

Hands-on Exercise 6: Handling, Processing Visualising and Analysing Movement Data

Dr. Kam Tin Seong

Assoc. Professor of Information Systems

**School of Computing and Information Systems,
Singapore Management University**

2020-2-15 (updated: 2022-05-21)

Content

In this hands-on exercise, you will learn how to handling, processing, visualising and analysing movement data using R. By the end of this hands-on exercise, you will be able to:

- import geospatial data in *wkt* format into R and saved the imported data as **simple feature** objects by using **sf** package,
- mapping geospatial data using **tmap** package,
- import movement data in *wkt* format into R and saved the imported data as **simple feature** objects by using **sf** package,
- process movement data by using **sf** and **tidyverse** packages,
- visualising movement data by using **tmap** and **ggplot2** package,
- analysing movement data by using R methods.

Getting Started

In this Hands-on Exercise, the following R packages will be used:

- `sf`, an R package specially designed to handle geospatial data in simple feature objects.

Write a code chunk to check, install and launch `readr`, `sf` and `tmap` packages of R

Visualising Geographical Data

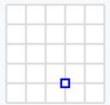
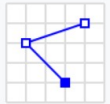

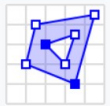
Importing wkt data

- *Well-known text (WKT)* is a human readable representation for spatial objects like points, lines, or enclosed areas on a map. Figure below shows the structure of point, line and polygons data in wkt format.

In the code chunk below, `read_sf()` of **sf** package is used to parse *School.csv* into R as an sf data.frame.

```
schools <- read_sf("data/wkt/Schools.csv",  
                  options = "GEOM_POSSIBLE_NAMES")
```

DIY: Using the code you had learned, parses *Pubs.csv*, *Apartments.csv*, *Buildings.csv*, *Employer.csv*, and *Restaurants.csv* into R

Type	Examples	
Point		POINT (30 10)
LineString		LINESTRING (30 10, 10 30, 40 40)
Polygon		POLYGON ((30 10, 40 40, 20 40, 10 20, 30 10))
		POLYGON ((35 10, 45 45, 15 40, 10 20, 35 10), (20 30, 35 35, 30 20, 20 30))

Structure of a simple point feature data.frame

- After importing the data file into R, it is important for us to review the data object.

```
print(schools)
```

```
## Simple feature collection with 4 features and 4 fields
## Geometry type: POINT
## Dimension:      XY
## Bounding box:   xmin: -4701.463 ymin: 1607.984 xmax: -376.7505 ymax: 6556.032
## CRS:           NA
## # A tibble: 4 × 5
##   schoolId monthlyCost maxEnrollment      location buildingId
##   <chr>      <chr>      <chr>          <POINT> <chr>
## 1 0         12.81244502 242          (-376.7505 1607.984) 662
## 2 450       91.14351385 418          (-2597.448 3194.155) 943
## 3 900       38.00537955 394          (-2539.158 6556.032) 262
## 4 1350      73.19785215 384          (-4701.463 5141.763) 123
```

Structure of a simple polygon feature data.frame

Now we will print the *buildings* simple feature data.frame.

```
print(buildings)
```

```
## Simple feature collection with 1042 features and 4 fields
## Geometry type: POLYGON
## Dimension:      XY
## Bounding box:   xmin: -4762.191 ymin: -30.08359 xmax: 2650 ymax: 7850.037
## CRS:            NA
## # A tibble: 1,042 × 5
##   buildingId      location buildingType maxOccupancy units
##   <chr>          <POLYGON> <chr>      <chr>      <chr>
## 1 1 ((350.0639 4595.666, 390.0633 459... Commercial ""
## 2 2 ((-1926.973 2725.611, -1948.191 2... Residential "12"
## 3 3 ((685.6846 1552.131, 645.9985 154... Commercial ""
## 4 4 ((-976.7845 4542.382, -1053.288 4... Commercial ""
## 5 5 ((1259.306 3572.727, 1299.255 357... Residential "2"
## 6 6 ((478.8969 1082.484, 473.6596 113... Commercial ""
## 7 7 ((-1920.823 615.7447, -1960.818 6... Residential ""
## 8 8 ((-3302.657 5394.354, -3301.512 5... Commercial ""
## 9 9 ((-600.5789 4429.228, -495.9506 4... Commercial ""
## 10 10 ((-68.75908 5379.924, -28.78232 5... Residential "5"
## # ... with 1,032 more rows
```

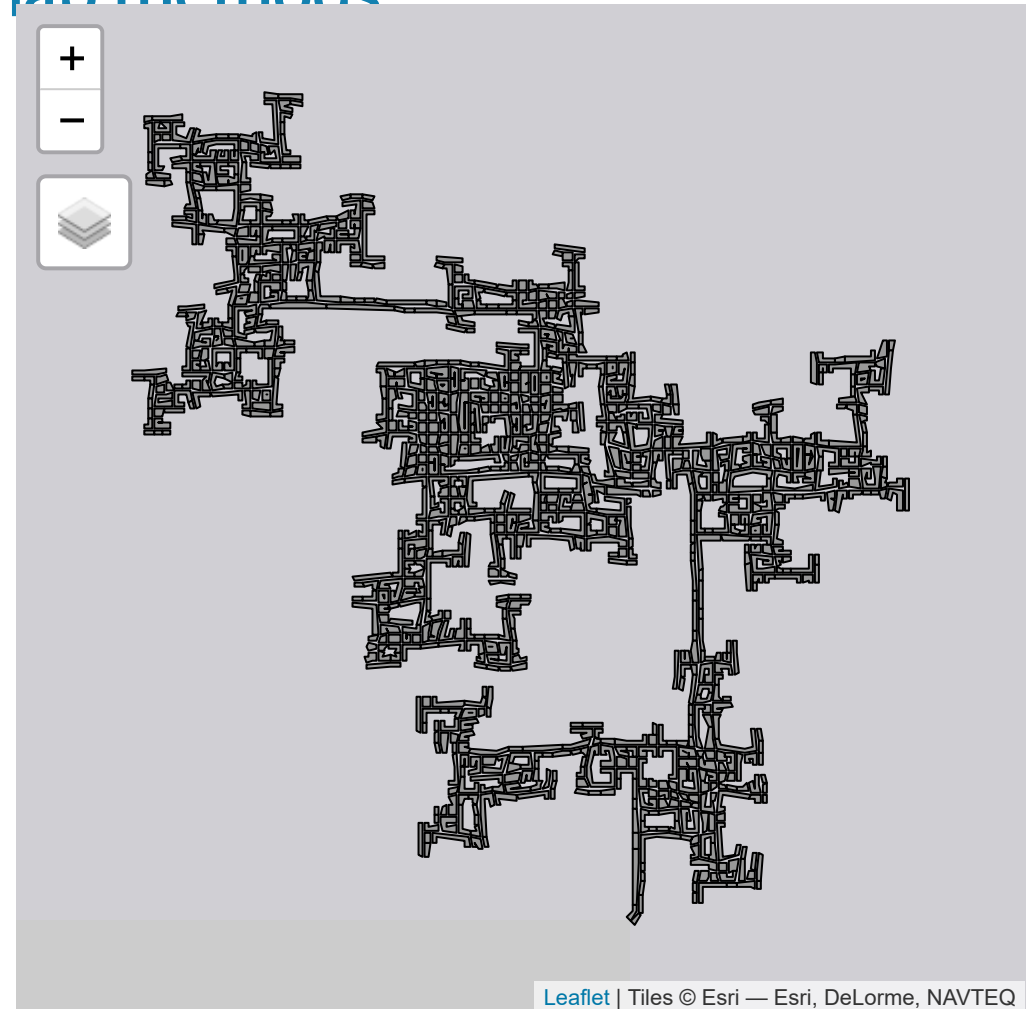
Plotting the building footprint map: tmap methods

The code chunk below plots the building polygon features by using `tm_polygon()`.

```
tmap_mode("view")
tm_shape(buildings)+
tm_polygons(col = "grey60",
            size = 1,
            border.col = "black",
            border.lwd = 1)
tmap_mode("plot")
```

Things to learn from the code chunk:

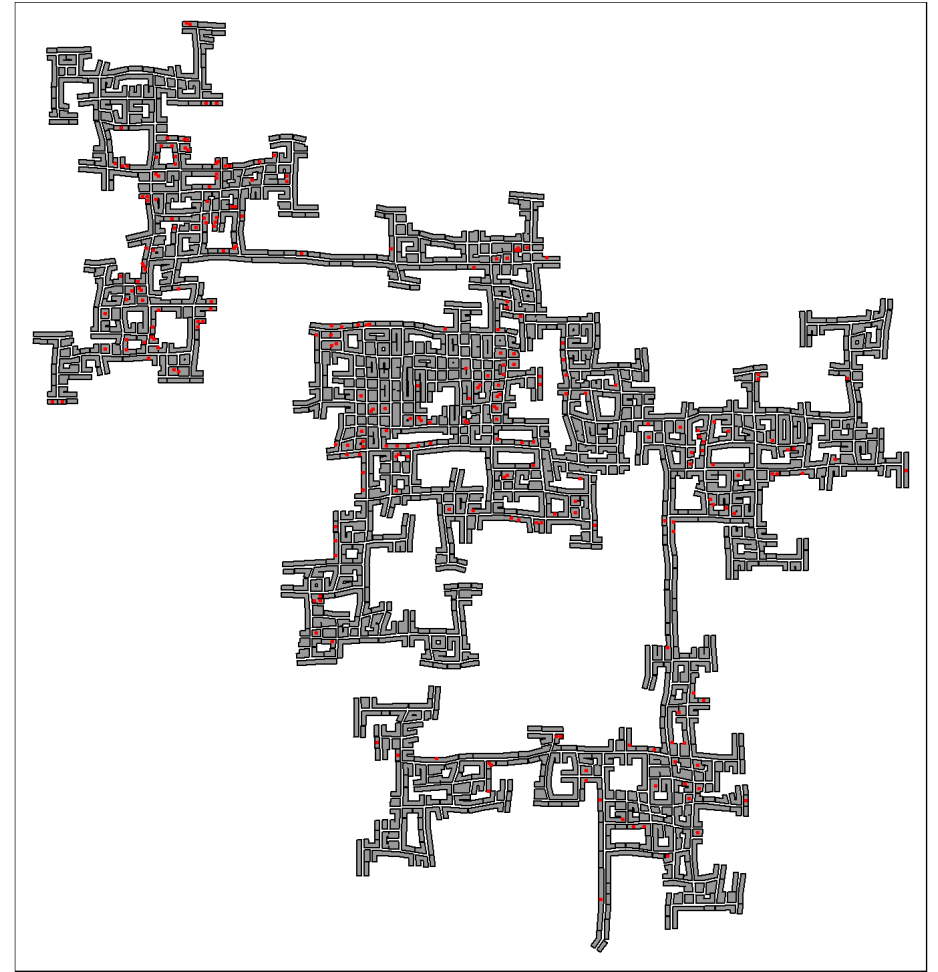
- `tmap_mode()` is used to switch the display from static mode (i.e. "plot") to interactive mode (i.e. "view").
- `tm_shape()` is used to create a **tmap-element** that specifies a spatial data object (i.e. buildings).
- `tm_polygon()` is used to create a **tmap-element** that draws polygon feature.



Plotting a composite map: tmap methods

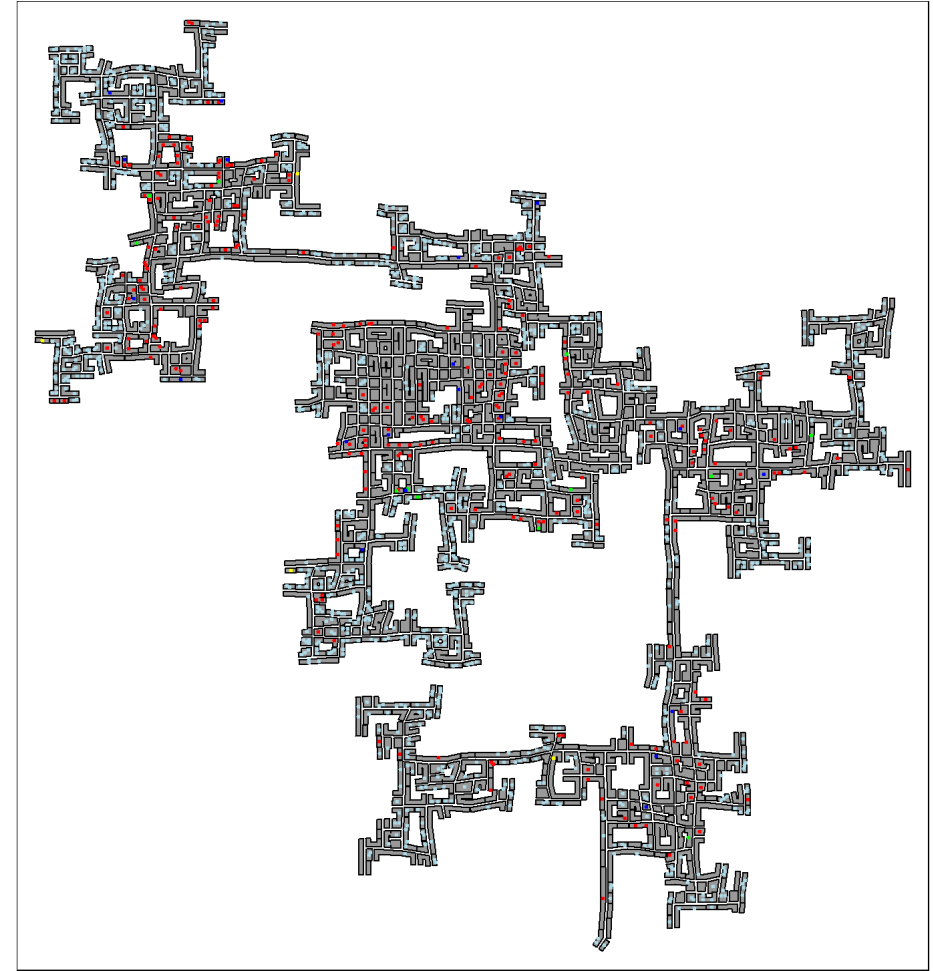
The code chunk below is used to plot a composite map by combining the buildings and employers simple feature data.frames.

```
tmap_mode("plot")
tm_shape(buildings)+
tm_polygons(col = "grey60",
            size = 1,
            border.col = "black",
            border.lwd = 1) +
tm_shape(employers) +
tm_dots(col = "red")
```



DIY: Plotting a composite map

The Task: Plot a composite map by combining *buildings, apartments, employers, pubs, restaurants, and schools*.



Movement Data

In this section, you will learn how to handle, process, visualise and analyse movement data. For the purpose of this hands-on exercise, *ParticipantStatusLogs1.csv* will be used.

Importing wkt data

DIY: By using the step you had learned, import *ParticipantStatusLogs1.csv* and save it as a simple feature data.frame.

```
logs <- read_sf("data/wkt/ParticipantStatusLogs1.wkt",  
               options = "GEOM_POSSIBLE_NAMES:"
```

Let us examine the structure of *logs* simple feature data.frame by using `glimpse()`.

```
glimpse(logs)
```

Notice that `read_sf()` failed to parse `timestamp` field into correct date-time data type.

Processing movement data

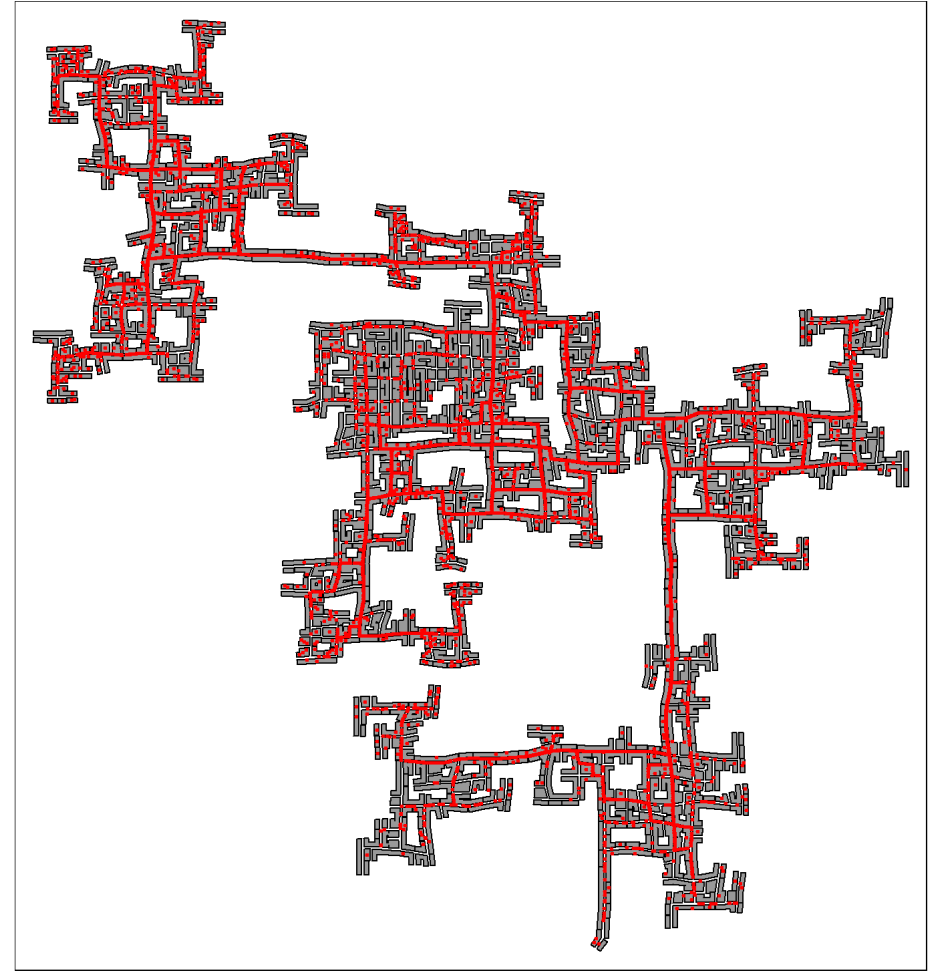
To process the movement data, the following steps will be performed:

- convert *timestamp* field from character data type to date-time data type by using `date_time_parse()` of clock package.
- derive a *day* field by using `get_day()` of clock package.
- extract records whereby *currentMode* field is equal to *Transport* class by using `filter()` of dplyr package.

DIY: Write a code chunk to perform the tasks described on the left.

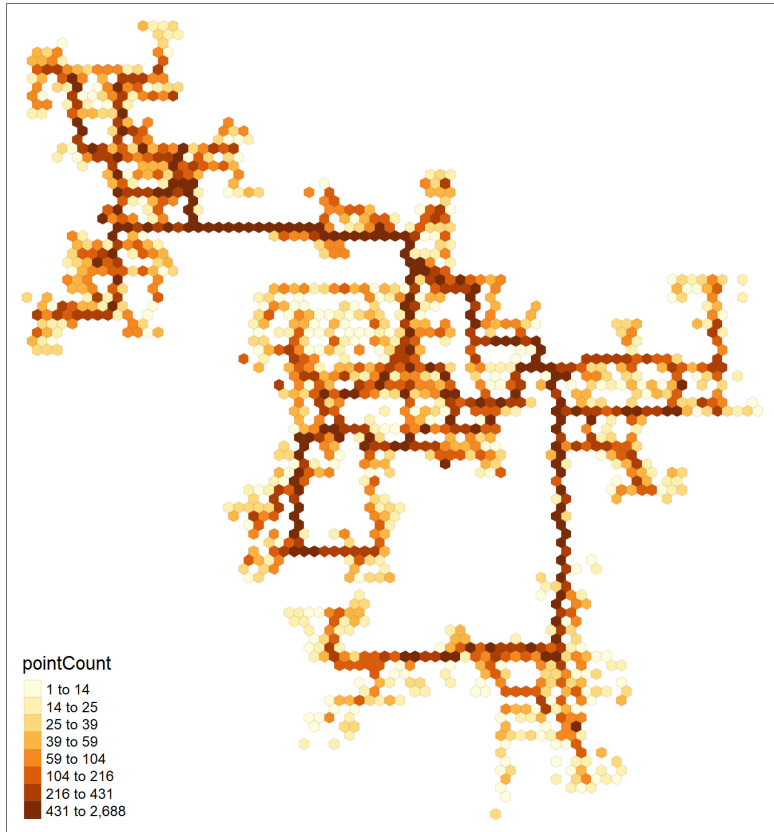
Plotting the moving data as points

- Using appropriate tmap functions to create a map look similar to the figure on the left.



Hexagon Binning Map

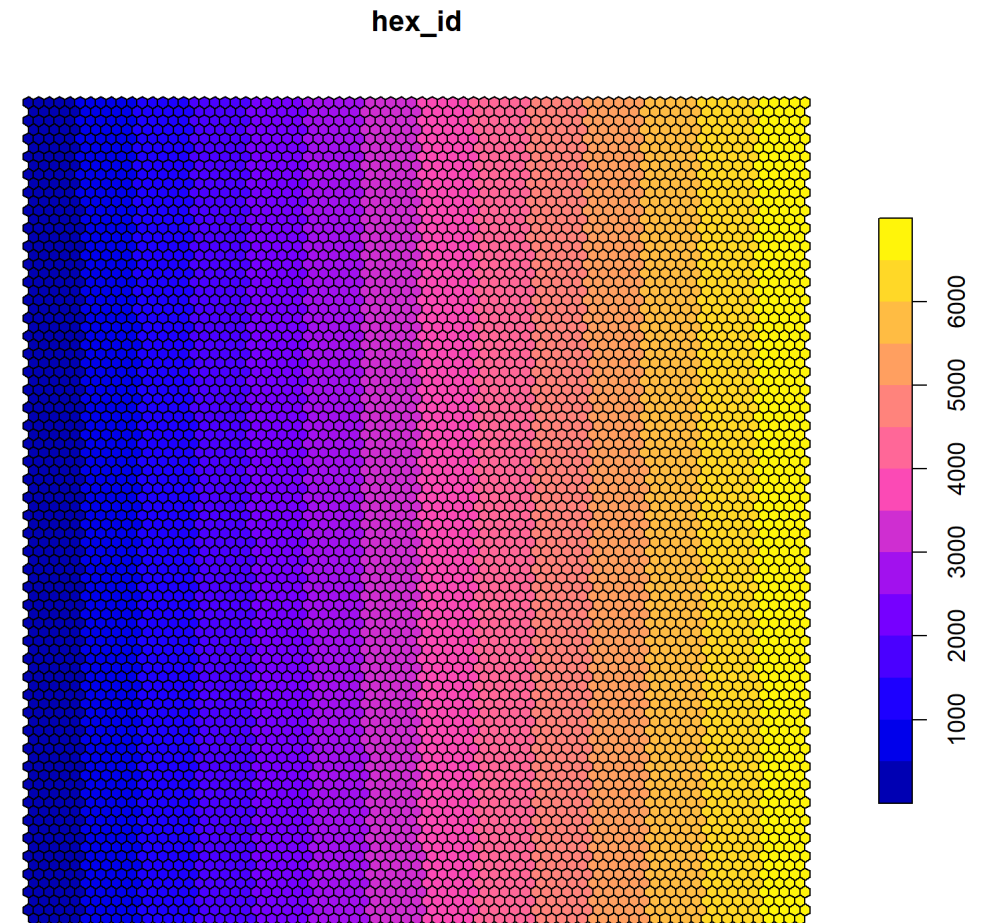
In this section, you will learn how to create a [hexagon binning map](#) by using R.



Computing the haxegons

In the code chunk below, `st_make_grid()` of `sf` package is used to create haxegons

```
hex <- st_make_grid(buildings,  
                    cellsize=100,  
                    square=FALSE) %>%  
  st_sf() %>%  
  rowid_to_column('hex_id')  
plot(hex)
```



Performing point in polygon overlay

The code chunk below perform point in polygon overlay by using `[st_join()]` of sf package.

```
points_in_hex <- st_join(logs_selected,  
                          hex,  
                          join=st_within)  
#plot(points_in_hex, pch='.')
```

Performing point in polygon count

In the code chunk below, `st_join()` of sf package is used to count the number of event points in the hexagons.

```
points_in_hex <- st_join(logs_selected,
                          hex,
                          join=st_within) %>%
  st_set_geometry(NULL) %>%
  count(name='pointCount', hex_id)
head(points_in_hex)
```

```
## # A tibble: 6 × 2
##   hex_id pointCount
##   <int>      <int>
## 1     169         35
## 2     212         56
## 3     225         21
## 4     226         94
## 5     227         22
## 6     228         45
```


Performing relational join

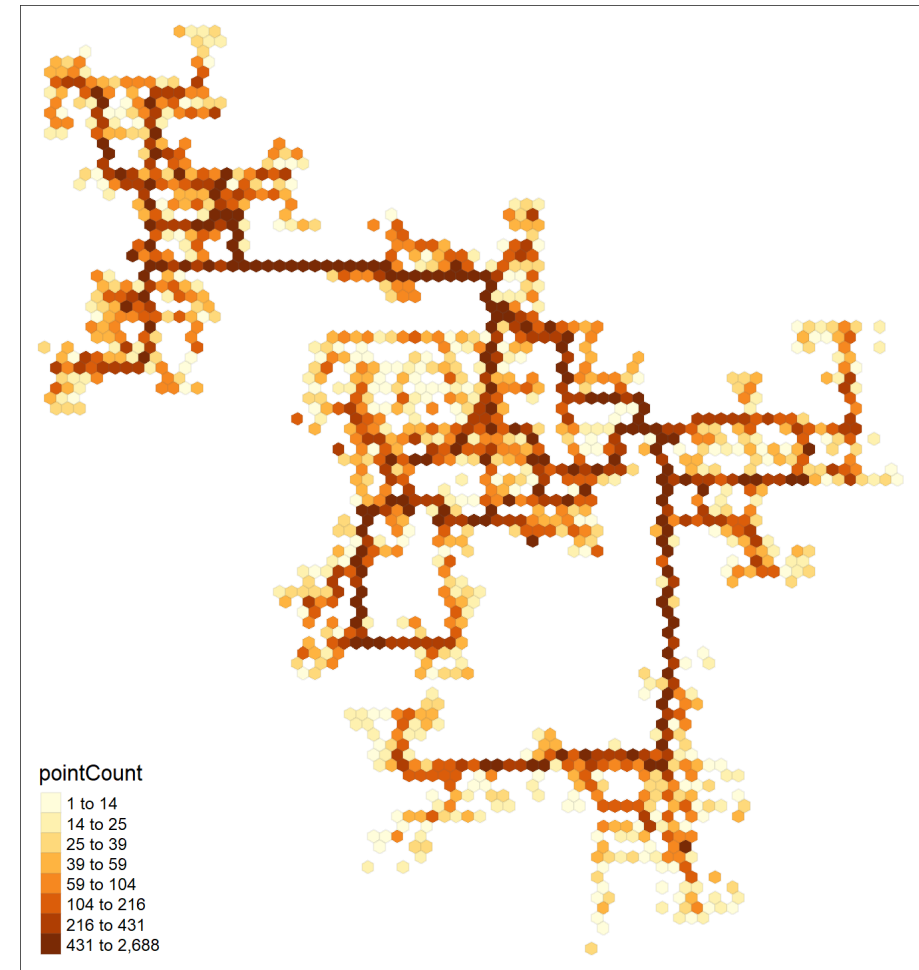
In the code chunk below, `left_join()` of `dplyr` package is used to perform a left-join by using *hex* as the target table and *points_in_hex* as the join table. The join ID is *hex_id*.

```
hex_combined <- hex %>%  
  left_join(points_in_hex,  
            by = 'hex_id') %>%  
  replace(is.na(.), 0)
```

Plotting the hexagon binning map

In the code chunk below, tmap package is used to create the hexagon binning map.

```
tm_shape(hex_combined %>%  
  filter(pointCount > 0))+  
  tm_fill("pointCount",  
    n = 8,  
    style = "quantile") +  
  tm_borders(alpha = 0.1)
```



Plotting Movement Path using R

In this section, you will learn how to plot movement path using R.

Creating movement path from event points

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

```
logs_path <- logs_selected %>%  
  group_by(participantId, day) %>%  
  summarize(m = mean(Timestamp),  
            do_union=FALSE) %>%  
  st_cast("LINESTRING")
```

```
print(logs_path)
```

```
## Simple feature collection with 5781 features and 3  
## Geometry type: LINESTRING  
## Dimension:      XY  
## Bounding box:   xmin: -4616.828 ymin: 35.4377 xmax:  
## CRS:            NA  
## # A tibble: 5,781 × 4  
## # Groups:   participantId [1,011]  
##   participantId   day m  
##   <chr>          <int> <dtm>  
## 1 0              1 2022-03-01 13:34:23 (-2721.  
## 2 0              2 2022-03-02 14:19:50 (-2721.  
## 3 0              3 2022-03-03 13:39:13 (-2721.  
## 4 0              4 2022-03-04 13:38:11 (-2721.  
## 5 0              5 2022-03-05 13:08:02 (-2721.  
## 6 0              6 2022-03-06 06:28:00 (-2721.  
## 7 1              1 2022-03-01 18:07:24 (-1531.  
## 8 1              2 2022-03-02 16:57:05 (-2619.  
## 9 1              3 2022-03-03 14:13:40 (-260.4  
## 10 1             4 2022-03-04 14:31:45 (-3903.  
## # ... with 5,771 more rows
```

Plotting the Movement Paths

Write a code chunk to overplot the gps path of participant ID = 0 onto the background building footprint map.

