Topic 2: Using geographic maps in R

This topic shows how to use geographic maps in R with sp and sf objects. In particular, we consider

- Basic **geographic maps in R**.
- Library **maps**.
- **GADM database** of Global Administrative Areas.
- **Plotting sp objects** using the world map.
- **Plotting sf objects** using the world map.

Geographic maps in R

The **maps** library for R is a basic tool for creating maps of countries and regions of the world. For example, you can create a map of the USA and its states as:

```
> library(maps)
> map("state", interior = FALSE)
> map("state", boundary = FALSE,  col = "gray", add = TRUE)
```

The coordinate system of the plot is latitude and longitude, so it's easy to overlay other spatial data on this map.

Unfortunately, the data for the MAPS library isn't sufficient for some applications. Also in many R packages maps are fairly low-resolution, and political boundaries can be incomplete or out-of-date.

To produce high-quality maps, it is better to use free online resources where you can find up-to-date high-resolution map data for use with R.

GADM database of Global Administrative Areas

GADM is a spatial database of the location of the world's administrative areas (or administrative boundaries) for use in GIS and similar software.

Administrative areas in this database are countries and lower-level subdivisions, such as provinces, departments, counties, etc. GADM describes where these administrative areas are (the "spatial features"), and it provides some attributes for each area.

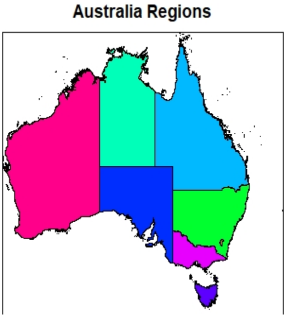
GADM goal is to map all administrative areas of all countries, at all levels, at any time period. They use a high spatial resolution and an extensive set of attributes for each spatial feature. The current version of GADM maps has 400,276 administrative areas.

GADM maps are available as native R objects that can be plotted directly with the SPPLOT function (from the SP package).

For example, here's how to load the data for Australia, and then plot each state with a random colour:

```
> library(sp)
> con<-gzcon(url(
+ "http://biogeo.ucdavis.edu/data/gadm2.8/rds/AUS_adm1.rds"))
> data <- readRDS(con)
```

```
> library(RColorBrewer)
> col <- rainbow(length(levels(data$NAME_1)))
> data$NAME_1 = as.factor(data$NAME_1)
> spplot(data, "NAME_1", col.regions=col,
+ main="Australia Regions", colorkey = FALSE, lwd=.4)
> col <- rainbow(length(levels(data$NAME_1)))
> spplot(data, "NAME_1", col.regions=col,
+ main="Australia Regions",  colorkey = FALSE, lwd=.4)
```



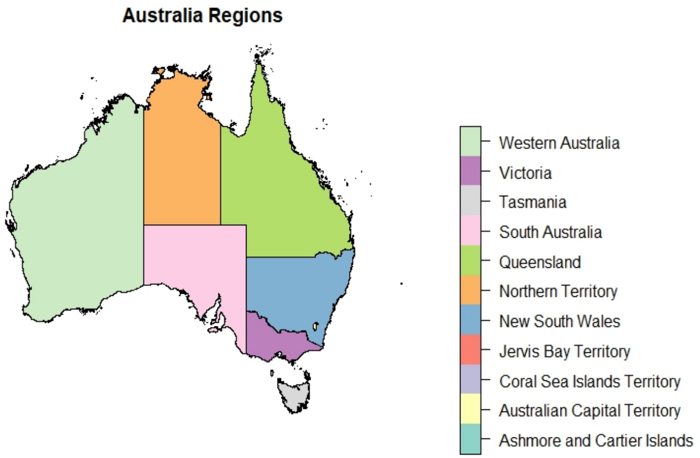
To use the Australia GADM maps in sf, first, download the data form the webpage

`https://geodata.ucdavis.edu/gadm/gadm4.1/gpkg/gadm41_AUS.gpkg`

or the LMS folder Data.
Then use the following commands to plot states (their information is in the layer ADM_ADM_1 and names are in NAME_1) in different colours. The parameters key.pos and key.width adjust the legend.

```
> data <- st_read("gadm41_AUS.gpkg", layer="ADM_ADM_1")
Reading layer ‘ADM_ADM_1’ from data source
using driver ‘GPKG’
Simple feature collection with 11 features and 11 fields
Geometry type: MULTIPOLYGON
Dimension:     XY
Bounding box:  xmin: 112.9211 ymin: -55.11694
xmax: 159.1092 ymax: -9.142176
Geodetic CRS:  WGS 84
```

```
> str(data)
Classes sf and 'data.frame': 11 obs. of  12 variables:
$ GID_1    : chr  "AUS.1_1" "AUS.2_1" "AUS.3_1" ...
$ GID_0    : chr  "AUS" "AUS" "AUS" "AUS" ...
$ COUNTRY  : chr  "Australia" "Australia" "Australia" ...
$ NAME_1   : chr  "Ashmore and Cartier Islands"  ...
$ VARNAME_1: chr  "NA" "NA" "NA" "NA" ...
$ NL_NAME_1: chr  "NA" "NA" "NA" "NA" ...
$ TYPE_1   : chr  "Territory" "Territory" "Territory"  ...
$ ENGTYPE_1: chr  "Territory" "Territory" "Territory"  ...
$ CC_1     : chr  "12" "8" "11" "10" ...
$ HASC_1   : chr  "AU.AS" "AU.AC" "AU.CR" "AU.JB" ...
$ ISO_1    : chr  "NA" "AU-ACT" "NA" "NA" ...
$ geom     :sfc_MULTIPOLYGON of length 11;
...
> plot(data["NAME_1"],key.pos = 4,  key.width = lcm(7),
+ main = "Australia Regions")
```


Australia Regions
Western Australia
Victoria
Tasmania
South Australia
Queensland
Northern Territory
New South Wales
Jervis Bay Territory
Coral Sea Islands Territory
Australian Capital Territory
Ashmore and Cartier Islands

Plotting sp/sf objects using world map

Example 1. Let us plot cities with the largest population on the world map. We will use the dataset worldcities.csv from the World Cities Database (`https://simplemaps.com/data/world-cities`) with information about large cities across the world. It is also available in the LMS folder Data.

```
> cities <- read.csv("worldcities.csv", header = TRUE)
> str(cities)
'data.frame':   12893 obs. of  11 variables:
$ city      : chr  "Malishev" "Prizren" "Zubin Potok" ...
$ city_ascii: chr  "Malisheve" "Prizren" "Zubin Potok"  ...
$ lat       : num  42.5 42.2 42.9 42.6 42.3 ...
$ lng       : num  20.7 20.7 20.7 21.6 21.4 ...
$ country   : chr  "Kosovo" "Kosovo" "Kosovo" "Kosovo" ...
$ iso2      : chr  "XK" "XK" "XK" "XK" ...
$ admin_name: chr  "Malishev" "Prizren" "Zubin Potok"  ...
$ capital   : chr  "admin" "admin" "admin" "admin" ...
$ population: num  NA NA NA NA NA NA NA NA NA NA ...
$ id        : int  1901597212 1901360309 1901608808   ...
```

We consider only the cities with a population of more than 10000000:

```
> cities <-cities[complete.cases(cities), ]
> sum(cities$population > 10000000)
[1] 19
> mcities <- cities[cities$population > 10000000, ]
> mcities <- mcities[, c("lng","lat", "population")]
```

```
> mcities<-mcities[, c("lng","lat", "population")]
> mcities
           lng      lat population
1071   -99.1310  19.4424   19028000
1683   120.9822  14.6042   11100000
1786    66.9900  24.8700   12130000
2216    37.6155  55.7522   10452000
3390    29.0100  41.1050   10061000
4218   -58.3975 -34.6025   12795000
4530    90.4086  23.7231   12797394
5045   -43.2250 -22.9250   11748000
5097   -46.6250 -23.5587   18845000
5736   121.4365  31.2165   14987000
5960   116.3883  39.9289   11106000
6538    31.2500  30.0500   11893000
7441    72.8570  19.0170   18978000
7468    88.3247  22.4950   14787000
7512    77.2300  28.6700   15926000
7812   139.7514  35.6850   35676000
7849   135.4601  34.7500   11294000
9739   -73.9249  40.6943   19164071
10410 -118.4068  34.1140   12740381
```

We rescale the population in 10000000 and create SpatialPointsDataFrame using cities' locations and rescaled population:

```
> mcities$population <- mcities$population/10000000
> mcities_coor <- cbind(mcities$lng, mcities$lat)
> row.names(mcities_coor) <-  1:nrow(mcities_coor)
> row.names(mcities) <-  1:nrow(mcities)
> str(mcities_coor)
> llCRS <- CRS("+proj=longlat +ellps=WGS84")
> mcities_sp <- SpatialPoints(mcities_coor, proj4string = llCRS)
> summary(mcities_sp)
```

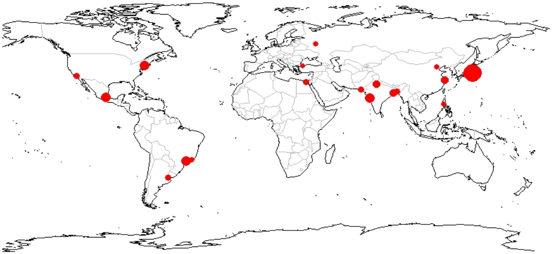
```
> mcities_sp <- SpatialPoints(mcities_coor, proj4string = llCRS)
> summary(mcities_sp)
Object of class SpatialPoints
Coordinates:
               min      max
coords.x1 -118.4068 139.7514
coords.x2  -34.6025  55.7522
Is projected: FALSE
proj4string : [+proj=longlat +ellps=WGS84 +no_defs]
Number of points: 19
```

Then we create a spatial data frame by using

```
> mcities_spdf <- SpatialPointsDataFrame(mcities_coor,
+ mcities,proj4string = llCRS, match.ID = TRUE)
> summary(mcities_spdf)
Object of class SpatialPointsDataFrame
Coordinates:
min       max
coords.x1 -118.4068 139.7514
coords.x2  -34.6025  55.7522
Is projected: FALSE
proj4string : [+proj=longlat +ellps=WGS84]
Number of points: 19
Data attributes:
lng              lat             population
Min.   :-118.41   Min.   :-34.60   Min.   :1.006
...
```

Finally, we add the cities to the world map using their populations as the graphical parameter cex, that controls the sizes of the circled on the map.

```
> library(maps)
> map("world", interior = FALSE)
> map("world", boundary = FALSE,  col = "gray", add = TRUE)
> plot(mcities_spdf,  pch = 16, col = "red",
+ cex = (mcities_spdf$population), add = TRUE)
```



Example 2.

In this example we visualise the cities by using the sf package.
After transforming the data to the sf format it is straightforward to plot them, by using the MAPVIEW command:

```
> mcities_sf <-  st_as_sf(mcities_spdf)
> str(mcities_sf)
Classes sf and 'data.frame': 19 obs. of  4 variables:
$ lng       : num  -99.1 121 67 37.6 29 ...
$ lat       : num  19.4 14.6 24.9 55.8 41.1 ...
$ population: num  1.9 1.11 1.21 1.05 1.01 ...
$ geometry  :sfc_POINT of length 19; first list element:
'XY' num  -99.1 19.4
- attr(*, "sf_column")= chr "geometry"
- attr(*, "agr")= Factor w/ 3 levels "constant","aggregate",..:
 NA NA NA
..- attr(*, "names")= chr [1:3] "lng" "lat" "population"
> mapview(mcities_sf, col.regions = "red",  cex = "population",
+ fgb = FALSE)
```


mcities_sf
NORTH AMERICA
SOUTH AMERICA
AFRICA
ASIA
OCEANIA
Pacific Ocean
Indian Ocean
3000 km
2000 mi
mcities_sf
Leaflet | © OpenStreetMap contributors © CARTO

Now, let us draw straight lines connecting the first 6 cities. First, we create a linestring object from the cities' coordinates. Then, we add the coordinate reference system to it and plot the result:

```
> lnstr_sfg2 <- st_linestring(as.matrix(mcities_sf[1:6,
+ c("lng", "lat"), drop = TRUE]))
> lnstr_sfc <- st_sfc(lnstr_sfg2,   crs = 4326)
> mapview(lnstr_sfc, alpha.regions = 0, color = "red",
+ lwd = 2, fgb = FALSE)
```

Key R commands

st_multipoint(...)	*creates the multipoint simple feature*
st_sfc(...)	*creates simple feature geometry list columns*
st_sf(...)	*creates sf objects*
st_polygon(...)	*creates the polygon simple feature*
st_as_sf(...)	*converts external objects to an sf object*
st_crs(...)	*sets or retrieves coordinate reference system*
as(...)	*converts sf to a Spatial* object*
mapview(...)	*produces an interactive map with a spatial object*
map(...)	*draws geographic maps*
st_read(...)	*reads simple features or layers from a file*
rainbow(...)	*create a vector of n colours*
row.names(...)	*gets and sets row names*
st_linestring(...)	*creates the linestring simple feature*