Spatial Data Analysis - Practical 1

Jumping Rivers

Question 1 - reading in data

First let's load the required libraries

```
library("sf")
library("jrSpatial")
```

We're going to get you to read in some data, but first you need some data to read in! Running the following function from the **jrSpatial** package will copy a folder into your current working directory containing shape files

```
get_ukgeom()
```

1. Using `st_read()`, read the shape file in to R
2. Plot the data
3. Plot the data, but this time only showing the plots for the population in 2015 and 2017

*Question 2a - cooking with **sf***

sf objects are built very differently to normal R objects, such as a vector or tibble. So in this question we're going to get you to build up a couple of them from scratch.

We'll go through the 7 types of simple features objects. Have a go at running the code for each object, then at the end we'll give you a few questions.

1. Point. You can make a **sf** point using `st_point()`

   ```
   p = st_point(c(13, 15))
   plot(p)
   ```

2. Multipoint. This is one **sf** object that contains multiple points. Before converting it to a multipoint object the points need to be a matrix. Where each row is a point

   ```
   ps = rbind(c(10, 10), c(10, 9), c(11, 9), c(11, 10))
   mp = st_multipoint(ps)
   plot(mp)
   ```

3. Linestring. This is several points joined up to make a line. We can convert a multipoint to a linestring easily using `st_linestring()`

```
l = st_linestring(mp)
plot(l)
```

4. Multilinestring. This is one **sf** object that contains multiple separate lines. Use `sf_multilinestring()` to create these. Notice the lines have to be in a list

```
ps2 = rbind(c(11, 11), c(12, 11))
mls = st_multilinestring(list(ps, ps2))
plot(mls)
```

5. Polygon. This is a shape. On a map you'd think of this as a country or island. Use `st_polygon()` to create a polygon. Again, in a list!

```
pol_points = rbind(c(10, 10), c(10, 9), c(11, 9), c(11, 10), c(10, 10))
pol = st_polygon(list(pol_points))
plot(pol)
```

6. Multipolygon. This is one **sf** object containing multiple polygons. The function used is `st_multipolygon()` and each polygon is within it's own list, inside another list.

```
pol_points2 = rbind(c(12, 12), c(13, 13), c(14, 12), c(12, 12))
mpol = st_multipolygon(list(list(pol_points), list(pol_points2)))
plot(mpol)
```

7. Geometry collection. This is an **sf** object that can be made up of all the previous 6 objects, made with `st_geometrycollection()`

```
gc = st_geometrycollection(list(p, mpol, mls))
plot(gc)
```

*Question 2b - cooking with **sf***

1) Hurricane Shary hit the Americas in October 2010. We've managed to attain the approximate longitude and latitude for the storm every 6 hours it was active. To get you started, I've created the matrix of longitudes and latitudes. Turn the matrix into a **sf** linestring and plot it

```
longitude = c(-61, -63, -65, -66, -66)
latitude = c(26, 27, 28, 29, 30)
shary_matrix = cbind(longitude, latitude)
```

Longitude	Latitude
-64	31
-61	33
-57	35

2) 12 hours after Shary was declassified to a tropical storm, it was upgraded again to a hurricane. You can see this data the table. Create a matrix called `shary_matrix2`, using code similar to that of question 1. Combine the two matrices `shary_matrix` and `shary_matrix2` into a multilinestring

3) Hurrican Colin hit America in August of 2010. Below is the code to turn storm Colin longitude and latitude into a **sf** linestring

```
longitude = c(-65, -66, -67, -67, -66)
latitude = c(23, 24, 25, 26, 27)
colin_matrix = cbind(longitude, latitude)
colin = st_linestring(colin_matrix)
```

4) Ideally we'd like to keep all of these inside of a data.frame with one column containing the geometry. Just like `uk_shape`. We can do this using `st_sf()` and `st_sfc()`

```
hurricane = st_sf(name = c("Shary", "Colin"), geometry = st_sfc(shary, colin))
```

Try plotting `hurricane`, what happens?

Solutions

```
vignette("solutions1", package = "jrSpatial")
```