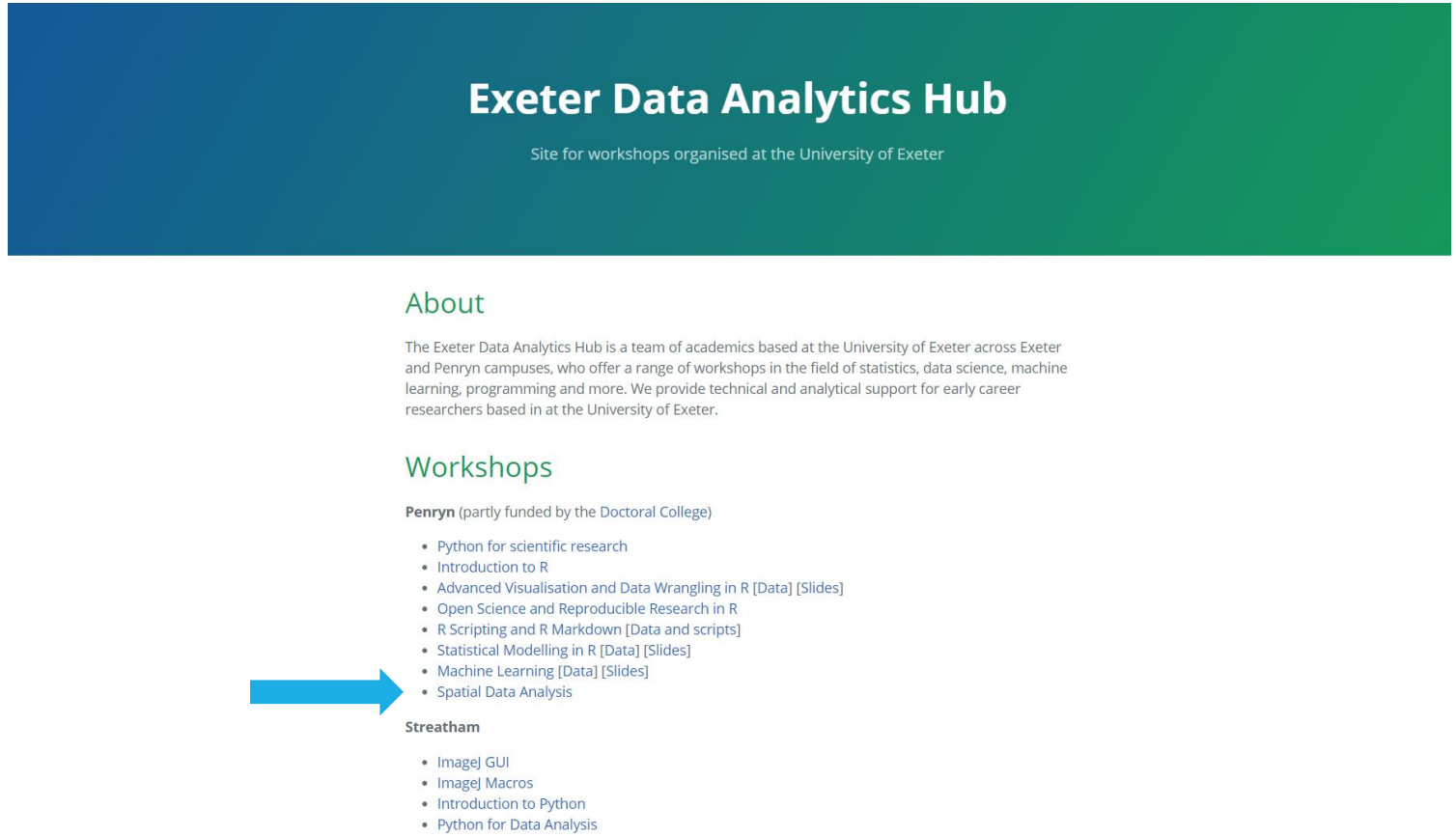
INTRODUCTION TO SPATIAL DATA ANALYSIS
WORKING WITH VECTOR DATA IN R

David March (d.march@exeter.ac.uk)
James Duffy (j.duffy@exeter.ac.uk)
Chris Yeomans (c.m.yeomans@exeter.ac.uk)

Penryn Campus, 20th & 27th November 2019

COURSE MATERIALS

https://exeter-data-analytics.github.io/



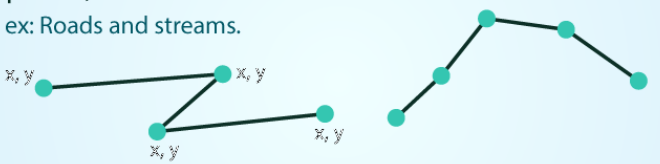
REAL WORLD TO VECTOR MODEL

POINTS: Individual **x, y** locations.

ex: Center point of plot locations, tower locations, sampling locations.

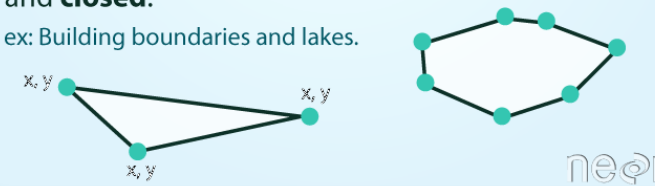
LINES: Composed of many (at least 2) vertices, or points, that are connected.

ex: Roads and streams.



POLYGONS: 3 or more vertices that are connected and **closed**.

ex: Building boundaries and lakes.



CAPTURE VECTOR DATA

- Satellite & cellular networks
- Digitizing map data
- Raster to vector conversion

Credits: NASA, Trimble

LEARNING BY HANDS-ON PRACTICAL SESSIONS

HELLO, WORLD!

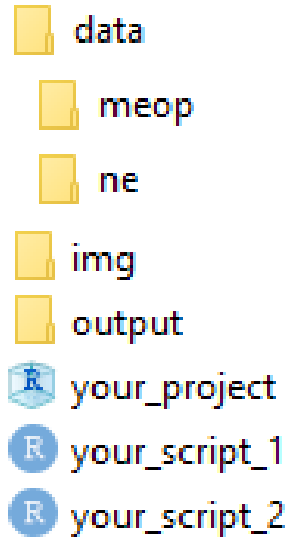
- Import spatial data (point & polygon)
- Change CRS
- Spatial operations: subset, overlap
- Visualize maps

ANIMAL OCEANOGRAPHERS

- Convert data.frame to spatial data
- Export spatial data
- Handle trajectory data
- Advanced visualizations

SET UP YOUR PROJECT

1. Create a new project from Rstudio
2. Download the Data
3. Unzip and store its contents in a folder named data/ within your new project.
4. Create folders output/ and img/
5. Install all required packages

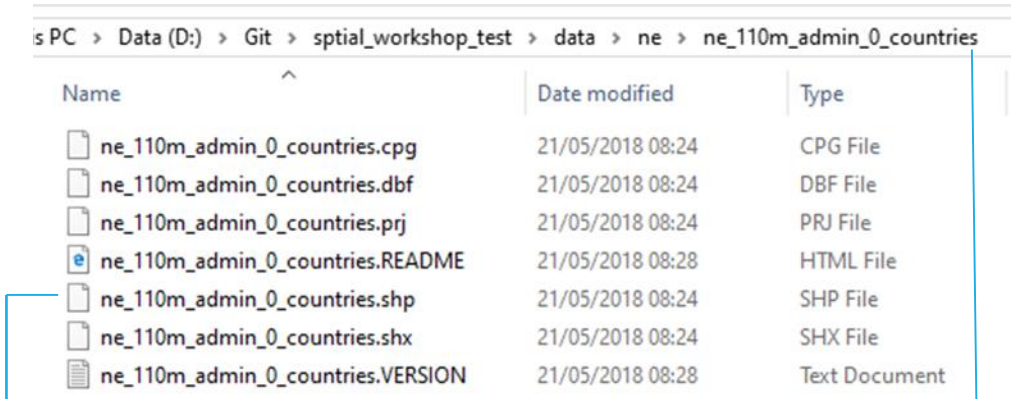


HELLO, WORLD

- Import spatial data (point & polygon)
- Change CRS
- Spatial operations: subset, overlap
- Visualize maps

VECTOR DATA FORMATS

IMPORT SHAPEFILES




```r
# Import countries
countries <- readOGR(dsn = "data/ne/ne_110m_admin_0_countries", layer = "ne_110m_admin_0_countries")
```

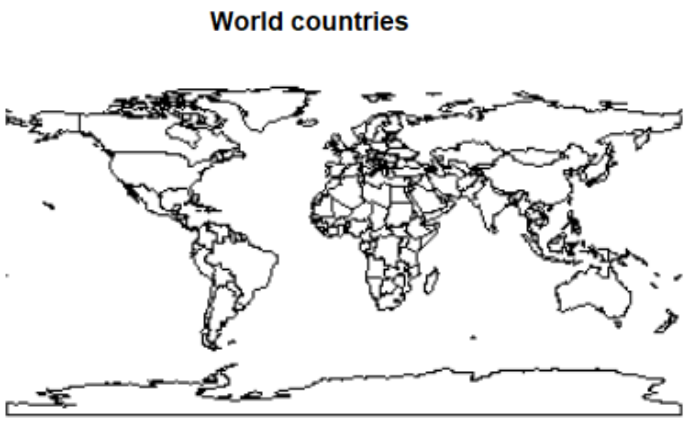
```
## OGR data source with driver: ESRI Shapefile
## Source: "D:\Git\spatial_data_analysis\data\ne\ne_110m_admin_0_countries", layer: "ne_110m_admin_0_countries"
## with 177 features
## It has 94 fields
## Integer64 fields read as strings:  POP_EST NE_ID
```

Num. polygons

Num. attributes

IMPORT SHAPEFILES

```
# Quick plot
plot(countries, main = "World countries")
```



IMPORT SHAPEFILES

```
# View spatial attributes
class(countries)  # sp class
```

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

```
# the attribute table
head(countries@data)
# the coordinates of each point making up each of the polygons
head(countries@polygons)
# the bounding box / extent of the data
countries@bbox
```

IMPORT GEOPACKAGE

```
places <- readOGR("data/ne/ne_110m_populated_places_simple.gpkg")
```

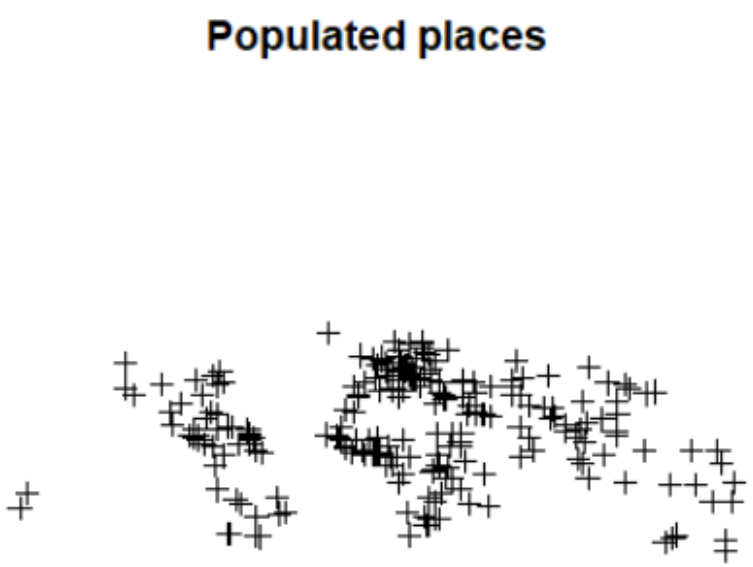
```
## OGR data source with driver: GPKG
## Source: "D:\Git\spatial_data_analysis\data\ne\ne_110m_populated_places_simple.gpkg", layer: "places"
## with 243 features
## It has 38 fields
```

```
# view spatial attributes
class(places)
```

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

IMPORT SHAPEFILES

```
# Quick plot
plot(places, main = "Populated places")
```



IMPORT SHAPEFILES

```
# Plot together with country maps
plot(countries, col = "grey80", border = "grey80")
plot(places, pch = 20, col = "darkblue", add = TRUE)
```

COORDINATE REFERENCE SYSTEMS (CRS)

```
# View spatial attributes
proj4string(countries)  # displays the coordinate reference system (CRS)
```

```
## [1] "+proj=longlat +datum=WGS84 +no_defs +ellps=WGS84 +towgs84=0,0,0"
```

```
# Change to Mollweide projection
countries_moll <- spTransform(countries, CRS("+proj=moll +ellps=WGS84"))

# Quick plot
plot(countries_moll, col = "grey80", main = "World countries")
```

SPATIAL SUBSETTING

- Select data of a region of interest (polygon)

```
# Import raster package
library(raster)

# Set min and maximum coordinates (lon/lat)
xmin <- -15
xmax <- 46
ymin <- 28
ymax <- 60

# Create an extent object
e <- extent(xmin, xmax, ymin, ymax)
class(e)
```

```
## [1] "Extent"
## attr(,"package")
## [1] "raster"
```

raster package not only works with raster data, but also provides useful functions to process vector data as we will see in this example

SPATIAL SUBSETTING

- Select data of a region of interest (polygon)

```
# Geographic subset of countries by the extent defined
countries_subset <- crop(countries, e)
plot(countries_subset)
```

SPATIAL SUBSETTING

- Select data of a region of interest (polygon)

Task

- Make a plot of South America
- Tip: you can use QGIS or Google Earth to search for coordinates to use in your extent object.

ATTRIBUTE SUBSETTING

- Select data of an attribute (or combination of attributes) of interest

```
df <- data.frame(countries@data)
head(df)
```

List of continents:

```
unique(countries$CONTINENT)
```

```
## [1] Oceania                Africa                 North America
## [4] Asia                   South America          Europe
## [7] Seven seas (open ocean) Antarctica
## 8 Levels: Africa Antarctica Asia Europe North America ... South America
```

ATTRIBUTE SUBSETTING

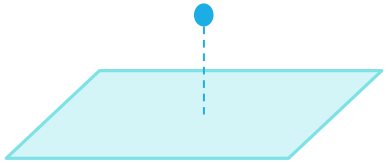
- Select data of an attribute (or combination of attributes) of interest

```
# Subset South America
south_america <- countries[countries$CONTINENT == "South America",]
plot(south_america, col = "lightgreen", border = "darkgreen", lwd=3, main = "South America")
```

```
extent(south_america)  # You can also get your extent
```

```
## class       : Extent
## xmin        : -81.41094
## xmax        : -34.72999
## ymin        : -55.61183
## ymax        : 12.4373
```

SPATIAL OVERLAP



- Point-in-polygon Overlay
  - Select points that occur within the spatial extent of the polygon
  - Add attribute information from the polygon

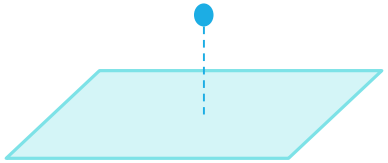
```
# Spatial overlay
ov <- over(places, countries)
class(ov)
```

```
## [1] "data.frame"
```

```
places$CONTINENT <- ov$CONTINENT
head(places)
```

Points that do not occur within the spatial extent will return *NA* in attributes from the polygon

SPATIAL OVERLAP

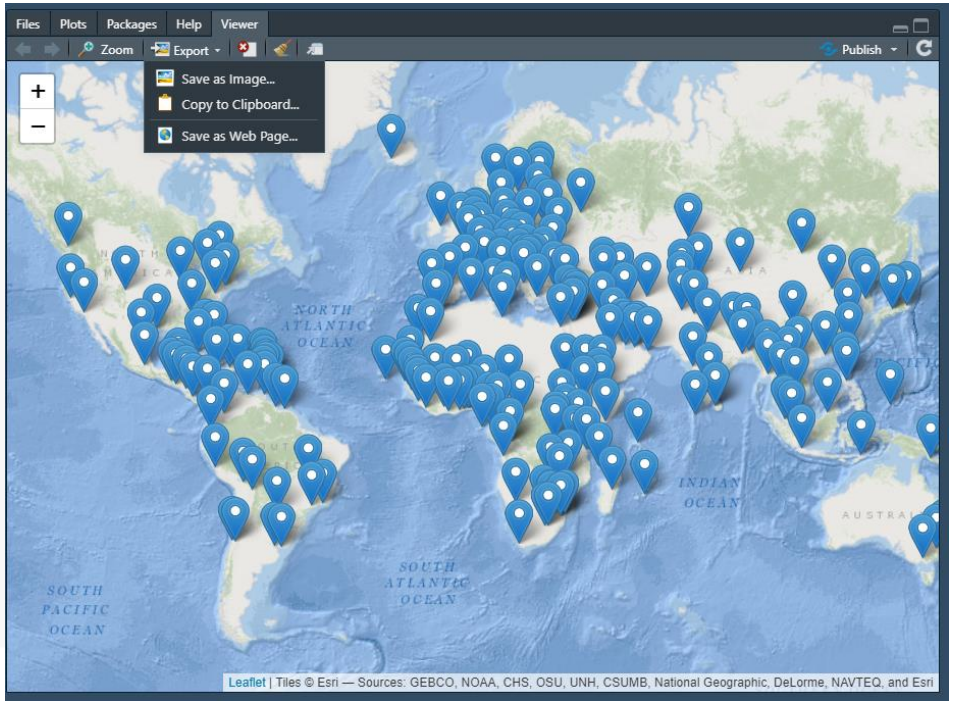


- Combine Point-in-polygon Overlay with Attribute Subsetting

```r
# Subset places and countries from Africa
places_africa <- places[which(places$CONTINENT == "Africa"),]

# Plot together with country maps
plot(countries, col = "grey80", border = "grey80")
plot(places_africa, pch = 20, col = "darkblue", add = TRUE)
```

INTERACTIVE MAPS

```r
# import leaflet package
library(leaflet)

# create leaftet map
map <- leaflet(data = places) %>%
  addProviderTiles("Esri.OceanBasemap") %>%  # Base map
  addMarkers(popup = ~name)

# plot leaflet map
map
```



ANIMAL OCEANOGRAPHERS

- Convert data.frame to spatial data
- Export spatial data
- Handle trajectory data
- Advanced visualizations

ANIMAL OCEANOGRAPHERS

Credit: Miquel Gomila

IMPORT DATA FROM CSV

```r
# Import data
# Subset of Mirounga leonina for 2012
mirounga <- read.csv("data/meop/miorunga.csv")
class(mirounga)
```

```
## [1] "data.frame"
```

```r
head(mirounga)
```

```
##          species    ptt         tag                date      lon      lat
## 1 Southern ellie 113368 ct77-167-12 2012-03-17 10:40:00 110.5548 -66.4558
## 2 Southern ellie 113368 ct77-167-12 2012-03-17 14:20:00 110.3725 -66.3939
## 3 Southern ellie 113368 ct77-167-12 2012-03-17 23:20:00 110.4566 -66.3640
## 4 Southern ellie 113368 ct77-167-12 2012-03-18 01:10:00 110.4468 -66.3704
## 5 Southern ellie 113368 ct77-167-12 2012-03-18 10:30:00 110.4070 -66.3601
## 6 Southern ellie 113368 ct77-167-12 2012-03-18 14:40:00 110.4506 -66.3885
##    scientific_name
## 1 Mirounga leonina
## 2 Mirounga leonina
## 3 Mirounga leonina
## 4 Mirounga leonina
## 5 Mirounga leonina
## 6 Mirounga leonina
```

We will work with a dataset of oceanographic sensors that were equipped on Southern Elephant Seals. This will bring us to the Antarctica!

Source data: MEOP-CTD database https://doi.org/10.17882/45461

TRANSFORM DATA.FRAME INTO SP CLASS

```r
# load library
library(rgdal)

# Convert to spatial class
coordinates(mirounga) <- ~lon+lat

# View spatial attributes
proj4string(mirounga)  # displays the coordinate reference system (CRS)
```

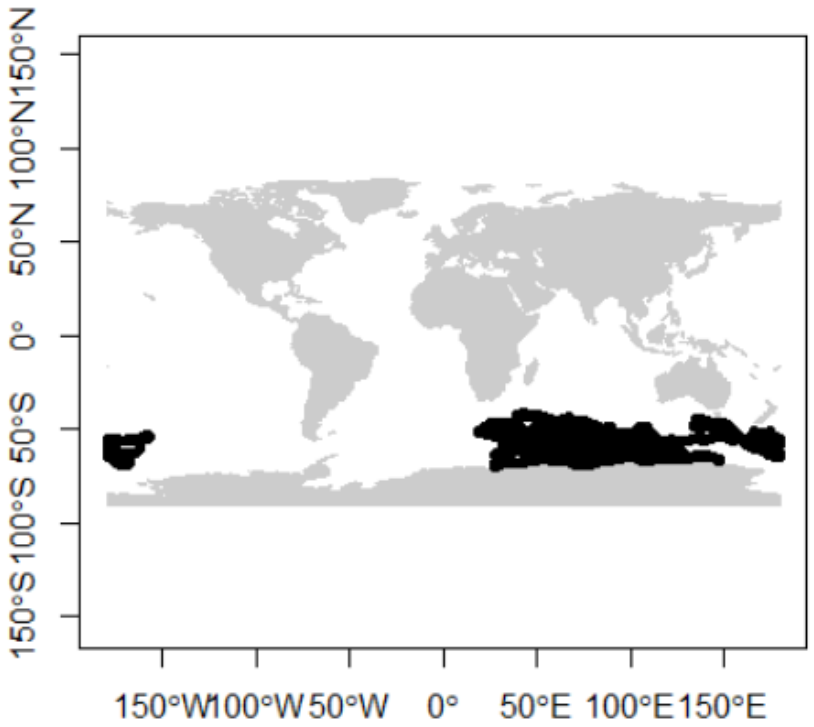
```
## [1] NA
```

```r
# Use EPSG code
proj4string(mirounga) <- CRS("+init=epsg:4326")

# Alternative using proj4 string
# proj4string(mirounga) <- CRS("+proj=longlat +datum=WGS84 +no_defs +ellps=WGS84 +towgs84=0,0,0")
```

Check spatialreference.org to find more about EPSG code

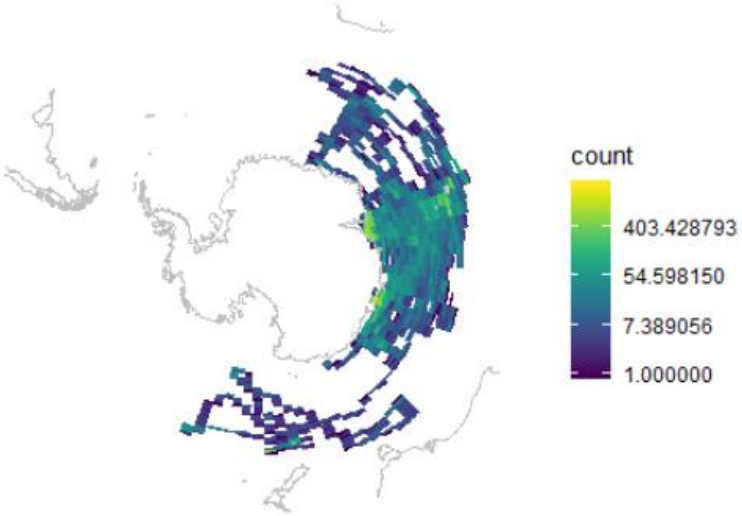
TRANSFORM DATA.FRAME INTO SP CLASS

```
plot(countries, col = "grey80", border = "grey80", axes = TRUE)
plot(mirounga, pch = 20, add = TRUE)
```



BINNED MAPS WITH GGPLOT2

```r
# load library
library(ggplot2)
library(viridis)

# Import a new world dataset
# country layer have some issues using orthographic projection
world <- map_data("world")  # turn data from the maps package in to a data frame suitable for plotting with ggplot2

# Also turn back the CTD dataset into a data.frame
mirounga_df <- data.frame(mirounga)

# Plot
ggplot() +
  geom_bin2d(data = mirounga_df, aes(x = lon, y = lat), bins = 100) +
  geom_path(data = world, aes(x = long, y = lat, group = group), colour = "#c0c0c0") +
  ylim(-90, -30) +
  xlab("") +
  ylab("") +
  coord_map("ortho", orientation = c(-90, 0, 0)) + # orthographic projection from South Pole
  scale_fill_viridis(option = "viridis", trans = "log") + # log scale for bin count
  theme(panel.background = element_rect("white"), # dark background
        axis.ticks = element_blank(),
        panel.grid = element_blank(), # remove panel grid
        axis.text = element_blank()) # remove x-axis value
```

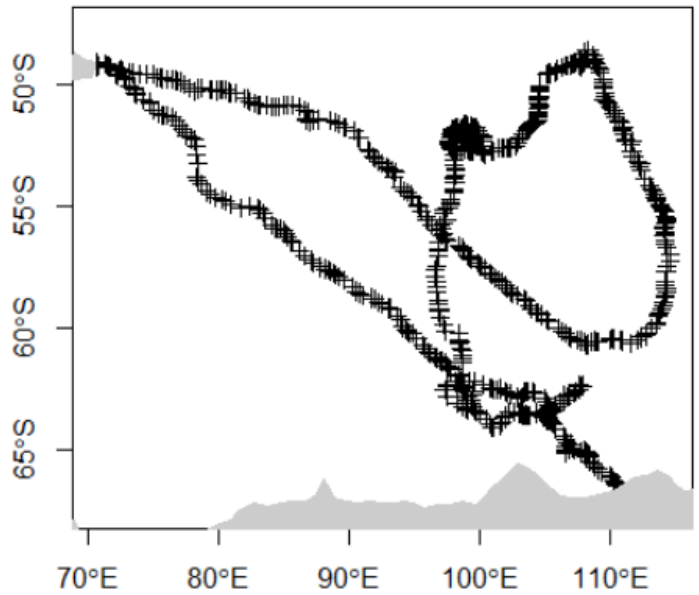


MOVEMENT DATA

x, y, t

SUBSET TRAJECTORY DATA

```r
# Select one tag
track <- mirounga[mirounga$tag == "ct77-167-12",]
plot(track, axes = TRUE)
plot(countries, col = "grey80", border = "grey80", add=TRUE)
```



CONVEX HULL OR MINIMUM CONVEX POLYGON

- Creates the smallest polygon possible that can contain a given set of points.

```
# Calculate convex hull polygon
library(adehabitatHR)
hull <- mcp(track, percent = 100)
plot(hull, col = "lightblue")
plot(track,add=TRUE)
plot(countries, col = "grey80", border = "grey80", add=TRUE)
```

EXPORT SPATIAL DATA

```r
# export to shapefile
writeOGR(track, dsn = "output", layer = "track_points", driver = "ESRI Shapefile", overwrite_layer = TRUE)
writeOGR(hull, dsn = "output", layer = "track_hull", driver = "ESRI Shapefile", overwrite_layer = TRUE)

# export to geopackage
writeOGR(track, dsn = "output/track_points.gpkg", layer = "track_points", driver = "GPKG", overwrite_layer = TRUE)
writeOGR(hull, dsn = "output/track_hull.gpkg", layer = "track_hull", driver = "GPKG", overwrite_layer = TRUE)

# export to KML
writeOGR(track, dsn = "output/track_points.kml", layer = "track", driver = "KML", overwrite_layer = TRUE)
writeOGR(hull, dsn = "output/track_hull.kml", layer = "hull", driver = "KML", overwrite_layer = TRUE)
```

HANDLING MOVEMENT TRAJECTORY DATA

```r
# Import libraries
library(lubridate)  # parse times
library(move)  # move object class and tools for trajectories
library(moveVis)  # animation plots
```

```r
# Back transform SpatialPointsDataFrame into a data.frame
track <- as.data.frame(track)

# parse date time
track$date <- parse_date_time(track$date, "Ymd HMS")

# convert to move object
mdata <- move(x=track$lon, y=track$lat, time=track$date,
                  data=track, proj=CRS("+proj=longlat +ellps=WGS84"),
                  animal=track$tag)
```

LINEAR INTERPOLATION OF TRAJECTORIES

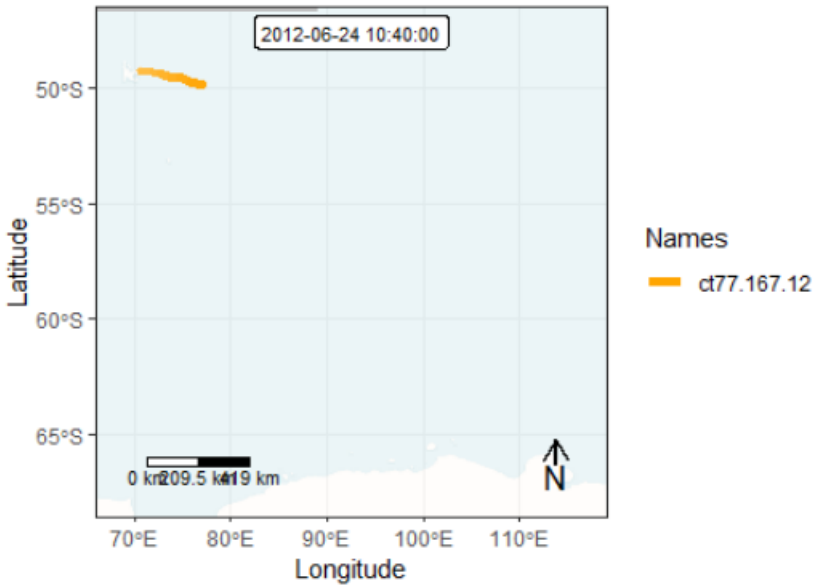
- Interpolate trajectories at regular time intervals

```r
# align move_data to a uniform time scale
# For animation, unique frame times are needed
m <- align_move(mdata, res = 1, digit = 0, unit = "days", spaceMethod = "greatcircle")
```

ANIMATIONS OF MOVEMENT DATA

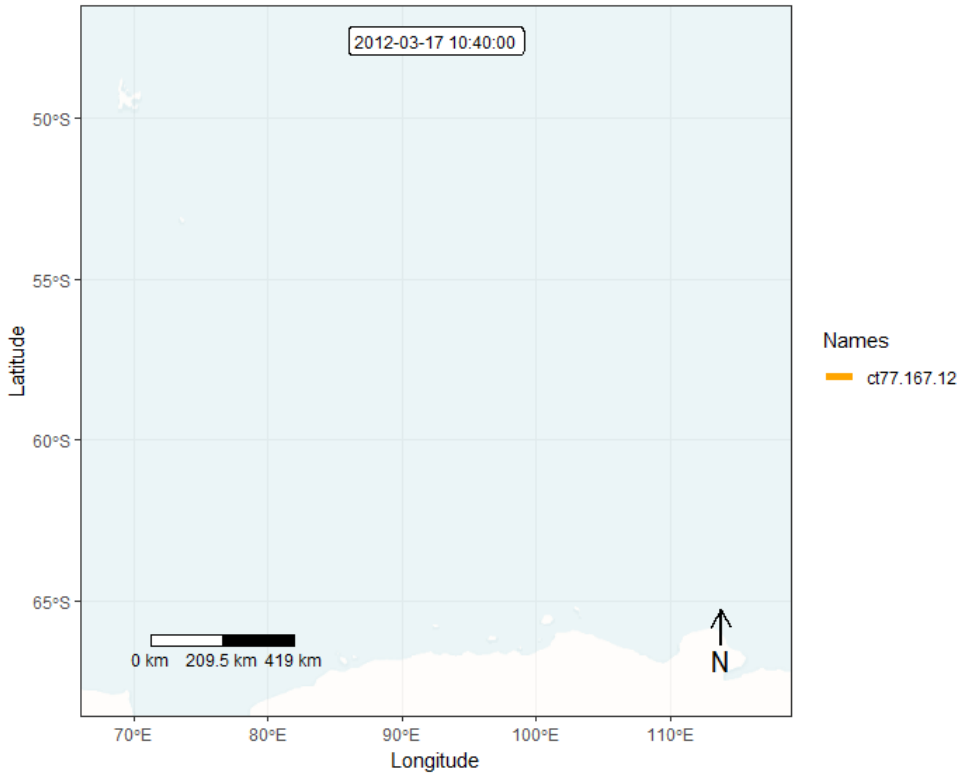
```
# create spatial frames with a OpenStreetMap watercolour map
frames <- frames_spatial(m,  # move object
                         map_service = "carto", map_type = "voyager_no_labels",  # base map
                         path_size = 2, path_colours = c("orange"), alpha = 0.5) %>%  # path
          add_labels(x = "Longitude", y = "Latitude") %>% # add some customizations
          add_northarrow(colour = "black", position = "bottomright") %>%
          add_scalebar(colour = "black", position = "bottomleft") %>%
          add_timestamps(m, type = "label") %>%
          add_progress(size = 2)
```

```
frames[[100]] # preview one of the frames, e.g. the 100th frame
```



ANIMATIONS OF MOVEMENT DATA

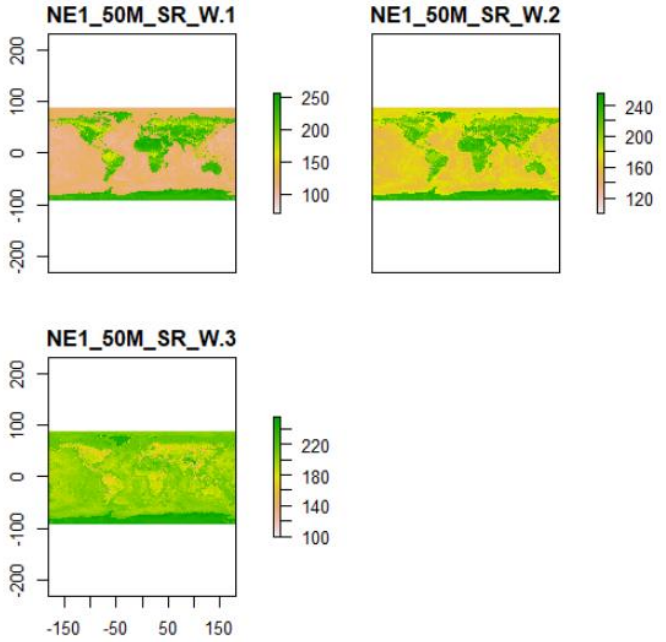
```r
# animate frames
animate_frames(frames, out_file = "img/animation.gif", overwrite = TRUE, display = FALSE)
```



ANIMATIONS OF MOVEMENT DATA (EXTRA)

- Customize the basemap with raster data

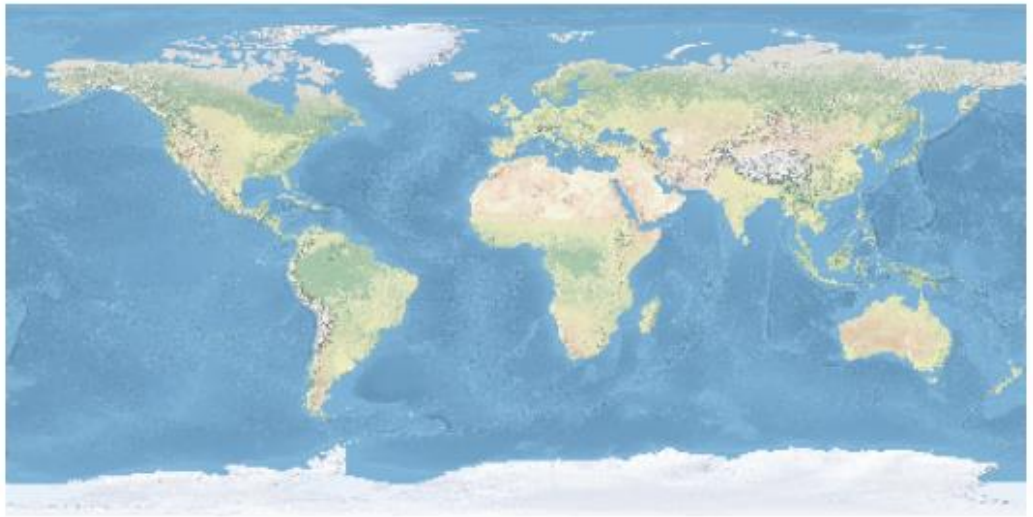
```
# import base map
bmap <- brick("data/ne/NE1_50M_SR_W/NE1_50M_SR_W.tif")

# plot base map raster
plot(bmap)
```



ANIMATIONS OF MOVEMENT DATA (EXTRA)

- Customize the basemap with raster data

```
# compose the 3 RGB bands of Tif file
plotRGB(bmap)
```



ANIMATIONS OF MOVEMENT DATA (EXTRA)

- Subset and convert data to polar stereographic projection

```
# transform trajectory to the polar stereographic projection (EPSG:3031)
mpol <- spTransform(m, crs("+init=epsg:3031"))
```

```
# crop basedmap to extent of track (plus 2 degrees)
bmap <- crop(bmap, extent(-180, 180, -90, -40))

# reproject raster to polar stereographic projection (EPSG:3031)
bmap_pol <- projectRaster(bmap, crs = crs("+init=epsg:3031"))

# plot data
plotRGB(bmap_pol, addfun = lines(mpol))
```

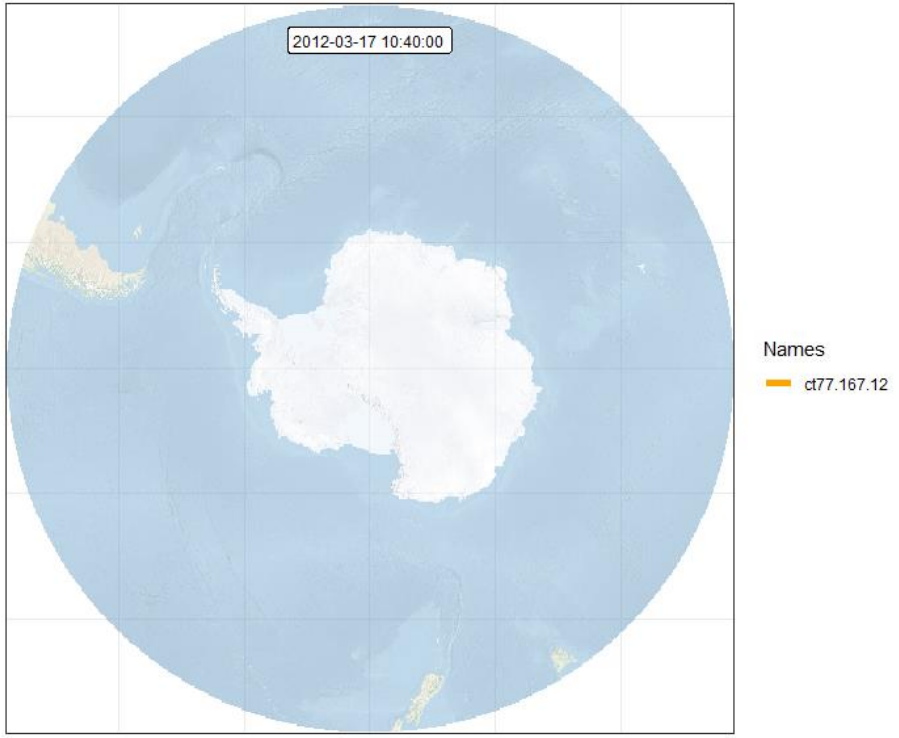
ANIMATIONS OF MOVEMENT DATA (EXTRA)

- Animate

```
# create spatial frames with a custom basemap
frames <- frames_spatial(mpol,
          r_list = bmap_pol, r_times = m$time[1], ext = extent(bmap_pol),  #custom base map
          path_size = 2, path_colours = c("orange"), alpha = 0.5) %>%
          add_timestamps(mpol, type = "label") %>%
          add_progress(size = 2)
```

```
# remove axis
frames <- add_gg(frames, gg = expr(theme(axis.ticks = element_blank(),
                                         axis.title = element_blank(),
                                         axis.text = element_blank())))
```

```
# animate frames
animate_frames(frames, out_file = "img/animation_polar.gif", overwrite = TRUE)
```



ACKNOWLEDGEMENTS

- JJ Valletta & TJ McKinley - for their time and efforts in setting up the data analytics program
- DM acknowledges funding from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No 794938.