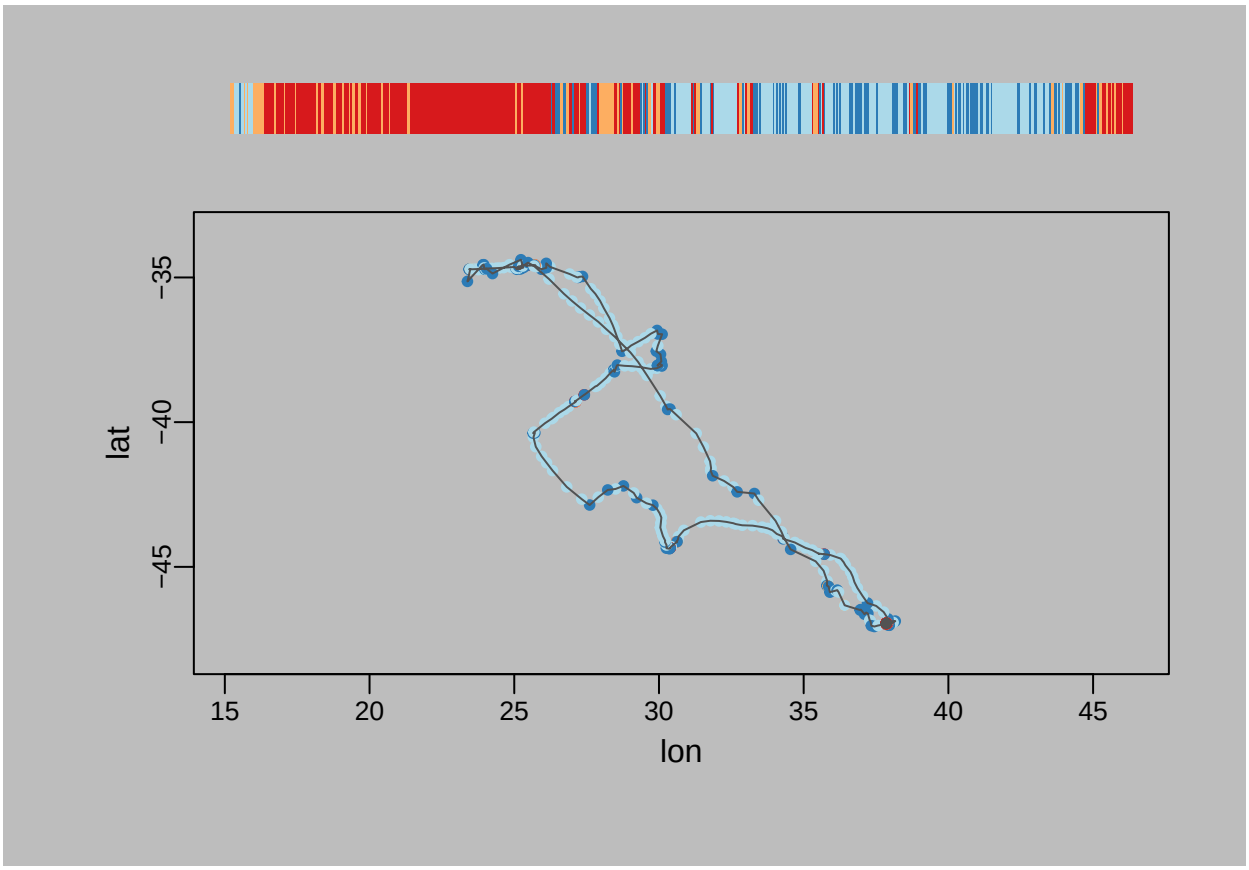
We can look at the labelled (annotated) trajectory over time.

```
lblp(mybcp)
```



Or map the labelled trajectory. Think about how these classifications relate to the animal's behaviour (for example, the Area Restricted Search behaviour we talked about in lectures 6 and 7).

```
view(mybcp)
```



We can get these specific outputs from the EMbC object, and add them to our data frame.

```
# We can look at the cutoff values for each classification with
mybcp@R
```

```
##         X1.min    X2.min     X1.max    X2.max
## 1.LL 0.0000000 0.0000000  0.3558894 0.6321050
## 2.LH 0.0000000 0.6321050  0.3536850 3.1415927
## 3.HL 0.3558894 0.0000000 24.9606512 0.4431287
## 4.HH 0.3536850 0.4431287 24.9606512 3.1415927
```

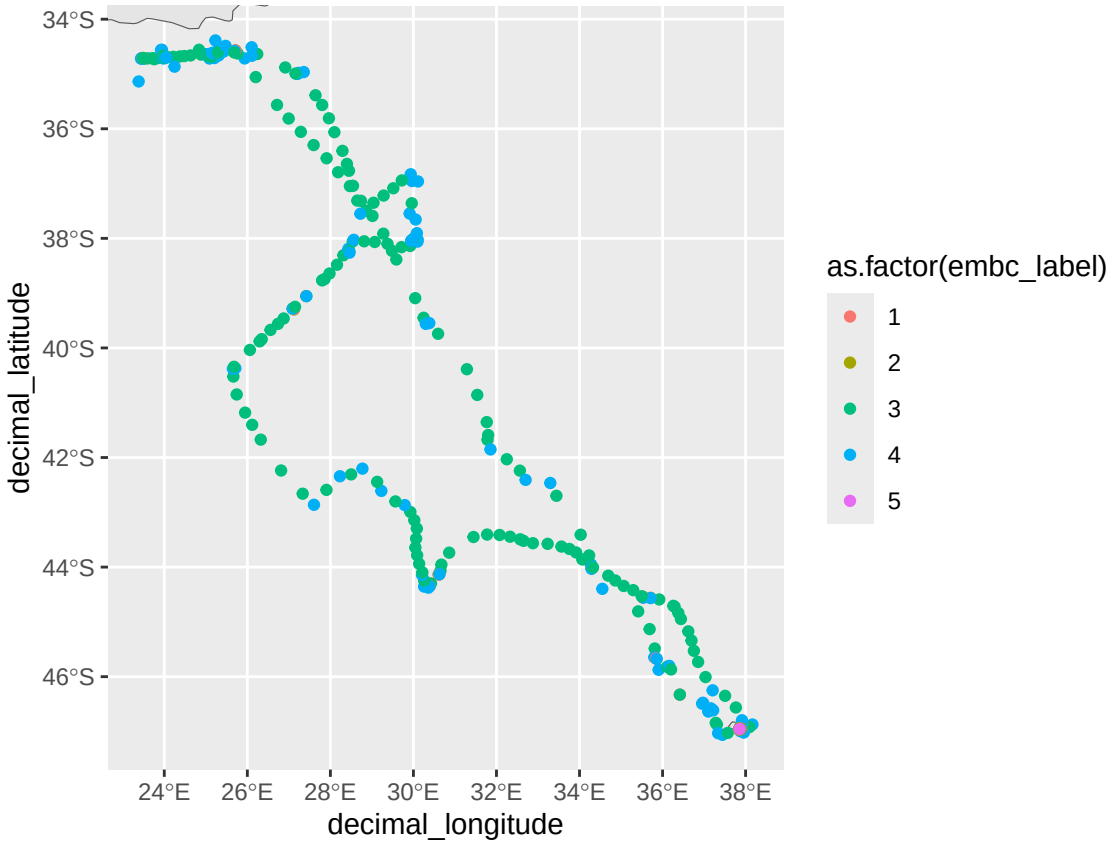
```
# Notice the '@' syntax. In some types of more complicated R objects, we use @ to access a particular '
str(mybcp) # See all the '@'s designating the parts or slots
```

```
## Formal class 'binClstPath' [package "EMbC"] with 19 slots
##   ..@ pth      :'data.frame':    586 obs. of  3 variables:
##   .. ..$ dTm: POSIXlt[1:586], format: "2015-09-15 17:49:44" "2015-09-15 17:49:45" ...
##   .. ..$ lon: num [1:586] 37.9 37.9 37.9 37.9 37.9 ...
##   .. ..$ lat: num [1:586] -47 -47 -47 -47 -47 ...
##   ..@ spn      : num [1:586] 1 1 1 1 1 1 1 1 1 1 ...
##   ..@ dst      : num [1:586] 0 0 0 2.35 1.11 ...
##   ..@ hdg      : num [1:586] 6.283 6.283 6.283 0.329 0 ...
##   ..@ bursted  : logi FALSE
##   ..@ tracks   :Formal class 'SpatialLinesDataFrame' [package "sp"] with 4 slots
##   .. .. ..@ data       :'data.frame':    0 obs. of  0 variables
## Formal class 'data.frame' [package "methods"] with 4 slots
##   .. .. .. .. ..@ .Data    : list()
##   .. .. .. .. ..@ names    : chr(0)
```

```
##   .. .. .. .. ..@ row.names: int(0)
##   .. .. .. .. ..@ .S3Class : chr "data.frame"
##   .. .. ..@ lines      : list()
##   .. .. ..@ bbox       : num[0 , 0 ]
##   .. .. ..@ proj4string:Formal class 'CRS' [package "sp"] with 1 slot
##   .. .. .. .. ..@ projargs: chr NA
##   ..@ midPoints:Formal class 'SpatialPointsDataFrame' [package "sp"] with 5 slots
##   .. .. ..@ data       :'data.frame':    0 obs. of  0 variables
##   .. .. ..@ coords.nrs : num(0)
##   .. .. ..@ coords     : logi [1, 1] NA
##   .. .. ..@ bbox       : logi [1, 1] NA
##   .. .. ..@ proj4string:Formal class 'CRS' [package "sp"] with 1 slot
##   .. .. .. .. ..@ projargs: chr NA
##   ..@ X        : num [1:586, 1:2] 0 0 0 2.35 1.11 ...
##   .. ..- attr(*, "dimnames")=List of 2
##   .. .. ..$ : NULL
##   .. .. ..$ : chr [1:2] "velocity (m/s)" "turn (rad)"
##   ..@ U        : num [1:586, 1:2] 1 1 1 1 1 1 1 1 1 1 ...
##   .. ..- attr(*, "dimnames")=List of 2
##   .. .. ..$ : NULL
##   .. .. ..$ : chr [1:2] "vlc.Certainty" "trn.Certainty"
##   ..@ stdv     : num [1:2] 0.1 0.0873
##   ..@ m        : int 2
##   ..@ k        : num 4
##   ..@ n        : int 586
##   ..@ R        : num [1:4, 1:4] 0 0 0.356 0.354 0 ...
##   .. ..- attr(*, "dimnames")=List of 2
##   .. .. ..$ : chr [1:4] "1.LL" "2.LH" "3.HL" "4.HH"
##   .. .. ..$ : chr [1:4] "X1.min" "X2.min" "X1.max" "X2.max"
##   ..@ P        :List of 4
##   .. ..$ :List of 2
##   .. .. ..$ M: num [1:2] 0.0629 0.1709
##   .. .. ..$ S: num [1:2, 1:2] 0.01 0 0 0.0378
##   .. ..$ :List of 2
##   .. .. ..$ M: num [1:2] 0.018 2.242
##   .. .. ..$ S: num [1:2, 1:2] 0.01 0 0 0.526
##   .. ..$ :List of 2
##   .. .. ..$ M: num [1:2] 10.676 0.143
##   .. .. ..$ S: num [1:2, 1:2] 41.5579 -0.1169 -0.1169 0.0175
##   .. ..$ :List of 2
##   .. .. ..$ M: num [1:2] 5.37 1.49
##   .. .. ..$ S: num [1:2, 1:2] 23.398 -1.209 -1.209 0.704
##   ..@ W        : num [1:586, 1:4] 9.80e-01 9.80e-01 9.80e-01 1.05e-112 9.50e-23 ...
##   ..@ A        : num [1:586] 1 1 1 3 3 3 4 3 3 1 ...
##   ..@ L        : num [1:31] -4.77 -4.51 -4.07 -3.66 -3.41 ...
##   ..@ C        : chr [1:5] "#FDAE61" "#D7191C" "#ABD9E9" "#2C7BB6" ...
```

```r
# We can write the outputs to the data frame
ngp_embc_data$embc_velocity <- mybcp@X[,1]
ngp_embc_data$embc_turnrad <- mybcp@X[,2]
ngp_embc_data$embc_label <- mybcp@A # These are the EMbC labels

# Let's map the labels
# Notice that I put as.factor() around the embc_label
```

```
# That's because embc returns the label as a numeric, but we want to treat it as a categorical factor.
ggplot(data = world) +
    geom_sf() +
  geom_point(data = ngp_embc_data, aes(x = decimal_longitude,
                                       y = decimal_latitude,
                                       colour = as.factor(embc_label))) +
  coord_sf(xlim = c(min(ngp_embc_data$decimal_longitude, na.rm = T),
                    max(ngp_embc_data$decimal_longitude, na.rm = T)),
           ylim = c(min(ngp_embc_data$decimal_latitude, na.rm = T),
                    max(ngp_embc_data$decimal_latitude, na.rm = T)))
```



Notice that there are five classes. The fifth class corresponds with locations that were not classified (NC in the first EMbC plot). Most location are in class 3 and 4. Class 4 seems to be associated with slower, more tortuous (low speed, high turning angles, or LH) movements than class 3, which looks like low turning angle and higher speed.

Environmental relationships

We can examine the relationship between animals' space use or movement behavior, and the environment. Again, there are many ways to do this, and many environmental variables we could look at, but let's use a very simple example, making use of what we have here, and what we did in previous weeks. We'll download some bathymetry data like we did in Lab 4, and look at how bathymetry corresponds with the EMbC labels.

Let's get the bathymetry data using `marmap`.

```
# First, we define the spatial extent, similar to what we did for the
# first plot. We use the min and max functions, on longitue (x)
```

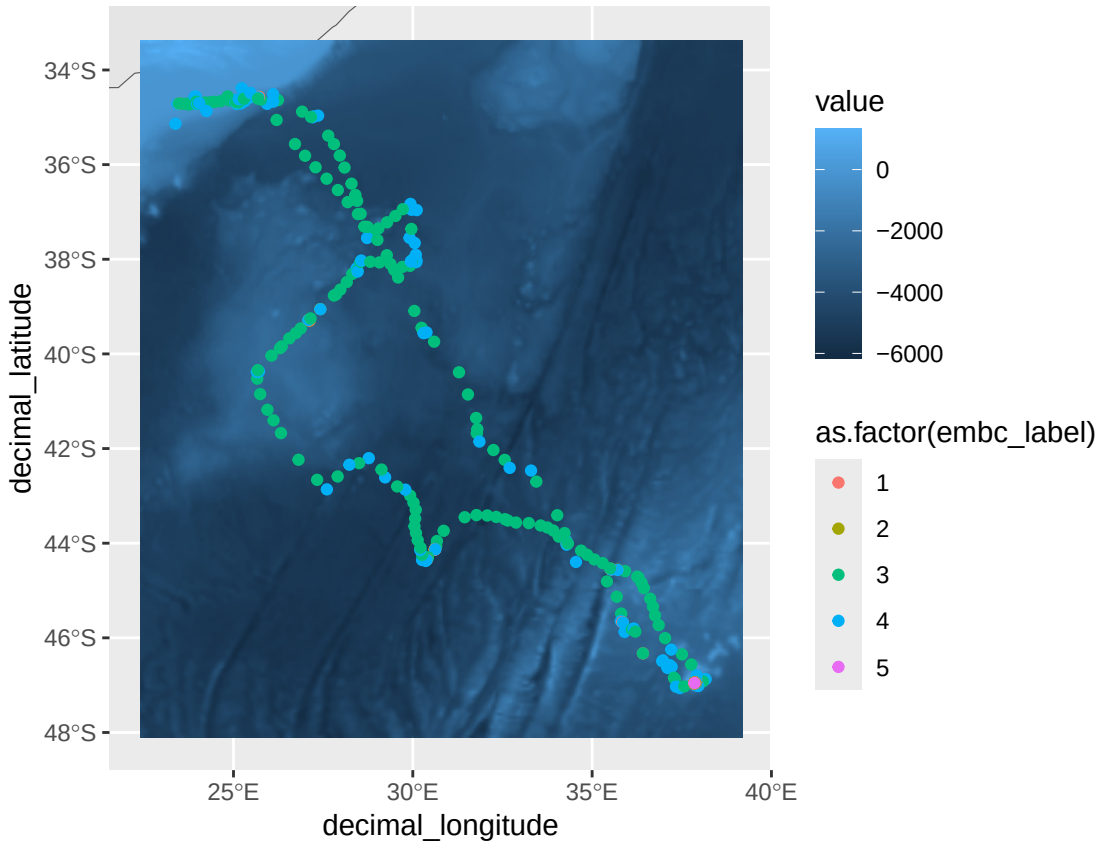
```
# and latitude (y)
min_x <- min(ngp_embc_data$decimal_longitude, na.rm = T) - 1
max_x <- max(ngp_embc_data$decimal_longitude, na.rm = T) + 1
min_y <- min(ngp_embc_data$decimal_latitude, na.rm = T) - 1
max_y <- max(ngp_embc_data$decimal_latitude, na.rm = T) + 1

bathy <- getNOAA.bathy(lon1=min_x,lon2=max_x, lat1=min_y,lat2=max_y,resolution=4)
```

```
## Querying NOAA database ...
```

```
## This may take seconds to minutes, depending on grid size
```

```
## Building bathy matrix ...
```

```
# If you can't get this function to work (sometimes NOAA's servers get
# overwhelmed), you can
# 'uncomment' (by removing the preceding # sign) and run the following line
# after downloading the file
# from github (https://github.com/ryanreisinger/SOES3056/blob/main/Lab%2005/data/bathy.RDS) and saving

# bathy <- readRDS("./data/bathy.RDS")

# Convert it to a raster-raster, then to a terra raster
bathy <- marmap::as.raster(bathy)
bathy <- terra::rast(bathy)

# Plot to check
ggplot(data = world) +
    geom_sf() +
  # first, the raster
  geom_spatraster(data = bathy) +
  # then, the tracking data
    geom_point(data = ngp_embc_data, aes(x = decimal_longitude,
                                y = decimal_latitude,
                                colour = as.factor(embc_label))) +
  # and then we 'zoom' the map -- note I euse the limits we just calculated
    coord_sf(xlim = c(min_x, max_x),
             ylim = c(min_y, max_y))
```



We can now use a handy feature to 'extract' the values from the bathymetry raster at each tracking location. That is, at each tracking location we use the 'extract' function to look up the depth value there.

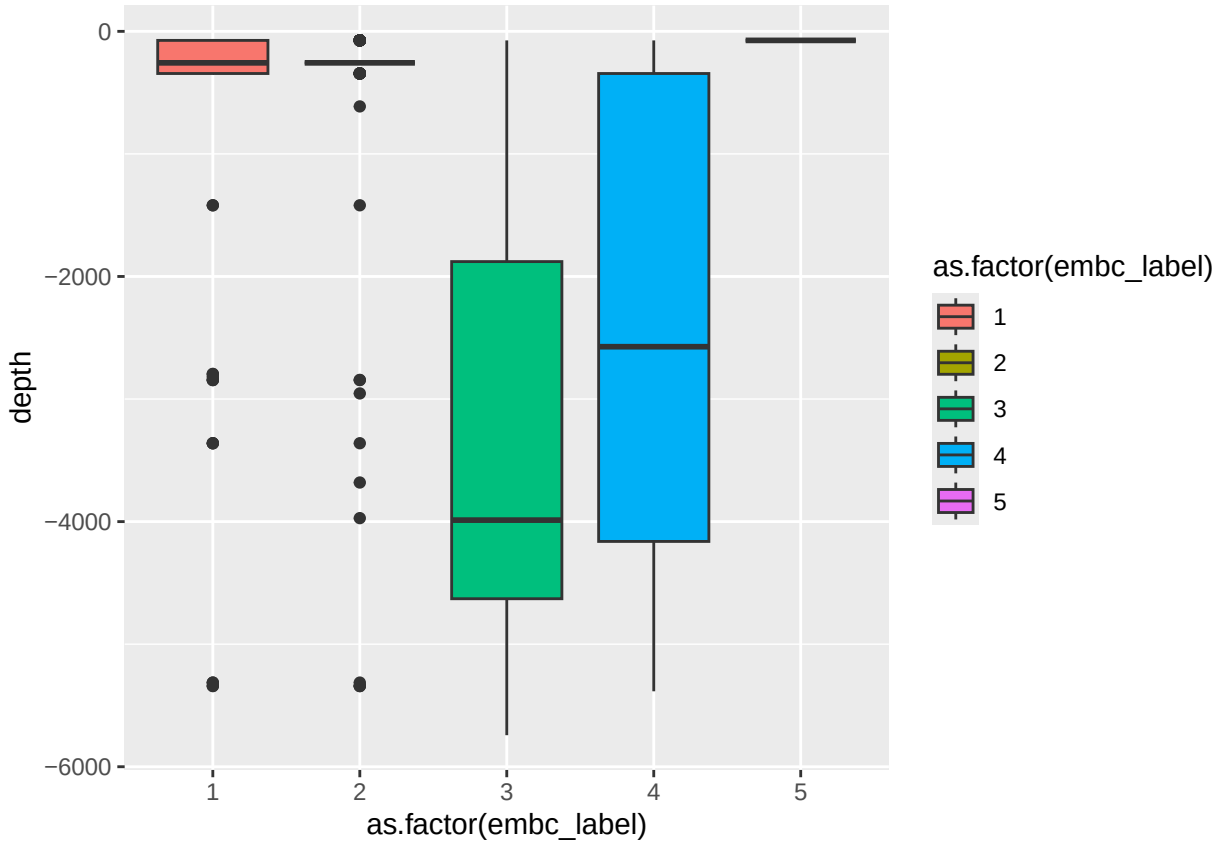
```
ngp_embc_data$depth <- terra::extract(bathy, ngp_embc_data[,c("decimal_longitude", "decimal_latitude")])

# If we look at the first few rows now, we see depth is added as a column
head(ngp_embc_data)
```

```
##             date_time decimal_longitude decimal_latitude embc_velocity
## 1 2015-09-15 17:49:44          37.85326        -46.95477      0.000000
## 2 2015-09-15 17:49:45          37.85326        -46.95477      0.000000
## 3 2015-09-15 17:49:46          37.85326        -46.95477      0.000000
## 4 2015-09-15 17:49:47          37.85326        -46.95477      2.352481
## 5 2015-09-15 17:49:48          37.85327        -46.95475      1.113195
## 6 2015-09-15 17:49:49          37.85327        -46.95474      1.113195
##   embc_turnrad embc_label    depth
## 1    0.0000000          1 -73.7877
## 2    0.0000000          1 -73.7877
## 3    0.0000000          1 -73.7877
## 4    0.0000000          3 -73.7877
## 5    0.3288924          3 -73.7877
## 6    0.0000000          3 -73.7877
```

We can then look at the depth values corresponding with each label class.

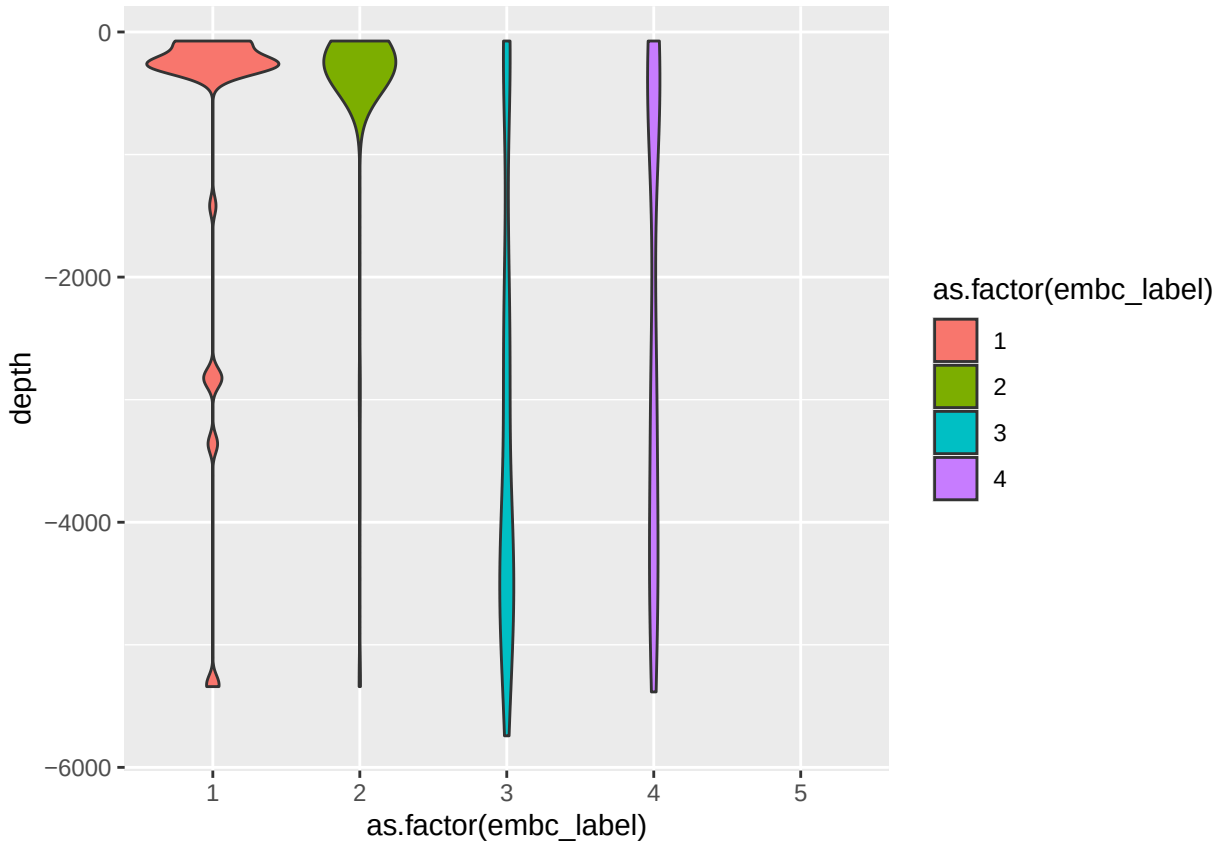
```
# Using a boxplot
ggplot(data = ngp_embc_data, aes(x = as.factor(embc_label),
                                 y = depth,
                                 fill = as.factor(embc_label))) +
```

```
  geom_boxplot()
```




```
# Or a violin plot
ggplot(data = ngp_embc_data, aes(x = as.factor(embc_label),
                                 y = depth,
                                 fill = as.factor(embc_label))) +
  geom_violin()
```

```
## Warning: Groups with fewer than two datapoints have been dropped.
## i Set `drop = FALSE` to consider such groups for position adjustment purposes.
```



So, we see that class 5 indeed corresponds with NC (not classified, there are no values) and classes 1 and 2 seem to be mainly on or near land locations (but not only)–notice the large number of values near 0 for depth. Remember: for this analysis we did not trim out the initial locations when the bird was probably still on its nest, which might explain all those near-zero values for depth.

We see that the boxes for class 3 and 4 overlap a great deal, but there is some difference. Class 4 is associated with shallower water. Let's focus on those two classes and see if there is a significance difference in the depth values according to a t-test. Note, in a thorough analysis you would probably not use a t-test, because these data violate the assumption of independence — they are auto-correlated because they are measured one after the other in a temporal sequence. They are also unlikely to be normally distributed. But that's a discussion for another time!

```
class_3 <- dplyr::filter(ngp_embc_data, embc_label == 3)
class_4 <- dplyr::filter(ngp_embc_data, embc_label == 4)
t.test(class_3$depth, class_4$depth)
```

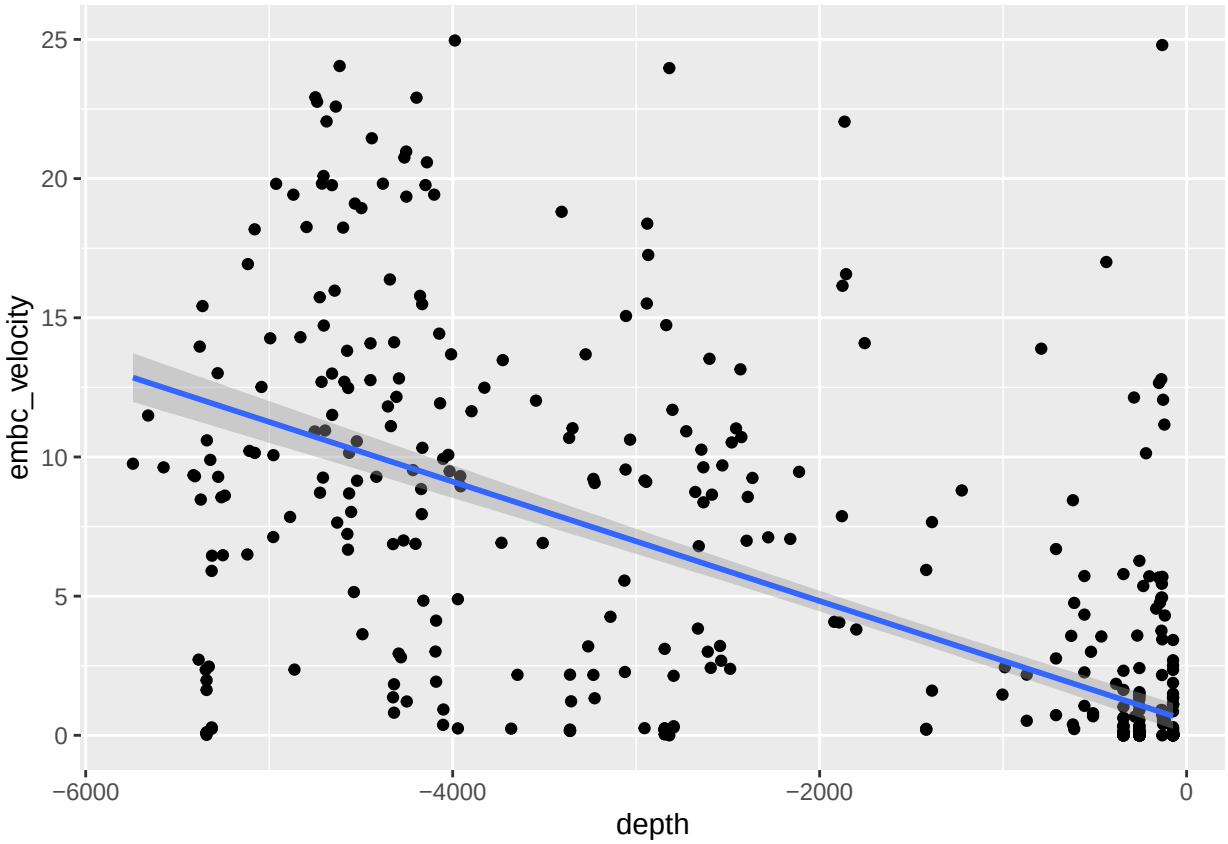
```
##
##  Welch Two Sample t-test
##
## data:  class_3$depth and class_4$depth
## t = -3.7716, df = 220.97, p-value = 0.0002083
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -1324.347  -415.328
## sample estimates:
## mean of x mean of y
## -3176.386 -2306.549
```

There is a significant difference, but as I say, we are violating some assumptions...

Looks look at the case where we have a continuous-value response variable, like the velocity (speed), which we calculated earlier. We can plot the relationship and fit a linear model.

```
ggplot(data = ngp_embc_data, aes(x = depth,
                                 y = embc_velocity)) +
  geom_point() +
  geom_smooth(method = "lm")
```

```
## `geom_smooth()` using formula = 'y ~ x'
```




```
# Correlation
cor(ngp_embc_data$embc_velocity, ngp_embc_data$depth)
```

```
## [1] -0.6721509
```

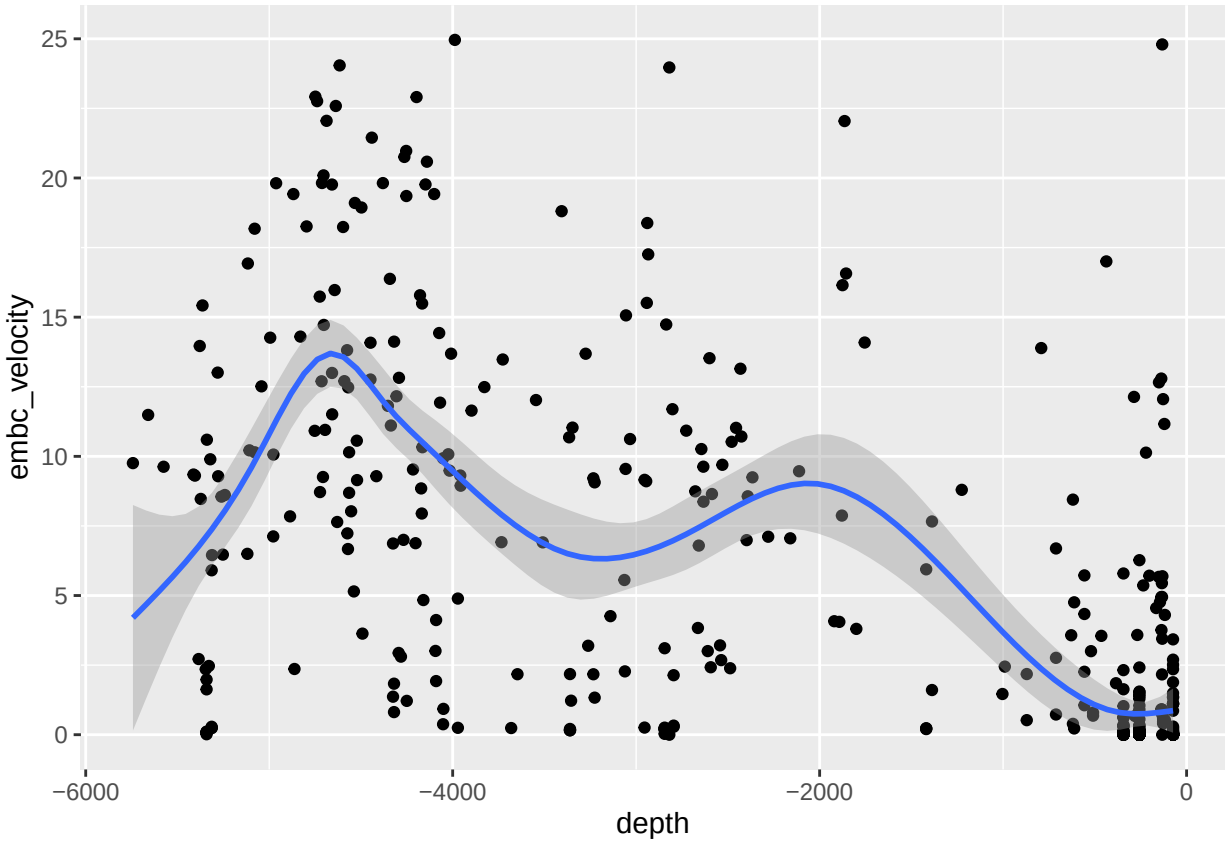
```
# Linear model
lm(ngp_embc_data$embc_velocity ~ ngp_embc_data$depth)
```

```
##
## Call:
## lm(formula = ngp_embc_data$embc_velocity ~ ngp_embc_data$depth)
##
## Coefficients:
##         (Intercept)  ngp_embc_data$depth
##            0.535333            -0.002145
```

We fitted a linear regression, and find a strong correlation. We find that the bird travels faster over deeper water. However, these kinds of relationships are very unlikely to be linear, so we more often fit a 'smooth' of some kind, for example using a generalized additive model (GAM).

```
ggplot(data = ngp_embc_data, aes(x = depth,
                                 y = embc_velocity)) +
  geom_point() +
  geom_smooth(method = "gam")
```

```
## `geom_smooth()` using formula = 'y ~ s(x, bs = "cs")'
```



So, you've conducted your first movement analyses! There's a massive variety of other things you could analyse, and ways to do that, but this is a basic tutorial to get you started.

Sources of movement data

To find more movement data which you can use directly in R, explore the `move` package, which can be used to work with data from the www.movebank.org site:

https://cran.r-project.org/web/packages/move/vignettes/move.html

https://cran.r-project.org/web/packages/move/vignettes/browseMovebank.html