Hands-on Exercise 2: Geospatial Data Wrangling with R

Dr. Kam Tin SeongAssoc. Professor of Information Systems (Practice)

10 January 2019 (updated: 2021-05-03)

Table of Contents

# Overview

In this hands-on exercise, you will learn how to handle geospatial data in R by using appropriate R packages.

## Learning Outcome

By the end of this hands-on exercise, you should acquire the following competencies:

* import geospatial data by using appropriate functions of sf packages,
* import geospatial data as R spatial object type by using appropriate functions of sp packages,
* georeference geospatial data using either apporpriate rgeo  
  understand the concepts of Spatial Data Frame and to apply these understanding in handling geospatial data.
* understand the concepts of simple features and to apply these understanding in handling geospatial data.
* perform geospatial data handling tasks by using appropriate functions of sp, rgdal, rgeos and/or sf packages.

## Data Acquisition

Before you can start using R, you are required to extract the necessary data sets from the appropriate source

* Master Plan 2014 Subzone Boundary (Web) from [data.gov.sg](https://data.gov.sg/)
* Pre-Schools Location from [data.gov.sg](https://data.gov.sg/)
* Cycling Path from [LTADataMall](https://www.mytransport.sg/content/mytransport/home/dataMall.html)
* [AirBNB Singapore Listing Data](http://insideairbnb.com/get-the-data.html),download only listings.csv.

## Getting Started

Before we get started, it is important for us to ensure that all the R packages we need have been installed.

* Using the steps you leanred earlier, check if **sp**, **rgdal** and **rgeos** have been installed, if not, then install the uninstalled package. After the installation is completed, launch **sp**, **rgdal** and **rgeos**.

The code chunk:

packages = c('sp', 'rgdal', 'rgeos', 'sf', 'tidyverse')  
for (p in packages){  
 if(!require(p, character.only = T)){  
 install.packages(p)  
 }  
 library(p,character.only = T)  
}

# Working with **sf** package

In this hands-on exercise, you are required to import the following geospatial data into R:

* *MP14\_SUBZONE\_WEB\_PL* - a polygon feature layer in **ESRI shapefile** format,
* *CyclingPath* - a line feature layer in **ESRI shapefile** format, and
* *PreSchool* - a point feature layer in **kml** file format.

## Importing GIS data using *st\_read()*

### Importing polygon feature data in shapefile format

The code chunk below uses *st\_read()* function of **sf** package to import MP14\_SUBZONE\_WEB\_PL data into R as simple feature dataframes.

sf\_mpsz = st\_read(dsn = "data/geospatial",   
 layer = "MP14\_SUBZONE\_WEB\_PL")

## Reading layer `MP14\_SUBZONE\_WEB\_PL' from data source `D:\tskam\GeoDSA\Hands-on\_Ex\Hands-on\_Ex02\data\geospatial' using driver `ESRI Shapefile'  
## Simple feature collection with 323 features and 15 fields  
## Geometry type: MULTIPOLYGON  
## Dimension: XY  
## Bounding box: xmin: 2667.538 ymin: 15748.72 xmax: 56396.44 ymax: 50256.33  
## Projected CRS: SVY21

### Importing polyline feature data in shapefile format

The code chunk below uses *st\_read()* function of **sf** package to import the CyclingPath layer into R as simple feature dataframes.

sf\_cyclingpath = st\_read(dsn = "data/geospatial",   
 layer = "CyclingPath")

## Reading layer `CyclingPath' from data source `D:\tskam\GeoDSA\Hands-on\_Ex\Hands-on\_Ex02\data\geospatial' using driver `ESRI Shapefile'  
## Simple feature collection with 1625 features and 2 fields  
## Geometry type: LINESTRING  
## Dimension: XY  
## Bounding box: xmin: 12711.19 ymin: 28711.33 xmax: 42626.09 ymax: 48948.15  
## Projected CRS: SVY21

### Importing GIS data in kml format

sf\_preschool = st\_read("data/geospatial/pre-schools-location-kml.kml")

## Reading layer `PRESCHOOLS\_LOCATION' from data source `D:\tskam\GeoDSA\Hands-on\_Ex\Hands-on\_Ex02\data\geospatial\pre-schools-location-kml.kml' using driver `KML'  
## Simple feature collection with 1359 features and 2 fields  
## Geometry type: POINT  
## Dimension: XYZ  
## Bounding box: xmin: 103.6824 ymin: 1.248403 xmax: 103.9897 ymax: 1.462134  
## z\_range: zmin: 0 zmax: 0  
## Geodetic CRS: WGS 84

## Checking the contents of a SpatialDataFrame

You can check the contents of mpsz\_rg data object by using *summary*.

The code chunk:

summary(sf\_mpsz)

## OBJECTID SUBZONE\_NO SUBZONE\_N SUBZONE\_C   
## Min. : 1.0 Min. : 1.000 Length:323 Length:323   
## 1st Qu.: 81.5 1st Qu.: 2.000 Class :character Class :character   
## Median :162.0 Median : 4.000 Mode :character Mode :character   
## Mean :162.0 Mean : 4.625   
## 3rd Qu.:242.5 3rd Qu.: 6.500   
## Max. :323.0 Max. :17.000   
## CA\_IND PLN\_AREA\_N PLN\_AREA\_C REGION\_N   
## Length:323 Length:323 Length:323 Length:323   
## Class :character Class :character Class :character Class :character   
## Mode :character Mode :character Mode :character Mode :character   
##   
##   
##   
## REGION\_C INC\_CRC FMEL\_UPD\_D X\_ADDR   
## Length:323 Length:323 Min. :2014-12-05 Min. : 5093   
## Class :character Class :character 1st Qu.:2014-12-05 1st Qu.:21864   
## Mode :character Mode :character Median :2014-12-05 Median :28465   
## Mean :2014-12-05 Mean :27257   
## 3rd Qu.:2014-12-05 3rd Qu.:31674   
## Max. :2014-12-05 Max. :50425   
## Y\_ADDR SHAPE\_Leng SHAPE\_Area geometry   
## Min. :19579 Min. : 871.5 Min. : 39438 MULTIPOLYGON :323   
## 1st Qu.:31776 1st Qu.: 3709.6 1st Qu.: 628261 epsg:NA : 0   
## Median :35113 Median : 5211.9 Median : 1229894 +proj=tmer...: 0   
## Mean :36106 Mean : 6524.4 Mean : 2420882   
## 3rd Qu.:39869 3rd Qu.: 6942.6 3rd Qu.: 2106483   
## Max. :49553 Max. :68083.9 Max. :69748299

Let’s view the first few records in the spatial data object.

The code chunk

head(sf\_mpsz, n=4)

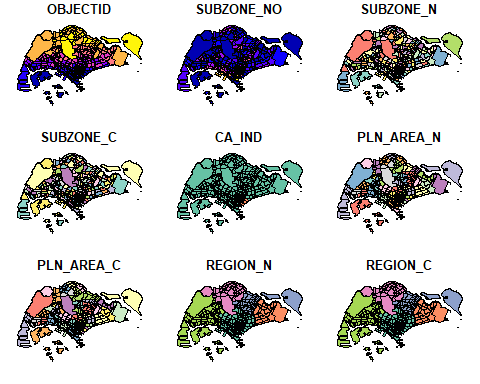
## Simple feature collection with 4 features and 15 fields  
## Geometry type: MULTIPOLYGON  
## Dimension: XY  
## Bounding box: xmin: 26403.48 ymin: 28369.47 xmax: 32362.39 ymax: 30396.46  
## Projected CRS: SVY21  
## OBJECTID SUBZONE\_NO SUBZONE\_N SUBZONE\_C CA\_IND PLN\_AREA\_N  
## 1 1 1 MARINA SOUTH MSSZ01 Y MARINA SOUTH  
## 2 2 1 PEARL'S HILL OTSZ01 Y OUTRAM  
## 3 3 3 BOAT QUAY SRSZ03 Y SINGAPORE RIVER  
## 4 4 8 HENDERSON HILL BMSZ08 N BUKIT MERAH  
## PLN\_AREA\_C REGION\_N REGION\_C INC\_CRC FMEL\_UPD\_D X\_ADDR  
## 1 MS CENTRAL REGION CR 5ED7EB253F99252E 2014-12-05 31595.84  
## 2 OT CENTRAL REGION CR 8C7149B9EB32EEFC 2014-12-05 28679.06  
## 3 SR CENTRAL REGION CR C35FEFF02B13E0E5 2014-12-05 29654.96  
## 4 BM CENTRAL REGION CR 3775D82C5DDBEFBD 2014-12-05 26782.83  
## Y\_ADDR SHAPE\_Leng SHAPE\_Area geometry  
## 1 29220.19 5267.381 1630379.3 MULTIPOLYGON (((31495.56 30...  
## 2 29782.05 3506.107 559816.2 MULTIPOLYGON (((29092.28 30...  
## 3 29974.66 1740.926 160807.5 MULTIPOLYGON (((29932.33 29...  
## 4 29933.77 3313.625 595428.9 MULTIPOLYGON (((27131.28 30...

### Plotting the sptial data

To view the spatial data, *plot* function can be used.

The code chunk:

plot(sf\_mpsz)



## Working with Projection

Task 2: Assigning EPSG code to sf\_mpsz simple feature dataframe.

The solution:

First, checking the projection of *sf\_mpsz* by using *st\_crs()*.

st\_crs(sf\_mpsz)

## Coordinate Reference System:  
## User input: SVY21   
## wkt:  
## PROJCRS["SVY21",  
## BASEGEOGCRS["SVY21[WGS84]",  
## DATUM["World Geodetic System 1984",  
## ELLIPSOID["WGS 84",6378137,298.257223563,  
## LENGTHUNIT["metre",1]],  
## ID["EPSG",6326]],  
## PRIMEM["Greenwich",0,  
## ANGLEUNIT["Degree",0.0174532925199433]]],  
## CONVERSION["unnamed",  
## METHOD["Transverse Mercator",  
## ID["EPSG",9807]],  
## PARAMETER["Latitude of natural origin",1.36666666666667,  
## ANGLEUNIT["Degree",0.0174532925199433],  
## ID["EPSG",8801]],  
## PARAMETER["Longitude of natural origin",103.833333333333,  
## ANGLEUNIT["Degree",0.0174532925199433],  
## ID["EPSG",8802]],  
## PARAMETER["Scale factor at natural origin",1,  
## SCALEUNIT["unity",1],  
## ID["EPSG",8805]],  
## PARAMETER["False easting",28001.642,  
## LENGTHUNIT["metre",1],  
## ID["EPSG",8806]],  
## PARAMETER["False northing",38744.572,  
## LENGTHUNIT["metre",1],  
## ID["EPSG",8807]]],  
## CS[Cartesian,2],  
## AXIS["(E)",east,  
## ORDER[1],  
## LENGTHUNIT["metre",1,  
## ID["EPSG",9001]]],  
## AXIS["(N)",north,  
## ORDER[2],  
## LENGTHUNIT["metre",1,  
## ID["EPSG",9001]]]]

Next, assigning EPSG 3414 to sf\_mpsz simple feature dataframe by using *st\_set\_crs()*.

sf\_mpsz3414 <- st\_set\_crs(sf\_mpsz, 3414)

Lets check the CSR again.

st\_crs(sf\_mpsz3414)

## Coordinate Reference System:  
## User input: EPSG:3414   
## wkt:  
## PROJCRS["SVY21 / Singapore TM",  
## BASEGEOGCRS["SVY21",  
## DATUM["SVY21",  
## ELLIPSOID["WGS 84",6378137,298.257223563,  
## LENGTHUNIT["metre",1]]],  
## PRIMEM["Greenwich",0,  
## ANGLEUNIT["degree",0.0174532925199433]],  
## ID["EPSG",4757]],  
## CONVERSION["Singapore Transverse Mercator",  
## METHOD["Transverse Mercator",  
## ID["EPSG",9807]],  
## PARAMETER["Latitude of natural origin",1.36666666666667,  
## ANGLEUNIT["degree",0.0174532925199433],  
## ID["EPSG",8801]],  
## PARAMETER["Longitude of natural origin",103.833333333333,  
## ANGLEUNIT["degree",0.0174532925199433],  
## ID["EPSG",8802]],  
## PARAMETER["Scale factor at natural origin",1,  
## SCALEUNIT["unity",1],  
## ID["EPSG",8805]],  
## PARAMETER["False easting",28001.642,  
## LENGTHUNIT["metre",1],  
## ID["EPSG",8806]],  
## PARAMETER["False northing",38744.572,  
## LENGTHUNIT["metre",1],  
## ID["EPSG",8807]]],  
## CS[Cartesian,2],  
## AXIS["northing (N)",north,  
## ORDER[1],  
## LENGTHUNIT["metre",1]],  
## AXIS["easting (E)",east,  
## ORDER[2],  
## LENGTHUNIT["metre",1]],  
## USAGE[  
## SCOPE["Cadastre, engineering survey, topographic mapping."],  
## AREA["Singapore - onshore and offshore."],  
## BBOX[1.13,103.59,1.47,104.07]],  
## ID["EPSG",3414]]

### Transforming the projection of sf\_preschool from wgs84 to svy21.

The solution:

First, checking the projection of sf\_preschool

st\_crs(sf\_preschool)

## Coordinate Reference System:  
## User input: WGS 84   
## wkt:  
## GEOGCRS["WGS 84",  
## DATUM["World Geodetic System 1984",  
## ELLIPSOID["WGS 84",6378137,298.257223563,  
## LENGTHUNIT["metre",1]]],  
## PRIMEM["Greenwich",0,  
## ANGLEUNIT["degree",0.0174532925199433]],  
## CS[ellipsoidal,2],  
## AXIS["geodetic latitude (Lat)",north,  
## ORDER[1],  
## ANGLEUNIT["degree",0.0174532925199433]],  
## AXIS["geodetic longitude (Lon)",east,  
## ORDER[2],  
## ANGLEUNIT["degree",0.0174532925199433]],  
## ID["EPSG",4326]]

Next, transforming sf\_preschool simple feature dataframe onto svy21 projected coordinate system (i.e. EPSG 3414) by using *st\_transform()*.

sf\_preschool3414 <- st\_transform(sf\_preschool, 3414)  
st\_crs(sf\_preschool3414)

## Coordinate Reference System:  
## User input: EPSG:3414   
## wkt:  
## PROJCRS["SVY21 / Singapore TM",  
## BASEGEOGCRS["SVY21",  
## DATUM["SVY21",  
## ELLIPSOID["WGS 84",6378137,298.257223563,  
## LENGTHUNIT["metre",1]]],  
## PRIMEM["Greenwich",0,  
## ANGLEUNIT["degree",0.0174532925199433]],  
## ID["EPSG",4757]],  
## CONVERSION["Singapore Transverse Mercator",  
## METHOD["Transverse Mercator",  
## ID["EPSG",9807]],  
## PARAMETER["Latitude of natural origin",1.36666666666667,  
## ANGLEUNIT["degree",0.0174532925199433],  
## ID["EPSG",8801]],  
## PARAMETER["Longitude of natural origin",103.833333333333,  
## ANGLEUNIT["degree",0.0174532925199433],  
## ID["EPSG",8802]],  
## PARAMETER["Scale factor at natural origin",1,  
## SCALEUNIT["unity",1],  
## ID["EPSG",8805]],  
## PARAMETER["False easting",28001.642,  
## LENGTHUNIT["metre",1],  
## ID["EPSG",8806]],  
## PARAMETER["False northing",38744.572,  
## LENGTHUNIT["metre",1],  
## ID["EPSG",8807]]],  
## CS[Cartesian,2],  
## AXIS["northing (N)",north,  
## ORDER[1],  
## LENGTHUNIT["metre",1]],  
## AXIS["easting (E)",east,  
## ORDER[2],  
## LENGTHUNIT["metre",1]],  
## USAGE[  
## SCOPE["Cadastre, engineering survey, topographic mapping."],  
## AREA["Singapore - onshore and offshore."],  
## BBOX[1.13,103.59,1.47,104.07]],  
## ID["EPSG",3414]]

## Importing and Converting Aspatial Data into simple features

In this section, you will learn how to import Singapore AirBNB listing data and convert the data into simple feature data.frame.

### Importing the aspatial data

Write the code chunk to import \*listing.csv into R.

listings <- read\_csv("data/aspatial/listings.csv")

##   
## -- Column specification --------------------------------------------------------  
## cols(  
## id = col\_double(),  
## name = col\_character(),  
## host\_id = col\_double(),  
## host\_name = col\_character(),  
## neighbourhood\_group = col\_character(),  
## neighbourhood = col\_character(),  
## latitude = col\_double(),  
## longitude = col\_double(),  
## room\_type = col\_character(),  
## price = col\_double(),  
## minimum\_nights = col\_double(),  
## number\_of\_reviews = col\_double(),  
## last\_review = col\_date(format = ""),  
## reviews\_per\_month = col\_double(),  
## calculated\_host\_listings\_count = col\_double(),  
## availability\_365 = col\_double()  
## )

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

### Creating a sf data frame

The code chunk below converts *listings* data frame into a simple feature data frame by using [*st\_as\_sf()*](https://r-spatial.github.io/sf/reference/st_as_sf.html) of **sf** packages

Things to learn from the arguments:

* The *coords* argument requires you to provide the column name of the x-coordinates first then followed by the column name of the y-coordinates.
* The *crs* argument required you to provide the coordinates system in epsg format. [EPSG: 3414](https://epsg.io/3414) is Singapore SVY21 Projected Coordinate System. You can search for other country’s epsg code by refering to [epsg.io](https://epsg.io/). ]]

.pull-right[

listing\_sf <- st\_as\_sf(listings,   
 coords = c("longitude", "latitude"),  
 crs= 4326)

### Trasforming Projection

Next, we will transform the listing simple feature from *wgs84* geographic coordinates systems to *svy21* projected coordinates system

listing\_sf <- st\_transform(listing\_sf, 3414)

## Conversion to sp class

Although simple feature data.frame is gaining popularity again sp’s Spatial\* classes, there are, however, many geospatial analysis packages require the input geospatial data in sp’s Spatial\* classes. In this section, you will learn how to convert simple feature data.frame to sp’s Spatial\* class.

### Converting point feature data.frame to SpatialPointsDataFrame

The code chunk below uses [*as\_Spatial()*](https://r-spatial.github.io/sf/reference/coerce-methods.html) of **sf** package to convert sf\_preschool3414 simple feature data.frame to sp’s Spatial\* class.

sp\_preschool <- as\_Spatial(sf\_preschool3414)

## Warning in showSRID(uprojargs, format = "PROJ", multiline = "NO", prefer\_proj  
## = prefer\_proj): Discarded datum Unknown based on WGS84 ellipsoid in Proj4  
## definition

## Warning in showSRID(SRS\_string, format = "PROJ", multiline = "NO", prefer\_proj =  
## prefer\_proj): Discarded datum SVY21 in Proj4 definition

Notice that the output is a **SpatialPointsDataFrame** class.

You can check the content of the SpatialPointsDataFrame by using *summary()* as shown in the code chunk below.

summary(sp\_preschool)

## Object of class SpatialPointsDataFrame  
## Coordinates:  
## min max  
## coords.x1 11203.01 45404.24  
## coords.x2 25667.60 49300.88  
## coords.x3 0.00 0.00  
## Is projected: TRUE   
## proj4string :  
## [+proj=tmerc +lat\_0=1.36666666666667 +lon\_0=103.833333333333 +k=1  
## +x\_0=28001.642 +y\_0=38744.572 +ellps=WGS84 +units=m +no\_defs]  
## Number of points: 1359  
## Data attributes:  
## Name Description   
## Length:1359 Length:1359   
## Class :character Class :character   
## Mode :character Mode :character

DIY: Using the steps you had learned, convert *sf\_mpsz3414* and *sf\_mpsz* simple feature data.frame to sp’s Spatial\* classes. After the conversion, examine the output spatial classes carefully. Write short notes to decribe your onservation of the output spatial classes.

## Geoprocessing with sf package

### Buffering

The scenario:

The authority is planning to upgrade the exiting cycling path. To do so, they need to acquire 5 metres reserve land on the both sides of the current cycling path. You are tasked to determine the extend of the land need to be acquired and their total areas.

The solution:

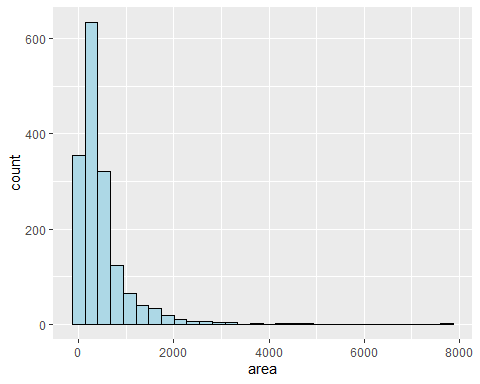
Creating 5-meter buffers around cycling path by using *st\_buffer()* and calculate the total area of the buffers by using *st\_area()*.

sf\_buffer\_cycling <- st\_buffer(sf\_cyclingpath,   
 dist=5, nQuadSegs = 30)  
sf\_buffer\_cycling$AREA <- st\_area(sf\_buffer\_cycling)  
sum(sf\_buffer\_cycling$AREA)

## 773143.9 [m^2]

Because the output is in tibble data.table format, you can plot the area easily by using *geom\_histogram()* of **ggplot2**.

sf\_buffer\_cycling$area <- as.numeric(sf\_buffer\_cycling$AREA)  
ggplot(sf\_buffer\_cycling, aes(x=area)) +  
 geom\_histogram(bins=30,   
 color="black",  
 fill="light blue")



### Point-in-polygon count

The scenario:

A pre-school services group want to find out numbers of pre-school in each Planning Subzone.

The solution:

The code chunk below first identify pre-schools located inside each Planning Subzone by using [st\_intersects()](https://r-spatial.github.io/sf/reference/geos_binary_pred.html). Then, the *length()* is used to calculate numbers of pre-school fall inside each planning subzone.

sf\_mpsz3414$`PreSch Count`<- lengths(st\_intersects(sf\_mpsz3414, sf\_preschool3414))

**Warning**: You should not confuse with [st\_intersection()](https://r-spatial.github.io/sf/reference/geos_binary_ops.html).

You can check the summary statistics of the newly derived *PreSch Count* field by using *summary()* as shown in the code chunk below.

summary(sf\_mpsz3414$`PreSch Count`)

## Min. 1st Qu. Median Mean 3rd Qu. Max.   
## 0.000 0.000 2.000 4.207 6.000 37.000

To list the planning subzone with the most number of of pre-school, the *top\_n()* of **dplyr** package is used as shown in the code chunk below.

top\_n(sf\_mpsz3414, 1, `PreSch Count`)

## Simple feature collection with 1 feature and 16 fields  
## Geometry type: MULTIPOLYGON  
## Dimension: XY  
## Bounding box: xmin: 23449.05 ymin: 46001.23 xmax: 25594.22 ymax: 47996.47  
## Projected CRS: SVY21 / Singapore TM  
## OBJECTID SUBZONE\_NO SUBZONE\_N SUBZONE\_C CA\_IND PLN\_AREA\_N PLN\_AREA\_C  
## 1 290 3 WOODLANDS EAST WDSZ03 N WOODLANDS WD  
## REGION\_N REGION\_C INC\_CRC FMEL\_UPD\_D X\_ADDR Y\_ADDR  
## 1 NORTH REGION NR C90769E43EE6B0F2 2014-12-05 24506.64 46991.63  
## SHAPE\_Leng SHAPE\_Area geometry PreSch Count  
## 1 6603.608 2553464 MULTIPOLYGON (((24786.75 46... 37

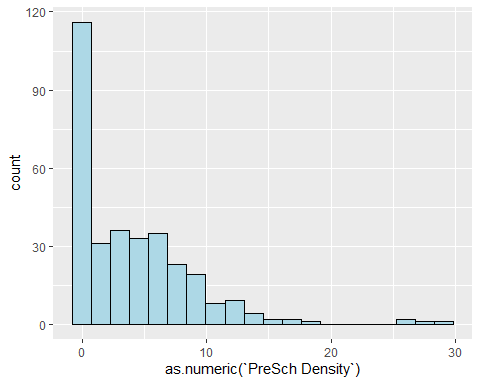
**Quiz: Calculate the density of pre-school by planning subzone. With the help of appropriate graphical method, describe the distribution of the newly derived variable.**

The code chunk below uses *st\_area()* of sf package to derive the area of each planning subzone.

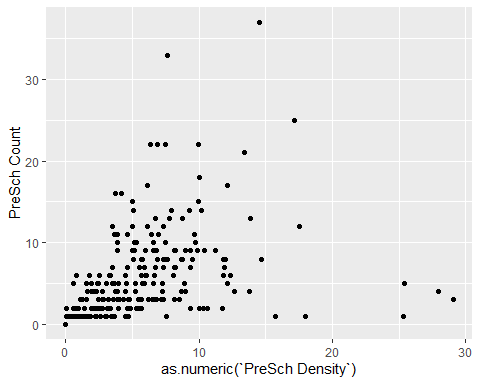
sf\_mpsz3414$Area <- sf\_mpsz3414 %>%  
 st\_area()

sf\_mpsz3414 <- sf\_mpsz3414 %>%  
 mutate(`PreSch Density` = `PreSch Count`/Area \* 1000000)

ggplot(data=sf\_mpsz3414,   
 aes(x= as.numeric(`PreSch Density`)))+  
 geom\_histogram(bins=20,   
 color="black", fill="light blue")



ggplot(data=sf\_mpsz3414,   
 aes(y = `PreSch Count`, x= as.numeric(`PreSch Density`)))+  
 geom\_point(color="black", fill="light blue")



# Working with **sp**, **gdal** and **rgeos** Packages (Optional)

In this section, you will learn how to handle geospatial data in shapefile format using sp, gdal and rgeos packages in R.

## Importing a shapefile

In this section, you will learn how to import MP14\_SUBZONE\_WEB\_PL GIS layer into R. It is stored in shapefile format. The spatial data model of this GIS data are polygon objects.

To import the GIS data layer into R, *readOGR()* from **rgdal** package will be used.

The data importing task is performed by using the code chunk below:

mpsz\_sp <- readOGR(dsn = "data/geospatial",   
 layer = "MP14\_SUBZONE\_WEB\_PL")

## OGR data source with driver: ESRI Shapefile   
## Source: "D:\tskam\GeoDSA\Hands-on\_Ex\Hands-on\_Ex02\data\geospatial", layer: "MP14\_SUBZONE\_WEB\_PL"  
## with 323 features  
## It has 15 fields

Notice that *mpsz\_sp* is in **SpatialPolygonDataFrame**.

### Checking the contents of a SpatialPolygonDataFrame

You can check the contents of *mpsz\_sp* data object by using *summary()*.

The code chunk:

summary(mpsz\_sp)

## Object of class SpatialPolygonsDataFrame  
## Coordinates:  
## min max  
## x 2667.538 56396.44  
## y 15748.721 50256.33  
## Is projected: TRUE   
## proj4string :  
## [+proj=tmerc +lat\_0=1.36666666666667 +lon\_0=103.833333333333 +k=1  
## +x\_0=28001.642 +y\_0=38744.572 +datum=WGS84 +units=m +no\_defs]  
## Data attributes:  
## OBJECTID SUBZONE\_NO SUBZONE\_N SUBZONE\_C   
## Min. : 1.0 Min. : 1.000 Length:323 Length:323   
## 1st Qu.: 81.5 1st Qu.: 2.000 Class :character Class :character   
## Median :162.0 Median : 4.000 Mode :character Mode :character   
## Mean :162.0 Mean : 4.625   
## 3rd Qu.:242.5 3rd Qu.: 6.500   
## Max. :323.0 Max. :17.000   
## CA\_IND PLN\_AREA\_N PLN\_AREA\_C REGION\_N   
## Length:323 Length:323 Length:323 Length:323   
## Class :character Class :character Class :character Class :character   
## Mode :character Mode :character Mode :character Mode :character   
##   
##   
##   
## REGION\_C INC\_CRC FMEL\_UPD\_D X\_ADDR   
## Length:323 Length:323 Length:323 Min. : 5093   
## Class :character Class :character Class :character 1st Qu.:21864   
## Mode :character Mode :character Mode :character Median :28465   
## Mean :27257   
## 3rd Qu.:31674   
## Max. :50425   
## Y\_ADDR SHAPE\_Leng SHAPE\_Area   
## Min. :19579 Min. : 871.5 Min. : 39438   
## 1st Qu.:31776 1st Qu.: 3709.6 1st Qu.: 628261   
## Median :35113 Median : 5211.9 Median : 1229894   
## Mean :36106 Mean : 6524.4 Mean : 2420882   
## 3rd Qu.:39869 3rd Qu.: 6942.6 3rd Qu.: 2106483   
## Max. :49553 Max. :68083.9 Max. :69748299

Let’s view the first few records in the *mpsz\_sp*.

The code chunk

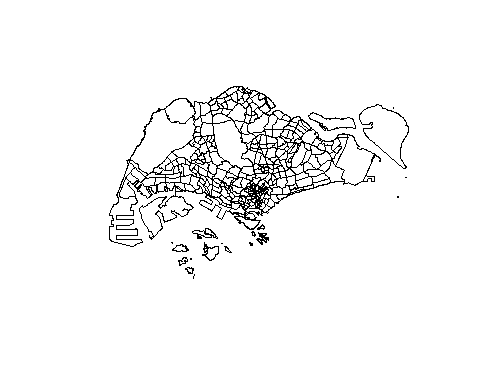
head(mpsz\_sp, n=4)

### Plotting the sptial data

To view the spatial data, *plot()* of **Base** R can be used.

The code chunk:

plot(mpsz\_sp)



## Now It’s Your Turn

Using the functions you had learned, import the Pre-School and Cycling Path GIS data files into R spatial objects.

The solution:

The pre-schools GIS data is in kml format. Before we can import the data file into R, we will use ogrListLayers function of rgdal package to check the actual data structure of the kml data file.

ogrListLayers("data/geospatial/pre-schools-location-kml.kml")

## [1] "PRESCHOOLS\_LOCATION"  
## attr(,"driver")  
## [1] "KML"  
## attr(,"nlayers")  
## [1] 1

### Importing kml GIS data

Notice that the file called pre-schools-location-kml is just the folder (refer to the list above). In order to important the layer, we need to use PRESCHOOL\_LOCATION layer instead.

The code chunk below will do the trick.

preschool <- readOGR("data/geospatial/pre-schools-location-kml.kml",  
 "PRESCHOOLS\_LOCATION")

## OGR data source with driver: KML   
## Source: "D:\tskam\GeoDSA\Hands-on\_Ex\Hands-on\_Ex02\data\geospatial\pre-schools-location-kml.kml", layer: "PRESCHOOLS\_LOCATION"  
## with 1359 features  
## It has 2 fields

### Importing GIS data layer from LTADataMall

In this section, you will learn how to import a line geospatial data into R. The geospatial data is the CyclingPath shapefile from LTA DataMall (<https://www.mytransport.sg/content/mytransport/home/dataMall.html>)

cyclingpath <- readOGR (dsn = "data/geospatial",   
 layer = "CyclingPath")

## OGR data source with driver: ESRI Shapefile   
## Source: "D:\tskam\GeoDSA\Hands-on\_Ex\Hands-on\_Ex02\data\geospatial", layer: "CyclingPath"  
## with 1625 features  
## It has 2 fields

* Show the codes you used to check the contents of *preschool* and *cyclingpath* spatial objects.
* Describe their spatial data models, boundary coordinates, projection, and attribute variables.

### Assigning a coordinate system

* Use *CRS* and *spTransform* functions of **rgdal**

mpsz\_svy21 <- spTransform(mpsz\_sp,   
 CRS("+init=epsg:3414"))

## Reprojecting a geospatial data

Now, it is your turn to change the projection system of the *preschool* data set from wgs84 to svy21.

The solution:

preschool\_svy21 <- spTransform(preschool,   
 CRS("+init=epsg:3414"))

## Geoprocessing with **rgeos**

The scenario

The authority is planning to upgrade the exiting cycling path. To do so, they need to acquire 5 metres reserve land on the both sides of the current cycling path. You are tasked to determine the extend of the land need to be acquired and their total areas.

The solution:

buf\_cyclingpath <- gBuffer(cyclingpath, width = 5)

The solution:

buf\_cyclingpath <- gBuffer(cyclingpath, byid = TRUE,   
 width = 5)

The solution:

buf\_cyclingpath@data$Area <- gArea(buf\_cyclingpath,   
 byid = TRUE)  
sum(buf\_cyclingpath@data$Area)

## [1] 771024.9