Initial Experimentation

Aaron Brown

Saturday, May 16, 2015

Contents

1 Introduction

This introduction was taken as part of the Spatial.ly: R Spatial Tips site. The site was built to give R users an introduction into spatial analysis using R. ##Packages Required This analysis uses several packages in the analysis.

1. **ProjectTemplate** library to manage the files and structure of the project.
2. **maptools** and **rgdal** as required on this page.

This project is loaded by executing the following codechunk.

```
library(knitr)
opts_knit$set(root.dir = '..')
```

```
library(ProjectTemplate)
load.project()
```

```
## Loading project configuration
## Autoloading helper functions
## Autoloading packages
##  Loading package: maptools
## Loading required package: maptools
## Loading required package: sp
## Checking rgeos availability: FALSE
##      Note: when rgeos is not available, polygon geometry     computations in maptools depend on gpcli
##      which has a restricted licence. It is disabled by default;
##      to enable gpclib, type gpclibPermit()
##  Loading package: rgdal
## Loading required package: rgdal
## rgdal: version: 0.9-1, (SVN revision 518)
## Geospatial Data Abstraction Library extensions to R successfully loaded
## Loaded GDAL runtime: GDAL 1.11.0, released 2014/04/16
## Path to GDAL shared files: C:/Users/abrow/Documents/R/projects/personal/Spatial.ly Proj/packrat/lib/i
## GDAL does not use iconv for recoding strings.
## Loaded PROJ.4 runtime: Rel. 4.8.0, 6 March 2012, [PJ_VERSION: 480]
## Path to PROJ.4 shared files: C:/Users/abrow/Documents/R/projects/personal/Spatial.ly Proj/packrat/lib
##  Loading package: downloader
## Loading required package: downloader
## Autoloading cache
## Autoloading data
## Munging data
```

1.1 Files downloaded

The London Sports Participate shapefile and London Cycle Hire Locations are downloaded to the *./data* directory.

```
if(!file.exists('./data/01/London_Sport1.zip')){
      url <- "http://spatialanalysis.co.uk/wp-content/uploads/2010/09/London_Sport1.zip"
      download(url, dest = "data/01/London_Sport1.zip", mode = "wb")
}
```

```
unzip("data/01/London_Sport1.zip", overwrite = TRUE, exdir = "data/01")

if(!file.exists('./data/01/London_cycle_hire_locs.csv')){
      url <- "http://dl.dropbox.com/u/10640416/London_cycle_hire_locs.csv"
      download(url, dest = "data/01/London_cycle_hire_locs.csv", mode = "wb")
}
```

1.2 Exploratory Analysis

We can read the London Cycle Hire data into the **cycle** variable with the following code.

```
cycle<- read.csv("data/01/London_cycle_hire_locs.csv", header=T)
```

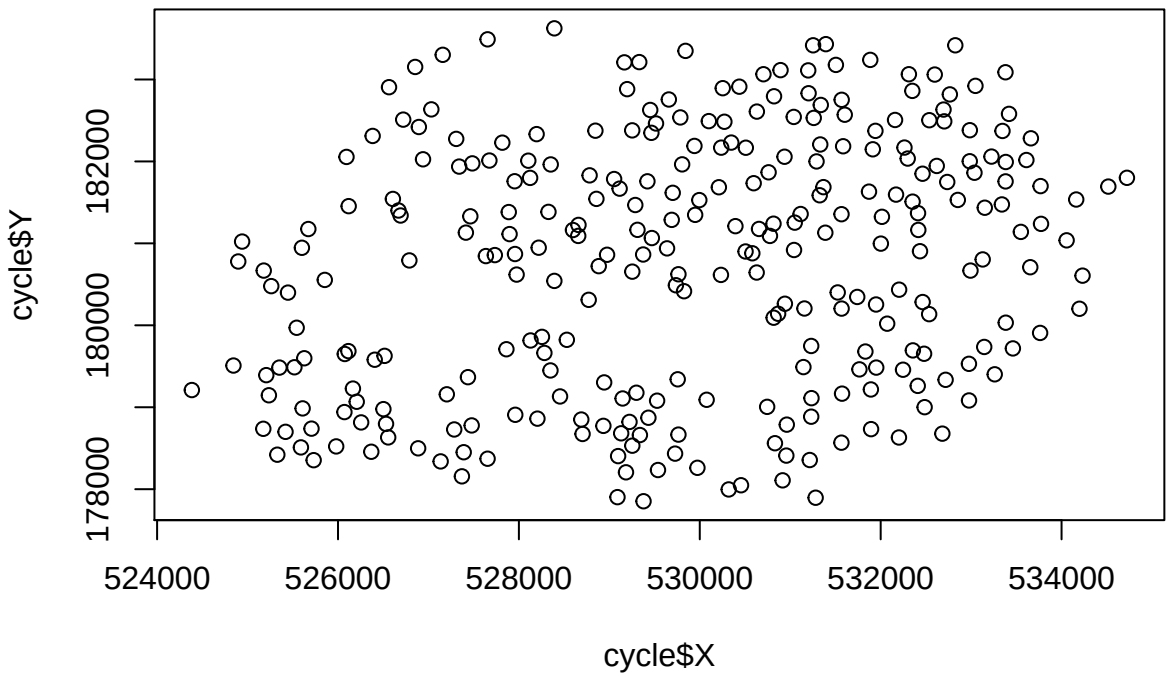
In the following code chunk, we can see what the column data looks like

```
head(cycle)
```

```
##                          Name      Village      X      Y Capacity
## 1 Kensington Olympia Station      Olympia 524384 179210       25
## 2            Ilchester Place   Kensington 524844 179509       24
## 3            Chepstow Villas Notting Hill 524897 180779       17
## 4           Turquoise Island Notting Hill 524940 181022       21
## 5         West Cromwell Road Earl's Court 525174 178737       24
## 6           Pembridge Villas Notting Hill 525179 180668       16
```

In order to illustrate the difference between plotting spatially and as a normal XY-plot we have the following figure which can be compared with a later figure.

```
plot(cycle$X, cycle$Y)
```



We can tell R that **cycle** contains spatial data with the *coordinates* function

```
coordinates(cycle)<- c("X", "Y")
```

A look at the cycle data.

```
class(cycle)
```

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

```
str(cycle)
```

```
## Formal class 'SpatialPointsDataFrame' [package "sp"] with 5 slots
##   ..@ data       :'data.frame':  292 obs. of  3 variables:
##   .. ..$ Name    : Factor w/ 292 levels "Abbey Orchard Street",..: 144 135 51 261 282 199 202 2 183 1
##   .. ..$ Village : Factor w/ 64 levels "Aldgate","Angel",..: 37 26 36 36 14 36 26 26 36 14 ...
##   .. ..$ Capacity: int [1:292] 25 24 17 21 24 16 37 17 16 18 ...
##   ..@ coords.nrs : int [1:2] 3 4
##   ..@ coords     : num [1:292, 1:2] 524384 524844 524897 524940 525174 ...
##   .. ..- attr(*, "dimnames")=List of 2
##   .. .. ..$ : NULL
##   .. .. ..$ : chr [1:2] "X" "Y"
##   ..@ bbox       : num [1:2, 1:2] 524384 177853 534723 183624
```

```
##   .. ..- attr(*, "dimnames")=List of 2
##   .. .. ..$ : chr [1:2] "X" "Y"
##   .. .. ..$ : chr [1:2] "min" "max"
##   ..@ proj4string:Formal class 'CRS' [package "sp"] with 1 slot
##   .. .. ..@ projargs: chr NA
```

The class has become a "SpatialPointsDataFrame" which is a type of S4 object that requires handling slightly differently. The str() output contains lots of @ symbols which denote a different slot. Typing cycle@data will extract the attribute data. The X and Y locational information can now be found in the coords slot. In addition bbox contains the bounding box coordinates and the pro4string contains the projection, or CRS (coordinate reference system) information. We have not specified this so it is empty at the moment. We therefore need to refer to the correct Proj4 string information. These are loaded with the rgdal package and can simply be referred to with an ID. To see the available CRSs enter:

```
EPSG<- make_EPSG()
head(EPSG)
```

```
##   code                                                note
## 1 3819                                             # HD1909
## 2 3821                                              # TWD67
## 3 3824                                              # TWD97
## 4 3889                                               # IGRS
## 5 3906                                            # MGI 1901
## 6 4001 # Unknown datum based upon the Airy 1830 ellipsoid
##                                                                                  prj4
## 1 +proj=longlat +ellps=bessel +towgs84=595.48,121.69,515.35,4.115,-2.9383,0.853,-3.408 +no_defs
## 2                                                          +proj=longlat +ellps=aust_SA +no_defs
## 3                                    +proj=longlat +ellps=GRS80 +towgs84=0,0,0,0,0,0,0 +no_defs
## 4                                    +proj=longlat +ellps=GRS80 +towgs84=0,0,0,0,0,0,0 +no_defs
## 5                          +proj=longlat +ellps=bessel +towgs84=682,-203,480,0,0,0,0 +no_defs
## 6                                                    +proj=longlat +ellps=airy +no_defs
```

In this case the data are in British National Grid. We can search for this within the EPSG object.

```
with(EPSG, EPSG[grep("British National", note),])
```

```
##       code                                note
## 3424 27700 # OSGB 1936 / British National Grid
##
## 3424 +proj=tmerc +lat_0=49 +lon_0=-2 +k=0.9996012717 +x_0=400000 +y_0=-100000 +datum=OSGB36 +units=m
```

The code we are after is 27700.

```
BNG<- CRS("+init=epsg:27700")

proj4string(cycle)<-BNG
```

From this point we can combine the data with other spatial information and also perform transformations on the data. This is especially useful when exporting to other software. See the XXX tutorial.

Shapefiles are extremely simple to import/ export to/from an R session and are handled as spatial objects in the same way as above. In this case we are going to load in a SpatialPolygonsDataframe. We can specify the CRS when the data at this stage (it is BNG as above).

```
sport<- readShapePoly("data/01/london_sport.shp", proj4string= BNG)
```

A look at the attribute table headings (these are the values stored in the @data slot)

```
names(sport)
```

```
## [1] "ons_label"  "name"       "Partic_Per" "Pop_2001"
```

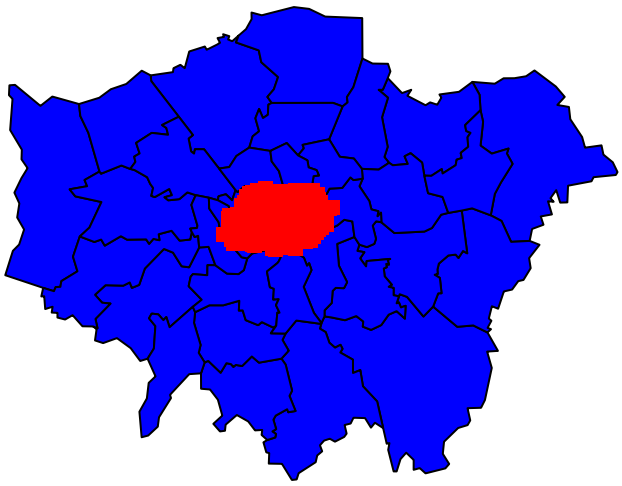
and you can double check the CRS:

```
sport@proj4string
```

```
## CRS arguments:
##  +init=epsg:27700 +proj=tmerc +lat_0=49 +lon_0=-2 +k=0.9996012717
## +x_0=400000 +y_0=-100000 +datum=OSGB36 +units=m +no_defs
## +ellps=airy
## +towgs84=446.448,-125.157,542.060,0.1502,0.2470,0.8421,-20.4894
```

Plot the data and add the cycle points to check they line up (they should be around the centre of London).

```
plot(sport, col="blue")
plot(cycle, add=T, col= "red", pch=15)
```



2 Further Explorations

Refer to other tutorials if you want to produce interesting maps with these data.

To export the data as a shapefile use the following syntax for the point data:

```
writePointsShape(cycle, "data/01/processed/cycle.shp")
```

and the polygon data (replace Poly with Lines for spatial lines)

```
writePolyShape(sport, "data/01/processed/london_sport.shp")
```

Done!