In-Class Exercise 8: MovementVis with R

Dr. Kam Tin Seong
Assoc. Professor of Information Systems

School of Computing and Information Systems,
Singapore Management University

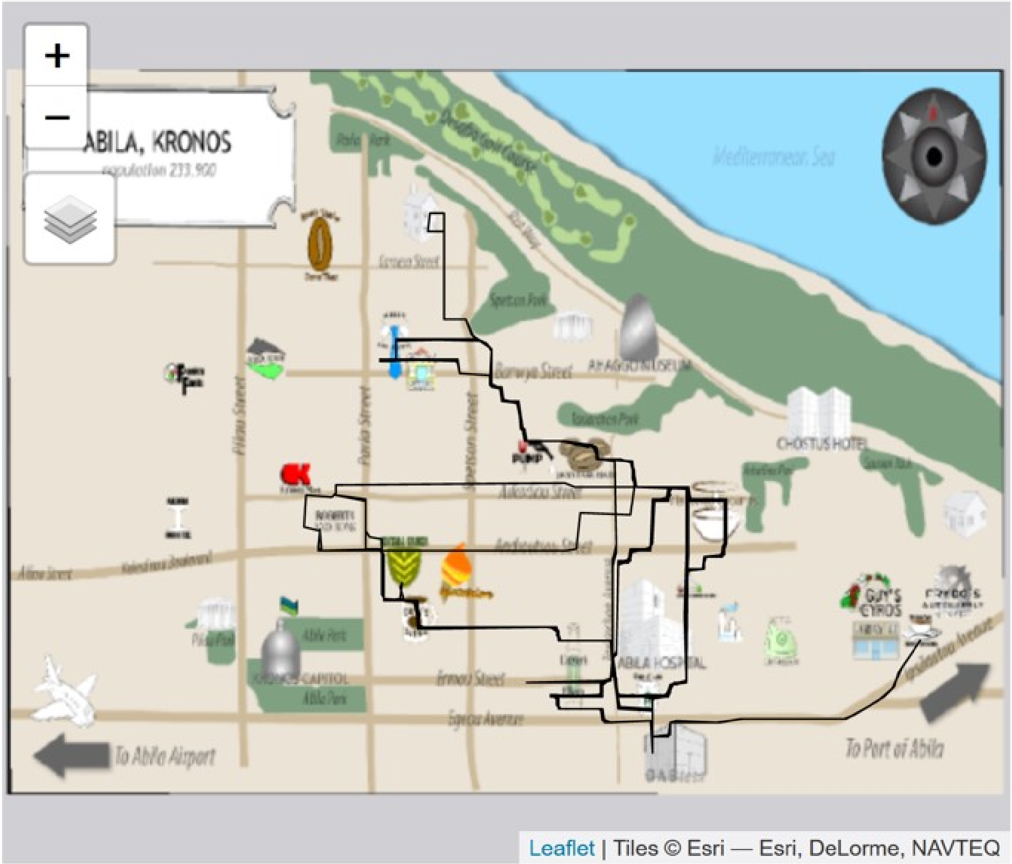
2020-2-15 (updated: 2021-08-12)

From Movement Data to MovementVis

Movement Data

	Timestamp	id	lat	long
1	01/06/2014 06:28:01	35	36.07623	24.87469
2	01/06/2014 06:28:01	35	36.07622	24.87460
3	01/06/2014 06:28:03	35	36.07621	24.87444
4	01/06/2014 06:28:05	35	36.07622	24.87425
5	01/06/2014 06:28:06	35	36.07621	24.87417
6	01/06/2014 06:28:07	35	36.07619	24.87406
7	01/06/2014 06:28:09	35	36.07619	24.87391
8	01/06/2014 06:28:10	35	36.07618	24.87381
9	01/06/2014 06:28:11	35	36.07617	24.87374
10	01/06/2014 06:28:12	35	36.07618	24.87362
11	01/06/2014 06:28:13	35	36.07616	24.87354
12	01/06/2014 06:28:14	35	36.07617	24.87347

MovementVis



Overview

- In this hands-on exercise, you will learn how to visualise movement data by using appropriate R packages.

- By the end of this hands-on exercise, you will be able:

    - to import GIS data file such as shapefile into R by using **sf** package,
    - to import a georeferenced image such as geotif file into R by using **raster** package,
    - to import an aspatial data into R by using **readr** pakage,
    - to convert the apstial data into simple point feature by using **sf** package,
    - to wrangling date-time field by using **clock** package,
    - to derive movement path (also known as trajectory) from the movement points by using **sf** package, and finally
    - to visualising the movement paths by using **tmap** package.

Getting Started

Write a code chunk to check, install and launch **raster**, **sf**, **clock**, **tmap** and **tidyverse** packages of R

The solution:

```
packages = c('raster', 'sf',
             'tmap', 'clock',
             'tidyverse')
for (p in packages){
  if(!require(p, character.only = T)){
    install.packages(p)
  }
  library(p,character.only = T)
}
```

Importing Raster file

Write a code chunk to import *MC2-tourist.tif* into R by uing *raster()* of **Raster** package.

The solution:

```
bgmap <- raster("data/geospatial/MC2-tourist.t
bgmap
```

```
## class       : RasterLayer
## band        : 1  (of  3  bands)
## dimensions  : 1595, 2706, 4316070  (nrow, ncol, nce
## resolution  : 3.16216e-05, 3.16216e-05  (x, y)
## extent      : 24.82419, 24.90976, 36.04499, 36.0954
## crs         : +proj=longlat +datum=WGS84 +no_defs
## source      : MC2-tourist.tif
## names       : MC2.tourist
## values      : 0, 255  (min, max)
```

Plotting Raster Layer

In general, tm_raster() will be used to plot a raster layer by using tmap package.

```
tmap_mode("plot")
tm_shape(bgmap) +
    tm_raster(bgmap,
              legend.show = FALSE)
```

However, *bgmap* layer is a three bands false colour image. Hence, *tm_rgb()* is used instead.

```
tm_shape(bgmap) +
tm_rgb(bgmap, r = 1,g = 2,b = 3,
       alpha = NA,
       saturation = 1,
       interpolate = TRUE,
       max.value = 255)
```

Importing Vector GIS Data File

Abila GIS data layer is in ESRI shapefile format. It is in vector data model and the feature class is line.

Using st_read() of sf package, import *Abila* shapefile into R.

The solution:

```
Abila_st <- st_read(dsn = "data/Geospatial",
                    layer = "Abila")
```

```
## Reading layer `Abila' from data source
##   `D:\tskam\ISSS608\In-class_Ex\In-class_Ex08\data
##   using driver `ESRI Shapefile'
## Simple feature collection with 3290 features and 9
## Geometry type: LINESTRING
## Dimension:     XY
## Bounding box:  xmin: 24.82401 ymin: 36.04502 xmax:
## Geodetic CRS:  WGS 84
```

Importing Aspatial Data

Using *read_csv()* of **readr** package, import *gps.csv* into R.

The solution:

```
gps <- read_csv("data/aspatial/gps.csv")
glimpse(gps)
```

```
## Rows: 685,169
## Columns: 4
## $ Timestamp <chr> "01/06/2014 06:28:01", "01/06/20
## $ id        <dbl> 35, 35, 35, 35, 35, 35, 35, 35,
## $ lat       <dbl> 36.07623, 36.07622, 36.07621, 36
## $ long      <dbl> 24.87469, 24.87460, 24.87444, 24
```

Be warned:

- *Timestamp* field is not in date-time format.
- *id* field should be in factor data type.

Converting Date-Time Field

In the code chunk below, *data-time_parse()* of **clock** package is used to convert *Timestamp* filed from *Character* data type to *date-time* (i.e. dttm) format.

```
gps$Timestamp <- date_time_parse(gps$Timestamp
                 zone = "",
                 format = "%m/%d/%Y %H:%M:%S")
gps$day <- as.factor(get_day(gps$Timestamp))
```

Note:

- **clock** is a new package released by RStudio on 31st March 2021. For more information, have a look at this blog.

In the code chunk below, *as_factor()* of **forcats** package is used to convert values in id field from numerical to factor data type.

```
gps$id <- as_factor(gps$id)
```

```
gps
```

```
## # A tibble: 685,169 x 5
##    Timestamp           id      lat  long day
##    <dttm>              <fct> <dbl> <dbl> <fct>
##  1 2014-01-06 06:28:01 35     36.1  24.9 6
##  2 2014-01-06 06:28:01 35     36.1  24.9 6
##  3 2014-01-06 06:28:03 35     36.1  24.9 6
##  4 2014-01-06 06:28:05 35     36.1  24.9 6
##  5 2014-01-06 06:28:06 35     36.1  24.9 6
##  6 2014-01-06 06:28:07 35     36.1  24.9 6
##  7 2014-01-06 06:28:09 35     36.1  24.9 6
##  8 2014-01-06 06:28:10 35     36.1  24.9 6
##  9 2014-01-06 06:28:11 35     36.1  24.9 6
## 10 2014-01-06 06:28:12 35     36.1  24.9 6
## # ... with 685,159 more rows
```

Notice that the Timesstamp field is in dttm (i.e. date-time) format and the id field is in factor data type.

Converting Aspatial Data into a Simple Feature Data Frame

Code chunk below converts *gps* data frame into a simple feature data frame by using *st_as_sf()* of **sf** packages

```
gps_sf <- st_as_sf(gps,
                   coords = c("long", "lat"),
                       crs= 4326)
```

Things to learn from the arguments:

- The *coords* argument requires you to provide the column name of the x-coordinates (i.e. long) first then followed by the column name of the y-coordinates (i.e. lat).
- The *crs* argument required you to provide the coordinates system in epsg format. EPSG: 4326 is wgs84 Geographic Coordinate System. You can search for other country's epsg code by referring to epsg.io.

```
gps_sf
```

```
## Simple feature collection with 685169 features and
## Geometry type: POINT
## Dimension:     XY
## Bounding box:  xmin: 24.82509 ymin: 36.04802 xmax:
## Geodetic CRS:  WGS 84
## # A tibble: 685,169 x 4
##    Timestamp           id    day                geom
##  * <dttm>              <fct> <fct>           <POINT
##  1 2014-01-06 06:28:01 35    6     (24.87469 36.07
##  2 2014-01-06 06:28:01 35    6      (24.8746 36.07
##  3 2014-01-06 06:28:03 35    6     (24.87444 36.07
##  4 2014-01-06 06:28:05 35    6     (24.87425 36.07
##  5 2014-01-06 06:28:06 35    6     (24.87417 36.07
##  6 2014-01-06 06:28:07 35    6     (24.87406 36.07
##  7 2014-01-06 06:28:09 35    6     (24.87391 36.07
##  8 2014-01-06 06:28:10 35    6     (24.87381 36.07
##  9 2014-01-06 06:28:11 35    6     (24.87374 36.07
## 10 2014-01-06 06:28:12 35    6     (24.87362 36.07
## # ... with 685,159 more rows
```

Creating Movement Path from GPS Points

Code chunk below joins the gps points into movement paths by using the drivers' IDs as unique identifiers.

```
gps_path <- gps_sf %>%
  group_by(id, day) %>%
  summarize(m = mean(Timestamp),
            do_union=FALSE) %>%
  st_cast("LINESTRING")
```

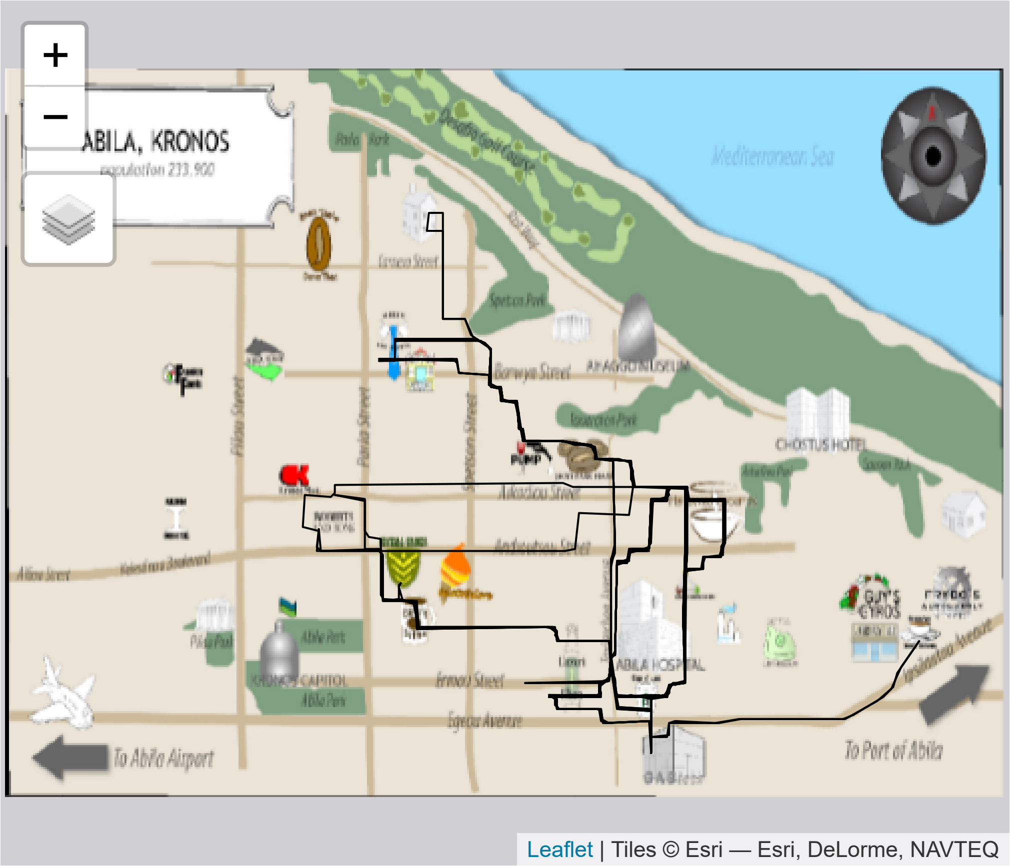
```
gps_path
```

```
## Simple feature collection with 508 features and 3
## Geometry type: LINESTRING
## Dimension:     XY
## Bounding box:  xmin: 24.82509 ymin: 36.04802 xmax:
## Geodetic CRS:  WGS 84
## # A tibble: 508 x 4
## # Groups:   id [40]
##    id    day   m
##    <fct> <fct> <dttm>
##  1 1     6     2014-01-06 15:02:08 (24.88258 36.06
##  2 1     7     2014-01-07 12:41:07 (24.87957 36.04
##  3 1     8     2014-01-08 14:35:25 (24.88265 36.06
##  4 1     9     2014-01-09 12:04:45 (24.88261 36.06
##  5 1     10    2014-01-10 16:04:58 (24.88265 36.06
##  6 1     11    2014-01-11 16:18:32 (24.88258 36.06
##  7 1     12    2014-01-12 13:31:05 (24.88259 36.06
##  8 1     13    2014-01-13 13:46:15 (24.88265 36.06
##  9 1     14    2014-01-14 14:04:23 (24.88261 36.06
## 10 1     15    2014-01-15 15:33:54 (24.88263 36.06
## # ... with 498 more rows
```

Note: I learn this trick from this issue.

Plotting the gps Paths

Write a code chunk to overplot the gps path of driver ID 1 onto the background tourist map.



The solution:

```
gps_path_selected <- gps_path %>%
  filter(id==1)
tmap_mode("view")
tm_shape(bgmap) +
  tm_rgb(bgmap, r = 1,g = 2,b = 3,
         alpha = NA,
         saturation = 1,
         interpolate = TRUE,
         max.value = 255) +
  tm_shape(gps_path_selected) +
  tm_lines()
```

Creating animated map with tmap_animation()

In the code chunk below, *tmap_animation()* of **tmap** package is used to create an animated gif for drivers' paths.

```
m <- tm_shape(bgmap) +
  tm_rgb(bgmap, r = 1,g = 2,b = 3,
       alpha = NA,
       saturation = 1,
       interpolate = TRUE,
       max.value = 255) +
  tm_shape(gps_path) +
  tm_lines() +
  tm_facets(along = "id")

tmap_animation(m,
               filename = "gif/drivers.gif",
               delay=40)
```