R Spatial Reference Card

Robin Lovelace and Rachel Oldroyd

R has a suite of powerful and effective spatial data processing tools. This reference card to summarises the fundamentals of R for geographical applications, in terms of packages and functions. The code has been tested on data from the ‘Creating-maps-in-R’ GitHub repository. Run the commands from the unzipped folder to see the commands in action!

Using the example of the `help` command, type: `?help` (for information on a specific function), `??help` (to search for word in R’s documentation) and `example(help)` (for pre-made examples of that function).

Key packages

Additional packages must be installed (using `install.packages(’packageName’)`) on top of R’s default packages to utilise R’s spatial capabilities. Packages are loaded using `library(packageName)`. For example, will load the **rgdal** package which is R’s interface to the ‘Geospatial Abstraction Library’ (GDAL). The most important and frequently used *general purpose* spatial packages are:

- **sp**: basis of most other spatial R packages - provides Spatial* classes
- **rgdal**: R’s interface to the GDAL data import library
- **rgeos**: provides a number of spatial functions
- **maptools**: tools for handling spatial objects

raster and **spatstat** are mature and extremely useful spatial packages, providing functions for raster and point-pattern applications, respectively. There are hundreds of additional R packages described on the CRAN website in the *spatial view*.

R’s Spatial* classes

To work with spatial data, R uses *classes* that are more complex than the default data classes in R’s base package; *S3* `vector`, `list` and `data.frame` s. Spatial object classes can be identified with the function `class(objectName)`. Each contains a number of different data *slots*: sub-classes of specific types. Some important spatial classes provided by the `sp` package include:

- `SpatialPoints`: point data, each point represented by an x and y coordinate
- `SpatialLines`: line data, composed of lists of the `Lines` sub-class or *slot*
- `SpatialPolygons`: polygon data, with `Polygons` and `Polygon` *slots* dewwed
- `SpatialPixels`: a spatial class for raster data comprised of pixels

It is important to note that each of the above classes cannot contain non-spatial attribute data (like attribute tables in GIS softward like QGIS). Therefore, they must first be converted into a class of the same name but suffixed with `DataFrame`. Thus `london <- SpatialPointsDataFrame(sp, df)` will create a new object containing points with spatial and non-spatial information called london. Likewise `SlDf <- SpatialLinesDataFrame` will create a new object containing lines with spatial and non-spatial information.

The attribute data of the london object can be accessed in this instance by using `spPdf@data`. The `@data` notation refers to the `data` slot of the new object. More fundamental spatial classes, which are in fact subsets of the `Spatial*` type classes listed above, include `bbox` (the object’s bounding box) and `proj4string` (the projection of the object, which may be `NA`).

Loading spatial data `readOGR` from **rgdal** can load a wide variety of spatial data types. The following line loads spatial data representing London:

```
london <- readOGR("data/", "london_sport")
```

Plot the london object using the geometry slot:

plot(london)

Analysing Attribute data

Print the column headings of the london object:

```
names(london)
```

Print the entire london object:

```
print(london)
```

Print a defined number of rows from the london object (2 in this case):

```
head(london@data, n=2)
```

Some basic column statistics:

```
mean(london$Partic_Per) #calculate the mean value of the Partic_Per column
max(london$Partic_Per)  #calculate the maximum value
min(london$Partic_Per)  #calculate the minimum value
sum(london$Partic_Per)  #Calculate the sum of the values

nrow(london)  #Return the number of rows in the dataset
ncol(london)  #Return the number of columns in the dataset
```

Additional information about the london object:

```
summary(london)
```

Allocating and changing projection

Before undertaking any further analysis it is useful to know the Coordinate Reference System(CRS) of the london object.

Query the current projection of an object:

```
proj4string(london)
```

Change the Coordinate Reference System (CRS) if it has been incorrectly assigned:

Change the Coordinate Reference System (CRS) if it has been incorrectly assigned:

```
proj4string(london) <-CRS("+init=epsg:27700")
```

Note the warning above which states that the coordinate reference system has been changed but the data has not been transformed. To transform the data use:

```
london.wgs84 <-spTransform(london, CRS("+init=epsg:4326"))
```

Attribute Joins

Attribute joins are used to link additional pieces of information to pre-existing polygons. In the london object, for example, there are 5 attribute variables (you saw these earlier when typing `names(london)`)

Spatial subsetting

Spatial aggregation

Spatial graphics

Basemaps and advanced functions