GEO880: Patterns and Trends in Environmental Data

Patrick Laube and Nils Ratnaweera

Contents

Chapter 1

Introduction Chapter

For our practical `R` course building-up skills for analyzing movement data in the software environment `R`, you'll be using data from the ZHAW project "Prävention von Wildschweinschäden in der Landwirtschaft".

The project investigates the spatiotemporal movement patterns of wild boar (*Sus scrofa*) in agricultural landscapes. For more information you can ask Beni Sigrist, one of the course participants involved in this project. We will study the trajectories of these wild boar, practicing the most basic analysis tasks of Computational Movement Analysis (CMA).

Please note we are given this data under the condition of signing a non-disclosure agreement. Why? This is data coming out of an ongoing research project. Capturing wild living animals and then equipping them with GPS collars is a very labor and cost intensive form of research. Consequently, data resulting such campaigns is a very valuable asset that must be protected.

Introduce some basic concepts

- RStudio
- Tidyverse
- datacamp
- Rstudio Server / Rstudio Connect
- Links to GIS Ressources
- Animove Cheet sheets
- GIT In R
- GIT / CMAtools

Chapter 2

Lesson 1

Lesson 1 (L1) covers the necessary steps for getting ready in `R` and some basic concepts for setting up a well-structured `R` project. The lesson introduces how additional packages that provide useful functions are made available and how temporal data is handled. The lesson concludes with the creation of your first map featuring movement data.

2.1 Leaning outcomes

- You learn how to structure an `R` project.
- You can read movement data from a .csv-file into a `data.frame` and into a `sf` object.
- You can produce simple maps of your trajectory data using `ggplot2`, `leaflet`, `plotly` and `tmap`

2.2 Prerequisites

Readings Skills from "R for Data Science" (Wickham and Grolemund 2017):

- RS1.1 Preface (16p, ix-xxiv)
- RS1.2 Chap2 Workflow basics (3p, 37-39)
- RS1.3 Chap4 Workflow scripts (3p, 77-79)
- RS1.4 Chap6 workflow projects (6p, 111-116)
- RS1.5 Chap8 Data Import with `readr` (21p)
- RS1.6 Chap13 Date and Times with `lubridate` (18p, 237-256)

2.3 Tasks

2.3.1 Task 1: Prepare project

Create a new *RStudio Project*. As recommended in Wickham and Grolemund (2017), remove the option "*Restore .RData into workspace at startup*" and "*save workspace to .RData on exit*" to "*Never*".

Create a new R-File and divide it into the sections neccessary in a classical Data Science Workflow. "Sections" can be created within RStudio by adding Comments (`#`) with at least 4 Trailing dashes (`-`), equal signs (`=`), or pound signs (`#`) (see below). Sections allow code folding (try click on the small triangle next to the line number) and provides and navigation (try the shortcut: `Shift+Alt+J`).

In the first section (loading environment / libraries), add the code to install and load the package `tidyverse`. Once you've installed the package, you can uncomment the corresponding line of code, because you will not need to execute this line in your next R Session.

2.3.2 Task 2: Import data

In section "data import", import the file `wildschwein.csv`. For everyone working on the RStudio Server, this data is saved in a Folder named "Geodata" one level *above* your home folder. You will have to move "up" one or two levels from your project folder using the syntax and then into the correct folder using the syntax `"../../Geodata/"`(for two levels) or `"../Geodata/"` (for one level).

Note:

- If your are using a graphical tool to import your code, make sure you save the corresponding code in your R Script. This is important in regard to the reproducibility of your script and will ensure that your worflow is documented without gaps.
- I recommend using one of the `tidyverse` functions (`read_*`) to import your data. These functions are less error prone than the base `R` functions (`read.*`). Specifically to the wildboar data, I recomment `read_delim`.
- if you recieve warnings during import, have a look at these warnings by using the funciton `problems()`. Resolve these problems until import runs without warnings.
- Assign correct data types as neccessary and make sure the timezone is set correctly for the date/time columnm.

2.3.3 Task 3 Explore Data

We will use a variaty of different plotting techniques in this course, several have emerged in recent years, each with their specific strenghts and weaknesses. While `base::plot()`is quick and simple, it not very scaleable with growing complexity. ggplto offers solution for most use cases and has a elegant syntax that is easy to get accustomed to. `plotly()` offers great interactive and linked view facilites while `tmap()` was designed specifically for spatial data.

Get an overview of your data by creating a first "map-like" plot of your data producing a simple scatter plot with ggplot2. Assign every individual animal its own colour (using the ggplot2 argument colour). Do you spot outliers? If so, get rid of the outliers. Plot your data again, this time without outliers. Save your plot using `ggsave()`. Save your code in the appropirate section.

Setting up a `ggplot` with our data is done using the command `ggplot(roe_gps_all, aes(X, Y, colour = TierID))`. Creating a map is done via the basic scatter plot command `geom_point()`, using a fixed aspect ratio of 1.

2.3.4 Task 4: Handling spatial data

Till now, we've handled spatial data within dataframes. This works well for many tasks, but sometimes we need special *spatial* classes to handle our trajectories. Projecting the WGS84 (Lat/Long) coordinates into CH1903_LV95 is such a case.

Some of you might know the `sp` packge with the classes `SpatialPoints` and `SpatialPointsDataFrame`. Just recently the new and exiting package `sf`, was released on CRAN. `sf` has some huge advantages over sp:

- simple featrues are essentially dataframes, which mean they interface with the `tidyverse` (and the `dplyr` SAC paradigm)
- are OGC (ISO 19125-1:2004) compliant and interface with GDAL, PostGIS, GeoJSON and so fourth
- are being rapidly implemented in visualisation tools such as `ggplot2`, `gplotly` and `tmap`

A lot of reasons to learn `sf` and work with this library. The down side is however, that due to its youth not all packages have implemented `sf`. We have created a small package (`CMAtools`) to help you with such cases, so we will not have to switch back and fourth between classes during this course.

Use the function `st_as_sf()` while correctly specifying your Lat/Long Coordinates. Set the coordinate reference system using the EPSG Code as an integer value. You can set the argument `agr` to `constant`. Save to output of this operation to a new variable `wildschwein_BE_sf`.

Take a look at this sf object (`head()`, `str()`,`View()`) and try a few classical dataframe operations on it (subseting, filtering). Now transform the coordinates into CH1903_LV95 using the function st_transform(). Again, use the EPSG code as an integer.

2.3.5 Task 5

Now that we have a spatial object of our point data, we can do spatial operations on them. We can for example calculate the minimum

Extract the new Coorindates using `st_coordinates()` and attach them (`cbind()`) to your original `dataframe`.

Note that `st_transform()` names the coordinates `X` and `Y`, but CH1903 LV03 names the Axes `E` and `N`. Rename the axes accordingly (before or after attaching them to your dataframe).

Keep your Long / Lat coordinates, since it is helpful to have both WGS84 and CH1903+ Coordinates stored in the dataframe:

- WGS84: this is our original data therefore we should not discard this information. Additionally, some visualization tools (eg. `Leaflet` and `ggmap`) need Lat/Long Coordinates
- CH1903+ these cartesian coordinates are helpful when calculating euclidean distances between positions (this is much more complicated with WGS84 Data) and if we use swisstopo background- and other context data

2.4 Solutions (RCode)

```
## Task 1 ####################


# Loading enironment / libraries ####

# install.packages("tidyverse")
library(tidyverse)
library(sf)



## Task 2 ####################


# Data import ####
wildschwein_BE <- read_delim("../Geodata/wildschwein_BE.csv",",")

## Task 3 ####################
```

```
ggplot(wildschwein_BE, aes(Lat,Long, colour = TierID)) +
  geom_point() +
  coord_fixed(1) +
  theme(legend.position = "none")

wildschwein_BE <- filter(wildschwein_BE, Lat < 50)

## Task 4 ####################

wildschwein_BE_sf = st_as_sf(wildschwein_BE, coords = c("Long", "Lat"), crs = 4326, agr = "constant", re


wildschwein_BE_sf <- st_transform(wildschwein_BE_sf, 2056)


mcp <- wildschwein_BE_sf %>%
  group_by(TierID) %>%
  summarise() %>%
  st_convex_hull()

ggplot(mcp,aes(fill = TierID)) +
  geom_sf(alpha = 0.4) +
  coord_sf(datum = 2056) +
  theme(
    legend.position = "none",
    panel.grid.major = element_line(colour = "transparent"),
    panel.background = element_rect(fill = "transparent")
    )



coordinates <- st_coordinates(wildschwein_BE_sf)

colnames(coordinates) <- c("E","N")

wildschwein_BE_sf <- cbind(wildschwein_BE_sf,coordinates)

## NA
```

Chapter 3

Lesson 2

3.1 Learning Outcomes

3.2 Prerequisites

Readings Skills from "R for Data Science" (Wickham and Grolemund 2017):

- RS2.1 Chap3 Data Transformation with `dplyr` (31p, 43-76)
- RS2.2 Chap10 Relational data with `dplyr` (21p, 171-193)
- RS2.3 Chap14 Pipes with `magrittr` (6p, 261-268)

Readings Theory - R2.1 Laube and Purves (2011): How fast is a cow? cross - scale analysis of movement data.

3.3 Preperation

Open your R Project from last week. Load all libraries and run the scrip to import and clean your data. Install and load the following additional libraries.

3.4 Tasks

3.4.1 Task 1

Depending on your knowledge of `R`, getting an overview of the data we imported last week might have been quite a challange. Quite surprisingly, importing, cleaning and exploring your data can be the most challanging, time consuming part of a project. RStudio and the tidyverse offer many extremely helpful tools to make this part easier, and more fun.

To get an overview of the data, use the `dplyr` tools `group_by`, `summarise.` Try to answer the following questions:

- How many individuals were tracked?
- How long were the individual tracked? Are there gaps?
- Were all individuals tracked cuncurrently or sequentially?
- What is the temporal sampling interval between the locations?

3.4.2 Task 2

Now that we've established that we have different sampling intervals, we have to segement our trajectories in such a way, that we can perform further analysis during specific sampling intervals only. If we measure speed, or turning angles, we have to be very clear on what temporal (an thus spatial) scale we are performing this analysis.

We therefore have to define threshold to group segments with a similar sampling interval. Explore the dataset in more detail (e.g. using histograms at different scales), and choose reasonable threshold values to group the trajectories into different sampling intervals. Use the function `cut()` to apply the thresholds on the column `timelag`.

Note:

- It might make more sense to choose narrow group intervals at smaller timelags and wider groups intervals at higher timelags.
- The function `cut` splits a vector into segements according to the values specified in breaks = . The default lables can be a bit puzzling at first, but `(` and `]` are a standard form of notating intervals in mathematics.
- Store the groupnames in a new column named `samplingInt`

3.4.3 Task 3

Now that we've gotten the nifty job of specifying our intervals out of the way, let's get to a more fun part and calculate the speed of the animals movements.

- If you're working with dplyr, you can add `samplingInt` to `group_by()` (in addition to TierID) and so make sure your're not calculating speed across different sampling intervals.
- You can use the function `euclid()` from the `CMAtools` package to calculate Euclidean distances between subsequent rows. Use ?euclid to see what the function expects and returns.
- use `lead(E,1)` to address the the row `n+1`
- make sure you're clear in what unit you are measuring speed. Meters per second is a SI base unit, but might be unhandy for the speeds travelled by wild boar.

3.4.4 Task 4

Measuring speed between subsequent samples is great, but especially for short sampling intervals they can be missleading due to measurment error. It might be desireble to *smoothen* these erros using a moving window function. The `zoo` package offeres a variaty of moving window functions (`roll*`). Use `roll_mean()` to smoothen the calculated speed. Familiarise yourself with this function by working on some dummy data, for example:

```
example <- rnorm(10)
rollmean(example,k = 3,fill = NA,align = "left")
rollmean(example,k = 4,fill = NA,align = "left")
```

Visualize the output from your moving windows and compare different window sizes (`k =`).

3.4.5 Task 5

You've read Laube and Purves (2011) about segementing trajectories. In the paper, they define "*static*" fixes as "*those whose average Euclidean distance to other fixes inside a temporal window* v* is less than some threshold *d**". This sounds more complicated than it is, the figure illustrates the method nicely.

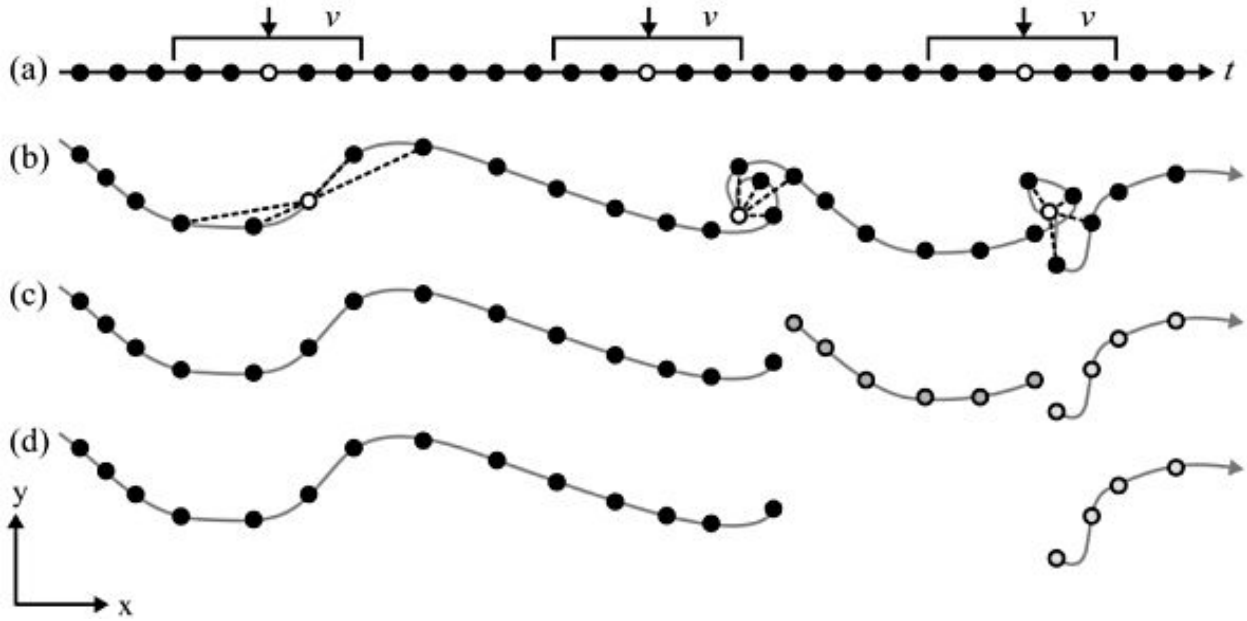
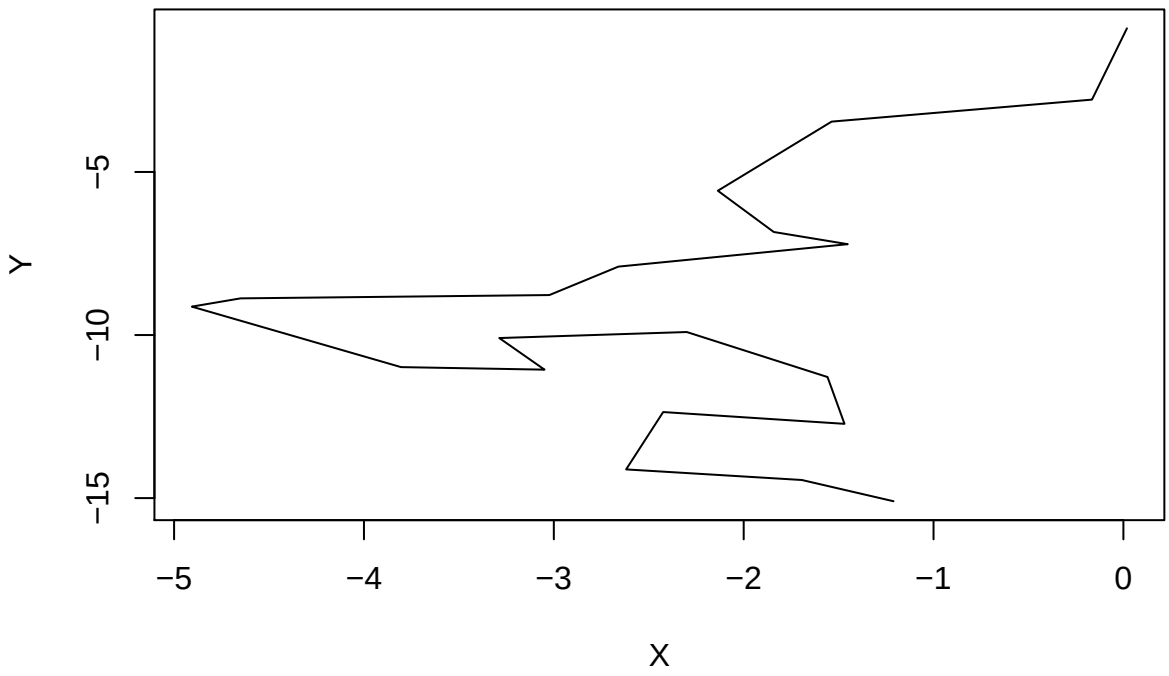


Figure 3.1:

Try to implement this method on some dummy data. Once you've solved the problem on this simple data, we will implement it on the wild boar data next week.

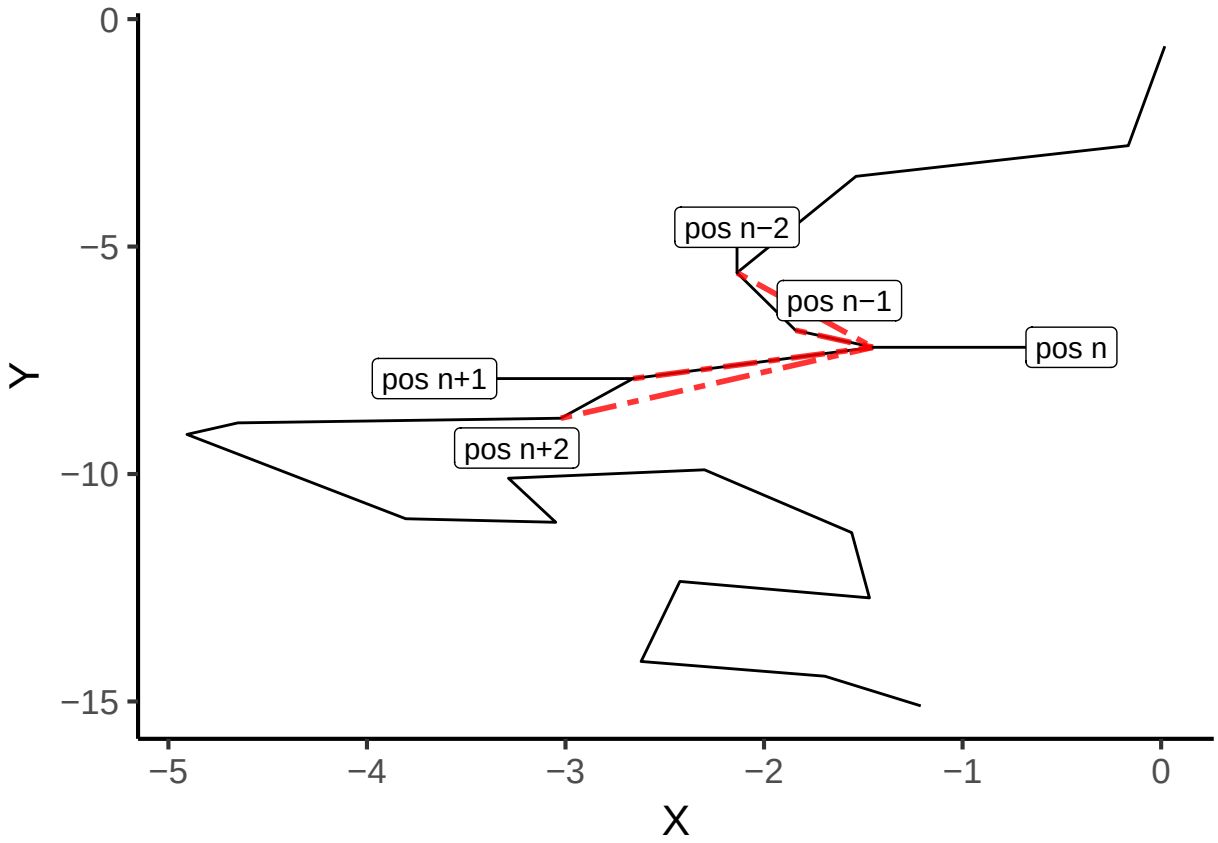
```
set.seed(10)
X = cumsum(rnorm(20))
Y = cumsum(rnorm(20))

plot(X,Y, type = "l")
```



Assume the sampling interval is 5 minutes. If we take a temporal window of 20 minutes, that would mean we include 5 fixes into the calculation. We need to calculate the following Eucledian distances (pos representing a `X`,`Y`-position):

1. pos[n-2] to pos[n]
2. pos[n-1] to pos[n]
3. pos[n] to pos[n+1]
4. pos[n] to pos[n+2]

You can use the function `euclid()` to calculate the distances and `dplyr` functions `lead()`/``lag()` to create the necessary offsets. On our dummy data, we want to calculate the mean of the following distances (red, dotted lines).



3.5 Solutions (RCode)

```
## install.packages("devtools")
## install.packages("zoo")
##
## devtools::install_git("https://github.engineering.zhaw.ch/PatternsTrendsEnvironmentalData/CMAtools.g
library(CMAtools)
library(zoo)
## Task 1 ####################

ggplot(wildschwein_BE_sf, aes(DatetimeUTC,TierID)) +
  geom_line()

wildschwein_BE_sf <- wildschwein_BE_sf %>%
  group_by(TierID) %>%
  mutate(
    timelag = as.numeric(difftime(lead(DatetimeUTC),DatetimeUTC,units = "mins"))
  )


ggplot(wildschwein_BE_sf, aes(timelag)) +
  geom_histogram(binwidth = 50)
```

```
ggplot(wildschwein_BE_sf, aes(timelag)) +
  geom_histogram(binwidth = 1) +
  lims(x = c(0,100)) +
  scale_y_log10()

wildschwein_BE_sf[1:50,] %>%
  ggplot(aes(DatetimeUTC,timelag)) +
  geom_line() +
  geom_point()


## Task 2 ####################

ggplot(wildschwein_BE_sf, aes(timelag)) +
  geom_histogram(binwidth = 0.1) +
  scale_x_continuous(breaks = seq(0,400,20),limits = c(0,400)) +
  # scale_x_continuous(breaks = seq(0,50,1),limits = c(0,50)) +
  scale_y_log10()

wildschwein_BE_sf <- wildschwein_BE_sf %>%
  group_by(TierID) %>%
  mutate(
    samplingInt = cut(timelag,breaks = c(0,5,seq(10,195,15)))
  )

wildschwein_BE_sf %>%
  group_by(samplingInt) %>%
  summarise(
    n = n()
  ) %>%
  ggplot(aes(samplingInt,n)) +
  geom_bar(stat = "identity") +
  theme(axis.text.x = element_text(angle = 45, hjust = 1)) +
  scale_y_log10()


## Task 3 ####################

wildschwein_BE_sf <- wildschwein_BE_sf %>%
  group_by(TierID,samplingInt) %>%
  mutate(
    steplength = euclid(lead(E),lead(N),E,N),
    speed = steplength/timelag
  )



example <- rnorm(10)
rollmean(example,k = 3,fill = NA,align = "left")
rollmean(example,k = 4,fill = NA,align = "left")

## Task 4 ####################
```

```
wildschwein_BE_sf <- wildschwein_BE_sf %>%
  group_by(TierID) %>%
  mutate(
    speed2 = rollmean(speed,3,NA,align = "left"),
    speed3 = rollmean(speed,5,NA,align = "left"),
    speed4 = rollmean(speed,10,NA,align = "left")
  )

wildschwein_BE_sf[1:30,] %>%
  gather(key,val,c(speed,speed2,speed3,speed4)) %>%
  ggplot(aes(DatetimeUTC,val,colour = key,group = key)) +
  geom_point() +
  geom_line()

## Task 5 ####################

nMinus2 <- euclid(lag(X, 2),lag(Y, 2),X,Y)  # distance to pos. -10 minutes
nMinus1 <- euclid(lag(X, 1),lag(Y, 1),X,Y)  # distance to pos.  -5 minutes
nPlus1  <- euclid(X,Y,lead(X, 1),lead(Y, 1)) # distance to pos   +5 mintues
nPlus2  <- euclid(X,Y,lead(X, 2),lead(Y, 2)) # distance to pos  +10 minutes

# Use cbind to bind all rows to a matrix
distances <- cbind(nMinus2,nMinus1,nPlus1,nPlus2)
distances

# This just gives us the overall mean
mean(distances, na.rm = T)

# We therefore need the function `rowMeans()`
rowmeans <- rowMeans(distances)
cbind(distances,rowmeans)

# and if we put it all together:
rowMeans(
  cbind(
    euclid(lag(X, 2),lag(Y, 2),X,Y),
    euclid(lag(X, 1),lag(Y, 1),X,Y),
    euclid(X,Y,lead(X, 1),lead(Y, 1)),
    euclid(X,Y,lead(X, 2),lead(Y, 2))
  )
)
## NA
```

Chapter 4

Lesson 3

4.1 Learning Outcomes

- You are able to segment a trajectory using Laube and Purves (2011)
- You are able to compute the similarity between given trajectories using the package `SimilarityMeasures`.
- You acquire further useful data processing skills.

4.2 Prerequisites

Readings Skills from "R for Data Science" (Wickham and Grolemund 2017):

- RS3.1 Chap1 Data visualization with ggplot2 (31, 3-35)
- RS3.2 Chap5 Exploratory Data Analysis (28p, 81.109)

Readings Theory Alan Both (2018) A Comparative Analysis of Trajectory Similarity Measures: Recommendations for Selection and Use, in review with PLOS ONE, confidential.

4.3 Preperation

4.4 Tasks

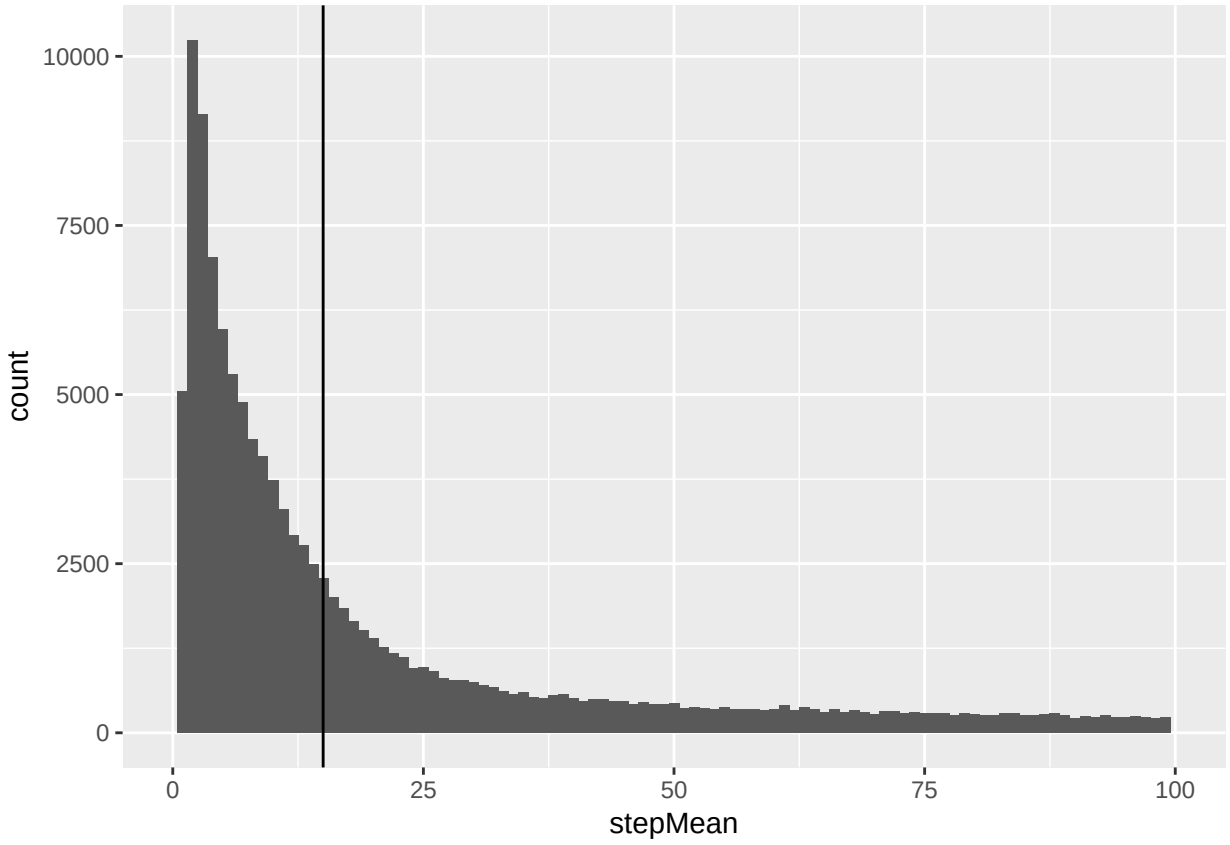
4.4.1 Task 1

Picking up at Task 4 from last week: we can now implement this algorithm using the `dplyr` / `mutate()` method. This might seem a little challenging at first, but if you have completed Task 4 or at least looked at the sample solution, it is quite easy. You can pass anything to a new column within `mutate()` as long as it is a vector of the same length as the original dataframe.

4.4.2 Task 2

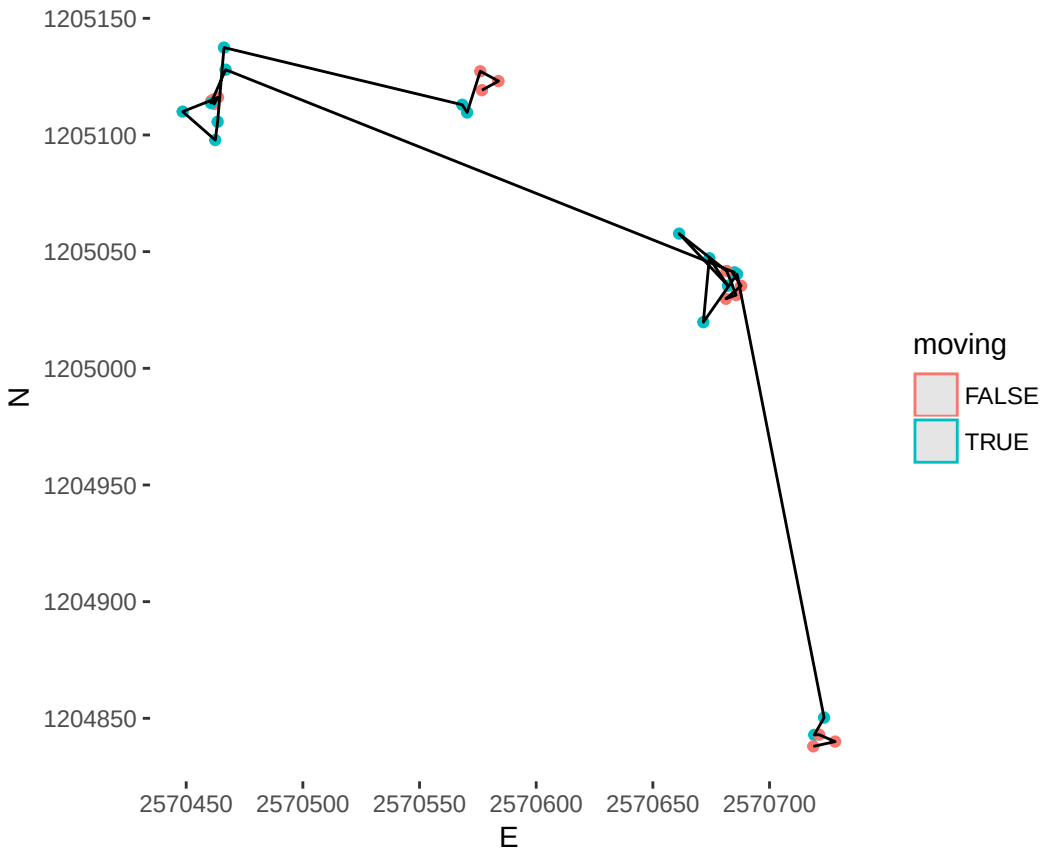
We can now explore the newly created Values `stepMean` using summary statistics (histogramms, boxplot, `summary()`) and define a reasonable threshold value to differentiate between "stops" and "moves". There is no "correct" way of doing this, specifying a threshold always depends on the question that needs to be answered.

```
##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.     NA's
##    0.000    5.109   14.211   57.382   62.224 1811.620       40
```



Based on this data, I will go with a threshold value of 15. Apply your threshold on you data (condition: `stepMean > 15`).

Plot a *subset* of data to verify if the arbitrary threshold was a reasonable choice. - subsetting the first ten rows can be done with `[1:10,]` (dont forget the comma!) - we have a couple of NA values in the beginning and end of each trajectory. Use `filter(!is.na(moving))` to remove these values from the plot - if you add the layer `geom_sf() ggplot` will only plot points. To add the lines in between you'll have to specify the x and y values (E and N) - add the layer `coord_sf(datum = 2056)` - wrap the whole plot function in `ggplotly()` if you want to enable zooming / panning



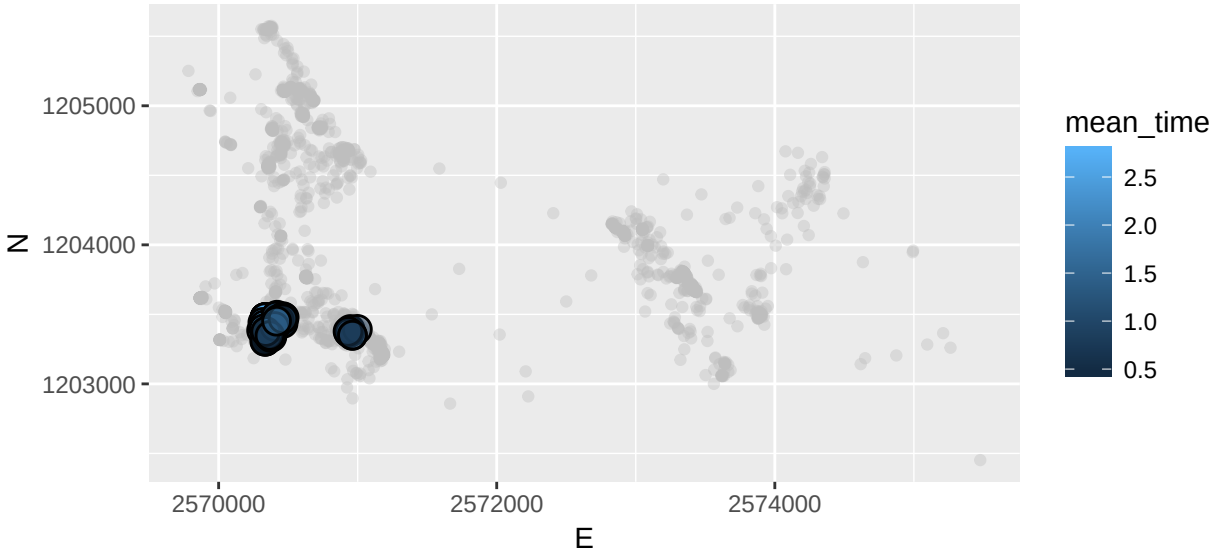
4.4.3 Task 3 (Optional)

To be honest, exploring segmentation with `ggplot` (and even `plotly`) is not very satisfying. Another great tool to visualize spatial data is leaflet. It's not all that hard to create a leaflet plot, just take a look at the documentation and try it with our segmented wild boar data.

```
## [1] "map only available in the online version of this document"
```

4.4.4 Task 4

Use the package `recurse` (function `getRecursions`) to determin sites which are often visited.



4.5 Solution (R Code)

```
library(tidyverse)
library(plotly)
library(CMAtools)
library(recurse)

## Task 1 ####################

wildschwein_BE_sf <- wildschwein_BE_sf %>%
  group_by(TierID) %>%
  mutate(
    stepMean = rowMeans(
      cbind(
        euclid(lag(E, 2),lag(N, 2),E,N),
        euclid(lag(E, 1),lag(N, 1),E,N),
        euclid(E,N,lead(E, 1),lead(N, 1)),
        euclid(E,N,lead(E, 2),lead(N, 2))
        )
      )
  )
## Task 2 ####################
```

```r
summary(wildschwein_BE_sf$stepMean)

ggplot(wildschwein_BE_sf, aes(stepMean)) +
  geom_histogram(binwidth = 1) +
  lims(x = c(0,100)) +
  geom_vline(xintercept = 15)


wildschwein_BE_sf <- wildschwein_BE_sf %>%
  mutate(
    moving = stepMean > 15
  )

wildschwein_BE_sf[20:50,] %>%
  filter(!is.na(moving)) %>%
  ggplot() +
  geom_sf(aes(colour = moving)) +
  geom_path(aes(E,N)) +
  coord_sf(datum = 2056) +
  theme(
    panel.grid.major = element_line(colour = "transparent"),
    panel.background = element_rect(fill = "transparent")
    )

## library(leaflet)
## library(scales)
## factpal <- colorFactor(hue_pal()(2), wildschwein_BE_sf$moving)
##
## # checking to see if this all makes sense in leaflet: (or better ggplot?)
## wildschwein_BE_sf[0:200,] %>%
##   filter(!is.na(moving)) %>%
##   leaflet() %>%
##   addCircles(radius = 1,lng = ~Long, lat = ~Lat, color = ~factpal(moving)) %>%
##   addPolylines(opacity = 0.1,lng = ~Long, lat = ~Lat) %>%
##   addTiles() %>%
##   addLegend(pal = factpal, values = ~moving, title = "Animal moving?")

library(recurse)
library(ggforce)

recurs <- wildschwein_BE_sf %>%
  filter(TierID == "001A") %>%
  select(E,N,DatetimeUTC,TierID) %>%
  st_set_geometry(NULL) %>%
  as.data.frame() %>%
  getRecursions(100)

recurStats <- recurs$revisitStats

recurStats <- recurStats %>%
  group_by(coordIdx) %>%
  summarise(
    number_of_visits = max(visitIdx),
```

```
    x = unique(x),
    y = unique(y),
    total_time = sum(timeInside),
    max_time = max(timeInside),
    mean_time = mean(timeInside)
  )

data1 = filter(recurStats, number_of_visits > 30)
wildschwein_BE_sf %>%
  ungroup() %>%
  filter(TierID == "001A") %>%
  ggplot(aes(E,N)) +
  geom_point(alpha = 0.4, colour = "grey") +
  geom_circle(data = data1, alpha = 0.5, aes(x0 = x,y0 = y,fill = mean_time,r = 100),inherit.aes = F) +
  coord_fixed(1)

## NA
```

Chapter 5

Lesson 4

5.1 Learning Outcomes

5.2 Prerequisites

5.3 Preperation

5.4 Tasks

5.4.1 Task 1

Find spatial overlap

5.4.2 Task 2

Find temporal overlap

5.4.3 Task 3

Combine spatial and temporal overlap and generate datframe with "pairs"

5.4.4 Task 4

Temprally join data and visualize.

5.5 Solutions (RCode)

```
## Task 1 ####################
```

```
mcp <- wildschwein_BE_sf %>%
  group_by(TierID) %>%
  summarise() %>%
  st_convex_hull()


overlap_spatial_mat <- overlap_spatial(mcp,"TierID")

## Task 2 ####################
library(lubridate)

wildschwein_intervals <- wildschwein_BE_sf %>%
  group_by(TierID) %>%
  summarise(
    min = min(DatetimeUTC,na.rm = T),
    max = max(DatetimeUTC,na.rm = T)
  ) %>%
  ungroup() %>%
  mutate(
    interval = interval(min,max)
  )

overlap_temporal_mat <- overlap_temporal(wildschwein_intervals,"interval","TierID")



## Task 3 ####################

overlap_spat_temp <- overlap_temporal_mat & overlap_spatial_mat

overlap_spat_temp <- overlap_spat_temp %>%
  as.data.frame() %>%
  rownames_to_column("id1") %>%
  gather(id2,bool,-id1) %>%
  filter(bool == TRUE) %>%
  select(-bool)
## Task 4 ####################


# temporal_join(wildschwein_BE_sf,TierID,overlap_spat_temp$id1[1],overlap_spat_temp$id2[1],DatetimeUTC,
#
#
# ggplot(temporal_join) +
#   geom_point(aes(E2,N2), colour = "blue") +
#   geom_point(aes(E1,N1), colour = "red")
#
#
# leaflet(temporal_join) %>%
#   addCircles(lng = ~Long1,lat = ~Lat1) %>%
#   addTiles()
```

Chapter 6

Individual Project

6.1 Possible topics

- In Lesson 3 (Task 1) we implemented Laube and Purves (2011) concept by using a fixed number of "offsets" `lead()` and `lag()`. However, this provides an issues with varying sampling intervals. Can you think of a way to implement the algorithm by specifying a temporal window size, rather than an offset?
- Implement recurse package from Chloe Bracis
- Implement Meet pattern from CMAtools
- Implement Segmentation (ReMuS) from Sean C. Ahearn and Somayeh Dodge

Alan Both, Matt Duckham, Kevin Buchin. 2018. "A Comparative Analysis of Trajectory Similarity Measures: Recommendations for Selection and Use."

Laube, Patrick, and Ross S. Purves. 2011. "How Fast Is a Cow? Cross - Scale Analysis of Movement Data." *Transactions in GIS* 15 (3): 401–18. doi:10.1111/j.1467-9671.2011.01256.x.

Wickham, Hadley, and Garrett Grolemund. 2017. *R for Data Science: Import, Tidy, Transform, Visualize, and Model Data*. 1st ed. O'Reilly Media, Inc.