# R Fridays GIS Tutorial

*Danielle Clake*

*12 March 2018*

```
setwd("C:/User/Documents/R Fridays/GIS Data in R")
```

## Spatial Data Types

The two major types of GIS data are *vector* format (a set of spatially referenced points, lines, or polygons that are each assigned one or more attributes) and *raster* format (a continuous grid of pixels or "cells" which each have an assigned value - think of a digital image as an example, where the assigned value would be a specific colour for each pixel). We will use both in this tutorial.

## Install and Load Packages

This code will install and load the packages used in this tutorial. If packages are already installed and/or loaded users should skip the code below. Prompts to load packages will also be provided in text before the first time each package is used.

```
x <- c("raster", "rgdal", "rgeos", "tmap")
```

Uncomment the code below if you need to install/load packages:

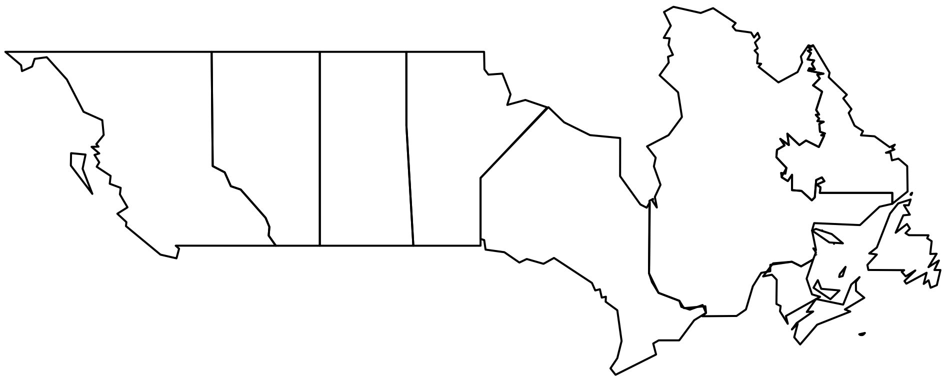
```
#install.packages(x)
lapply(x, library, character.only = TRUE)
```

## Vector Data

### Load data and view structure

We will be using the **rgal** package to load our spatial data in R. This package has a function *readOGR* which requires two inputs: dsn (the "data source name") and the name of the layer. Note that for this function we do not need to include a file extension after the data name.

```
library(rgdal)
PROVINCE <- readOGR(dsn = "Data/shapefiles", layer = "Prov_Boundary_GeoGratis")
plot(PROVINCE) #(sorry territories - lots of small islands made it slow to plot)
```



The information for this spatial data is stored in "slots":

```
#str(PROVINCE)
slotNames(PROVINCE)
```

```
## [1] "data"        "polygons"    "plotOrder"   "bbox"        "proj4string"
```

We can use the "@data" slot to see the attribute table for our data. Using the "$" will automatically query for that column name in the data.

```
PROVINCE@data
```

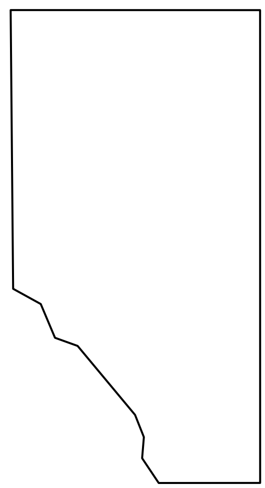
```
##   UUID TYPE_E                      NAME SRC_AGENCY
## 0   86   PROV          BRITISH COLUMBIA      NRCAN
## 1  509   PROV NEWFOUNDLAND AND LABRADOR      NRCAN
## 2   79   PROV              SASKATCHEWAN      NRCAN
## 3  490   PROV      PRINCE EDWARD ISLAND      NRCAN
## 4  497   PROV                   ONTARIO      NRCAN
## 5  534   PROV               NOVA SCOTIA      NRCAN
## 6  466   PROV                    QUEBEC      NRCAN
## 7   37   PROV                   ALBERTA      NRCAN
## 8  496   PROV                  MANITOBA      NRCAN
## 9  489   PROV             NEW BRUNSWICK      NRCAN
```

```
PROVINCE$NAME
```

```
##  [1] BRITISH COLUMBIA          NEWFOUNDLAND AND LABRADOR
##  [3] SASKATCHEWAN              PRINCE EDWARD ISLAND
##  [5] ONTARIO                   NOVA SCOTIA
```

```
##  [7] QUEBEC                    ALBERTA
##  [9] MANITOBA                  NEW BRUNSWICK
## 10 Levels: ALBERTA BRITISH COLUMBIA MANITOBA ... SASKATCHEWAN
```

```r
#To create a new shapefile that only includes Alberta:
AB <- PROVINCE[PROVINCE$NAME == "ALBERTA",]
plot(AB)
```



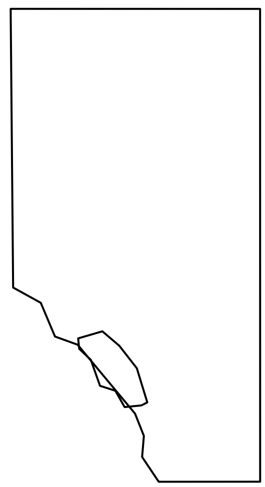
The "@proj4string" slot tells us the coordinate reference system (CRS) for this data.

```r
AB@proj4string
```

```
## CRS arguments:
##  +proj=longlat +datum=NAD83 +no_defs +ellps=GRS80 +towgs84=0,0,0
```

I won't go into too much detail about CRS and projections today, but we do need to make sure that we are using the same projection. The province layer is in a geographic coordinate system using lat/long and decimal degrees as units.

```r
#If we add data that is in a different CRS (e.g. UTM Zone 12N), it will not line up properly
RSA_U12 <- readOGR(dsn = "Data/shapefiles", layer = "RSA_UTM12")
RSA_U12@proj4string
#plot(RSA_U12, add = TRUE)

#We can reproject it to match our province CRS as follows:
RSA <- spTransform(RSA_U12, proj4string(AB))
plot(AB)
plot(RSA, add = TRUE)
```



### Add sampling points

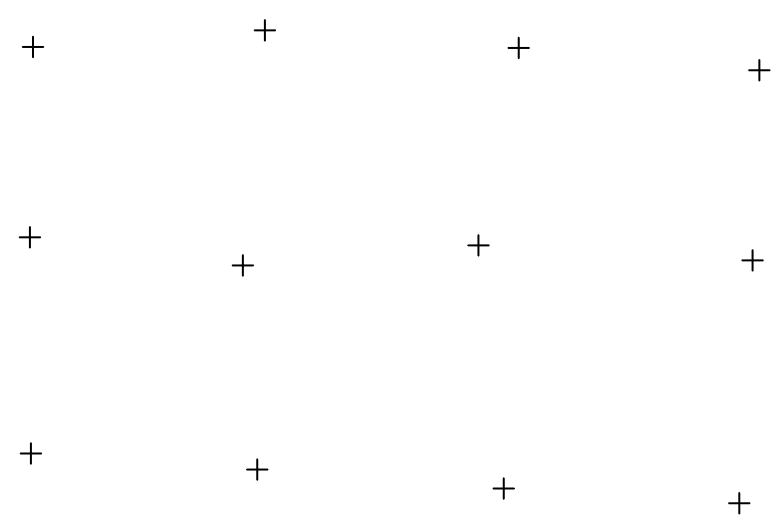
Often times, we will have a table with coordinates that we need to turn into spatial data

```
#Import coordinate table
SL <- read.csv("./Data/SampleLocations_UTM12.csv")

#Turn into a "Spatial Points Data Frame" by specifying columns with coordinates
coordinates(SL) <- cbind("UTM_East", "UTM_North")

#Specify projection
proj4string(SL) <- "+proj=utm +zone=12 +datum=NAD83 +units=m +no_defs +ellps=GRS80 +towgs84=0,0,0"

str(SL)
```

```
## Formal class 'SpatialPointsDataFrame' [package "sp"] with 5 slots
##   ..@ data       :'data.frame':  12 obs. of  1 variable:
##   .. ..$ Loc_ID: Factor w/ 12 levels "A","B","C","D",..: 1 2 3 4 5 6 7 8 9 10 ...
##   ..@ coords.nrs : int [1:2] 2 3
##   ..@ coords     : num [1:12, 1:2] 160623 166817 173597 180027 160541 ...
##   .. ..- attr(*, "dimnames")=List of 2
##   .. .. ..$ : NULL
##   .. .. ..$ : chr [1:2] "UTM_East" "UTM_North"
##   ..@ bbox       : num [1:2, 1:2] 160541 5713791 180027 5726432
##   .. ..- attr(*, "dimnames")=List of 2
##   .. .. ..$ : chr [1:2] "UTM_East" "UTM_North"
```

```
##    .. .. ..$ : chr [1:2] "min" "max"
##    ..@ proj4string:Formal class 'CRS' [package "sp"] with 1 slot
##    .. .. ..@ projargs: chr "+proj=utm +zone=12 +datum=NAD83 +units=m +no_defs +ellps=GRS80 +towgs84=0
```

```r
plot(SL)
```



### Add and view sample data

Now that we have our pretend locations, we can pretend that we collected some data at these locations, and add it to our points.

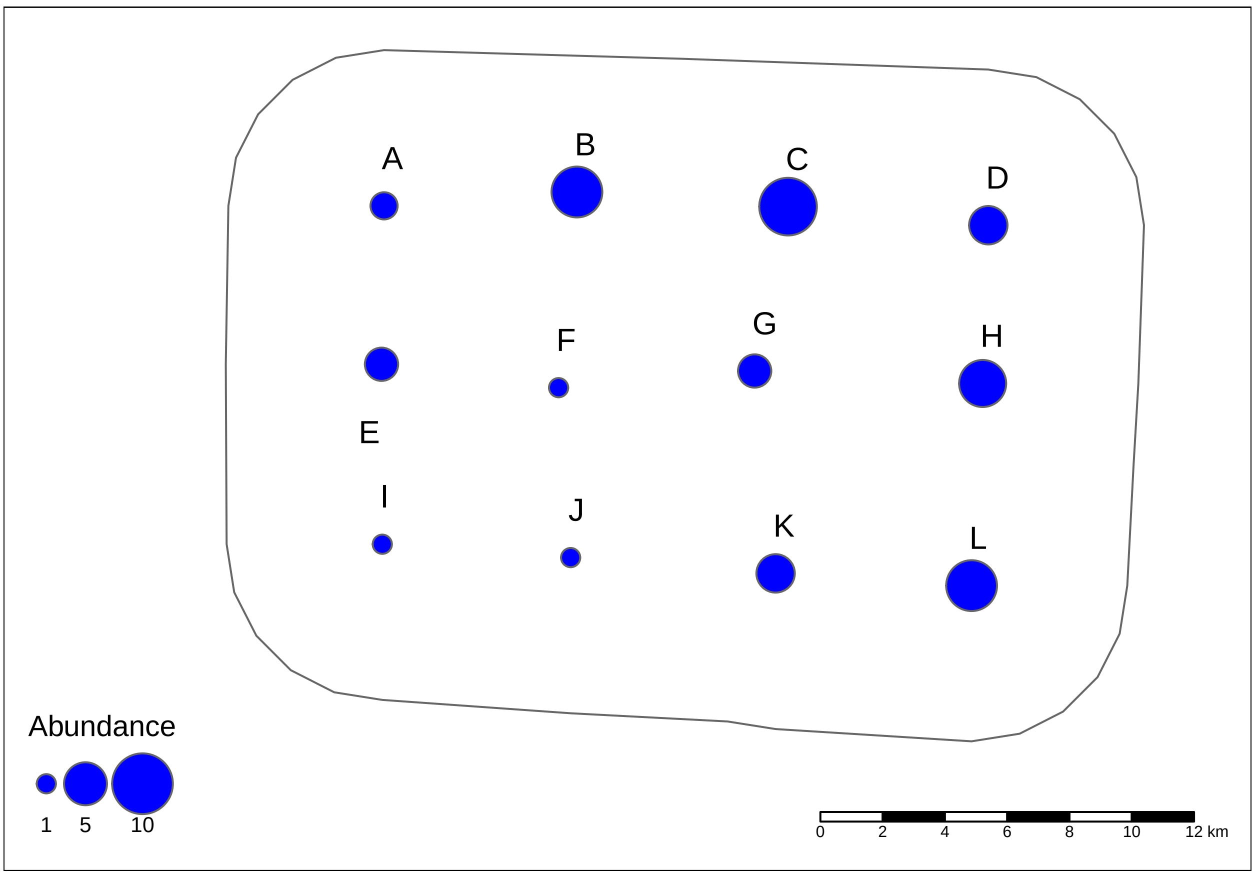
```r
set.seed(12)
SL$Abundance <- rpois(12, 5)
SL@data
```

```
##    Loc_ID Abundance
## 1       A         2
## 2       B         7
## 3       C         9
## 4       D         4
## 5       E         3
## 6       F         1
## 7       G         3
## 8       H         6
## 9       I         1
## 10      J         1
## 11      K         4
```

```
## 12       L         7
```

Next, use the *tmap* package to visualize these data

```
#First, add our local study area (LSA) to add context to our points
LSA_U12 <- readOGR(dsn = "Data/shapefiles", layer = "LSA_UTM12")

#Now create a quick map
Abun_Map <- tm_shape(LSA_U12) +
  tm_polygons(col = "white") +
tm_shape(SL) +
  tm_bubbles("Abundance",
             col = "blue",
             scale = 2,
             sizes.legend = c(1, 5, 10)) +
  tm_text("Loc_ID", auto.placement = 1) +
tm_layout(outer.margins=0, inner.margins=c(0.15,0.15,0.05,0.05), asp = 0) +
tm_scale_bar()

Abun_Map
```




## Basic spatial analysis

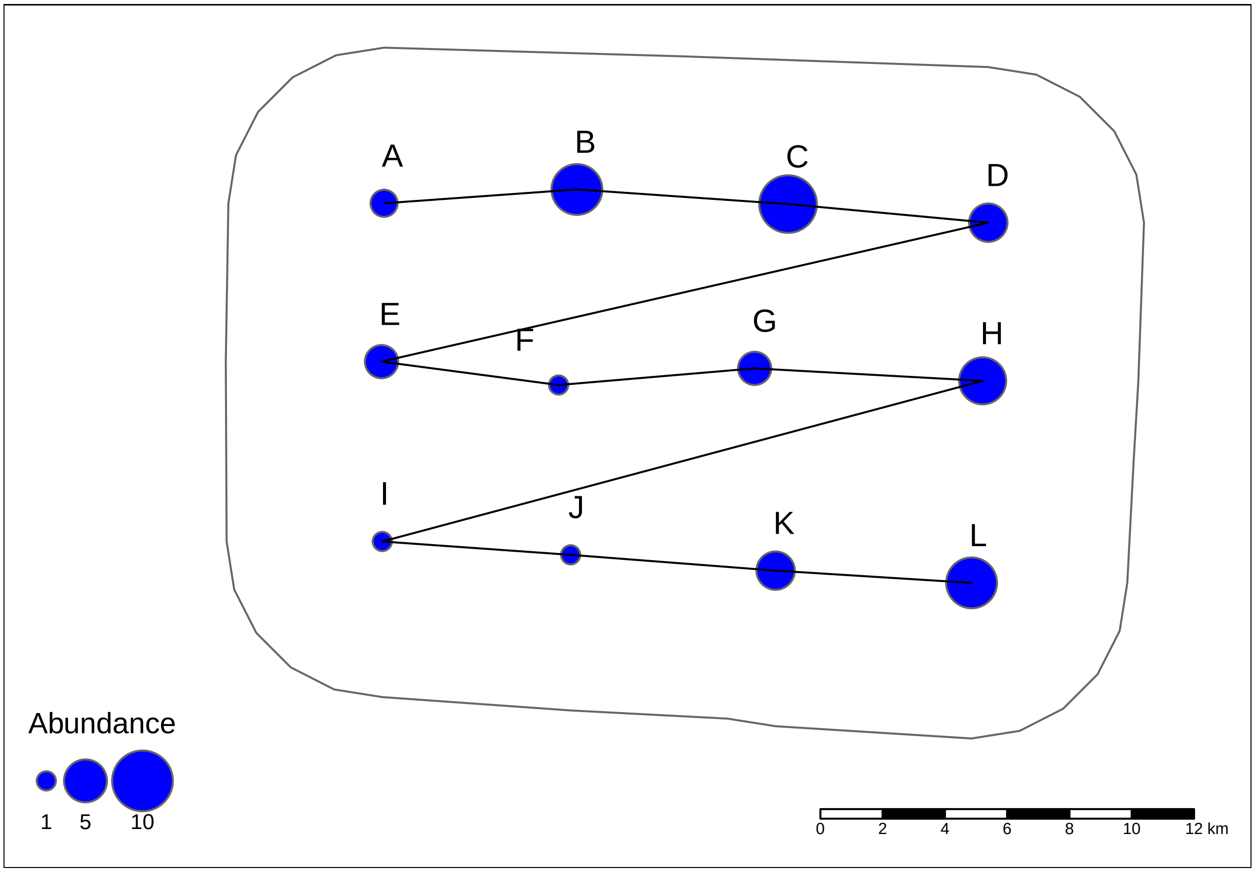
We can then do various analyses on these spatial points, for example by calculating the distance between them using the "spDists" tool in the *sp* package:

```r
spDists(SL)
```

```
##           [,1]      [,2]      [,3]      [,4]      [,5]      [,6]
##  [1,]     0.000  6209.893 12974.026 19413.999  5088.661  8094.543
##  [2,]  6209.893     0.000  6796.271 13253.022  8366.074  6311.543
##  [3,] 12974.026  6796.271     0.000  6457.655 14002.963  9386.413
##  [4,] 19413.999 13253.022  6457.655     0.000 19991.009 14752.271
##  [5,]  5088.661  8366.074 14002.963 19991.009     0.000  5736.504
##  [6,]  8094.543  6311.543  9386.413 14752.271  5736.504     0.000
##  [7,] 13029.437  8099.960  5385.964  8843.459 11984.947  6318.775
##  [8,] 20051.702 14407.118  8443.994  5085.222 19314.857 13618.669
##  [9,] 10867.144 12922.895 16950.168 21991.615  5779.058  7570.842
## [10,] 12786.856 11741.738 13257.497 17140.440  8685.202  5469.708
## [11,] 17245.761 13809.178 11783.791 13101.195 14327.092  9171.932
## [12,] 22467.048 17900.423 13523.035 11586.405 20239.575 14707.774
##           [,7]      [,8]      [,9]     [,10]     [,11]     [,12]
##  [1,] 13029.437 20051.702 10867.144 12786.856 17245.761 22467.048
##  [2,]  8099.960 14407.118 12922.895 11741.738 13809.178 17900.423
##  [3,]  5385.964  8443.994 16950.168 13257.497 11783.791 13523.035
##  [4,]  8843.459  5085.222 21991.615 17140.440 13101.195 11586.405
##  [5,] 11984.947 19314.857  5779.058  8685.202 14327.092 20239.575
##  [6,]  6318.775 13618.669  7570.842  5469.708  9171.932 14707.774
##  [7,]     0.000  7332.972 13187.753  8415.482  6533.753  9800.640
##  [8,]  7332.972     0.000 19958.108 14363.796  9021.907  6501.699
##  [9,] 13187.753 19958.108     0.000  6063.196 12664.636 18970.679
## [10,]  8415.482 14363.796  6063.196     0.000  6601.498 12907.485
## [11,]  6533.753  9021.907 12664.636  6601.498     0.000  6306.320
## [12,]  9800.640  6501.699 18970.679 12907.485  6306.320     0.000
```

Perhaps we also want to know how long it would take to visit each of these locations. We can do this by creating a line between them, then calculating the length of that line.

```r
#Create line that passes through all sample locations
PathLine <- SpatialLines(list(Lines(Line(cbind(SL$UTM_East,SL$UTM_North)), "L1")))

#Define projection for the line
proj4string(PathLine) <- "+proj=utm +zone=12 +datum=NAD83 +units=m +no_defs +ellps=GRS80 +towgs84=0,0,0

#Plot line on top of sample locations
Abun_Map +
  tm_shape(PathLine) +
    tm_lines(col = "black")
```




```
#Calculate length of line (the units here will be meters based on our projection)
SpatialLinesLengths(PathLine)
```

```
## [1] 97772.2
```

### Add landcover data as vector

In the "data" folder I have included a clipped land cover dataset downloaded from the Alberta Biodiversity Monitoring Institute (ABMI)

```
ABMI_LC <- readOGR(dsn = "Data/shapefiles", layer = "LC_ABMI_2010_UTM12")
head(ABMI_LC@data)
```

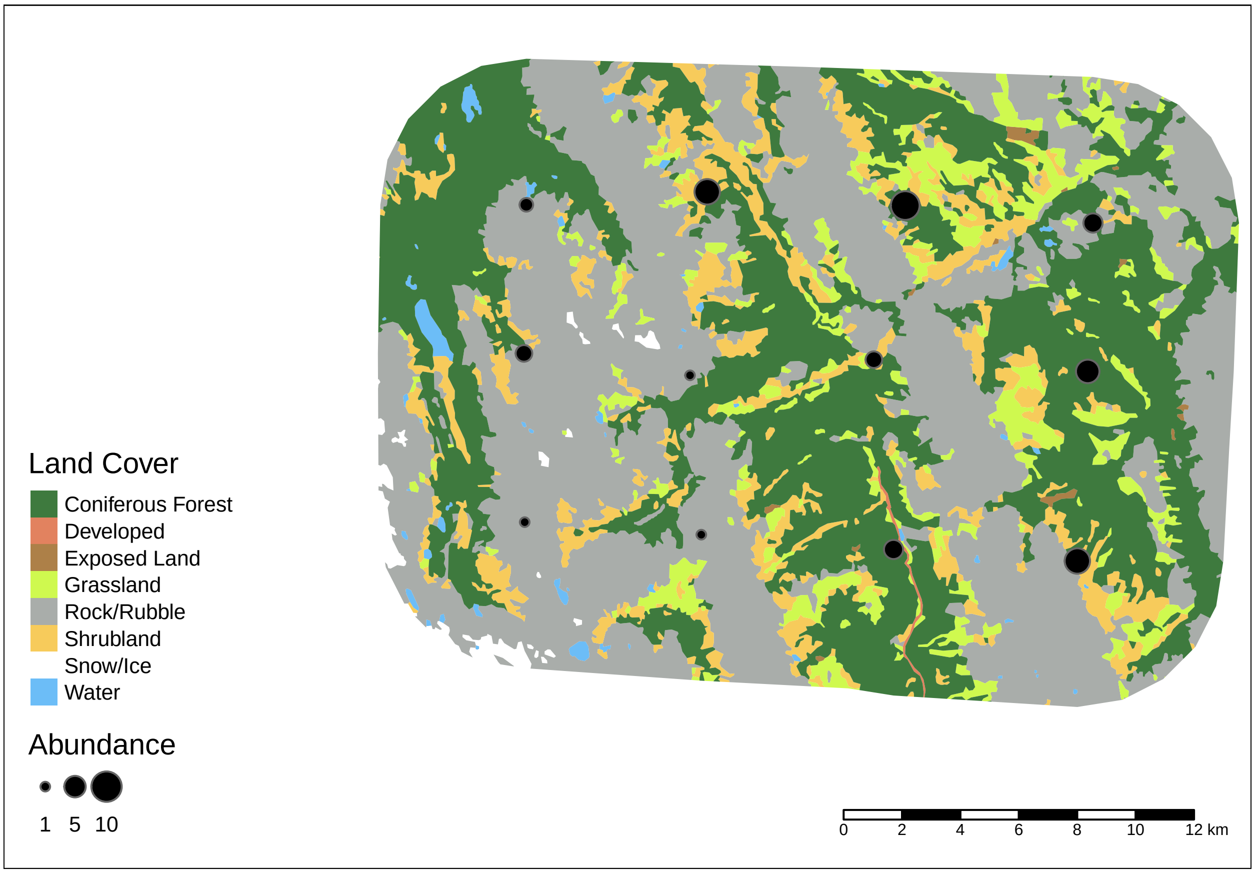
We can also map this land cover data. Here I have picked certain colours in a palette to represent the different land cover types. Since there were relatively few classes I picked them manually using the following website: https://htmlcolorcodes.com/

However there are also some good "cheatsheets" and programs available to do this automatically: https://www.nceas.ucsb.edu/~frazier/RSpatialGuides/colorPaletteCheatsheet.pdf

```
#Create colour palette for data
levels(ABMI_LC$LC_desc)
```

```
## [1] "Coniferous Forest" "Developed"         "Exposed Land"
## [4] "Grassland"         "Rock/Rubble"       "Shrubland"
## [7] "Snow/Ice"          "Water"
```

```r
LC.palette <- c("#3e7a3e", "#e2825f", "#ad8048", "#cff94f",
                "#a9adaa", "#f7cb5b", "#ffffff", "#6cbdf7")

#Create map of data:
tm_shape(ABMI_LC) + #Specify shapefile to be mapped
  tm_fill("LC_desc", palette = LC.palette,  title = "Land Cover") + #select fill data
tm_shape(SL) +
  tm_bubbles("Abundance",
             col = "black",
             scale = 1,
             sizes.legend = c(1, 5, 10)) +
tm_layout(outer.margins=0, inner.margins=c(0.15,0.3,0.01,0.01), asp = 0) +
tm_scale_bar()
```




## Raster Data

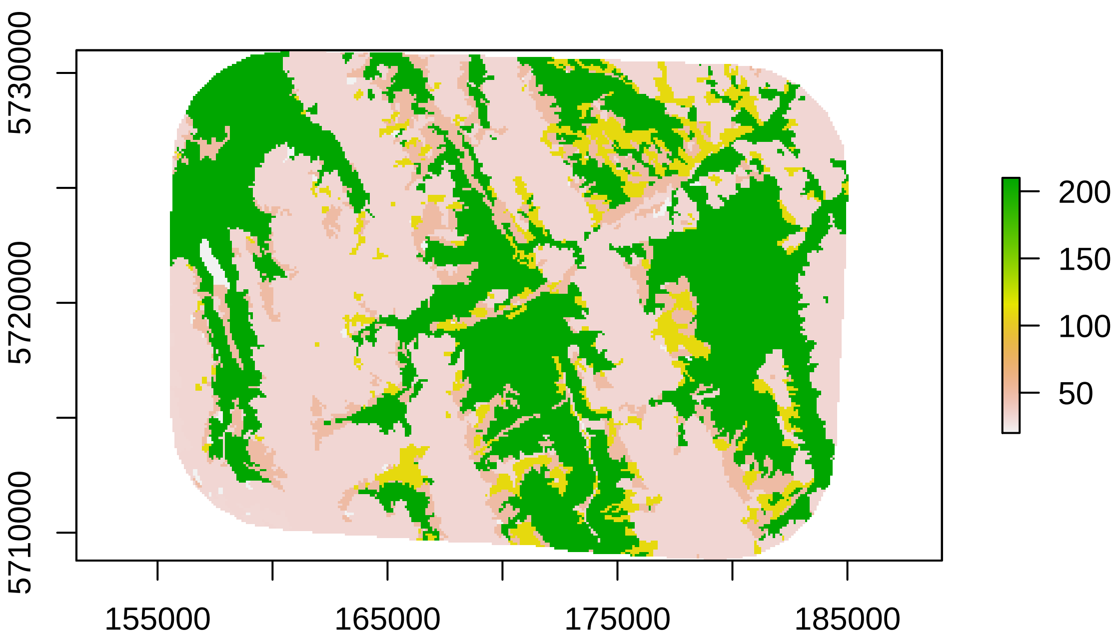
Often the spatial data that we want to use is best represented in raster format - especially for continuous variables (e.g. elevation) or variables covering a landscape (e.g. landcover)

First, create a template raster with the spatial extent ("ext") and projection ("crs") set to be the same as our polygon landcover layer ("ABMI_LC"). We are using a resolution, or pixel size, of 100 map units (in this case metres).

```r
template.raster <- raster(ext = extent(ABMI_LC), resolution = 100, crs = CRS(projection(ABMI_LC)))
```

Next we essentially "stamp" our land cover polygons onto our template raster.

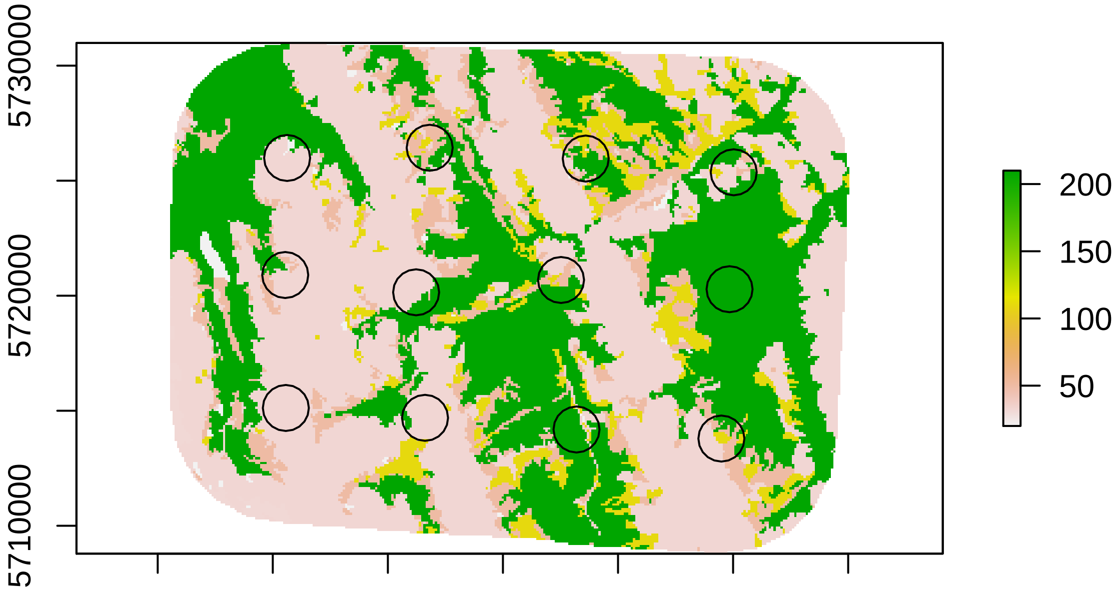
```
#This step may take a couple minutes to run:
LCrast <- rasterize(x = ABMI_LC, y = template.raster, field = "LC_class", fun = 'last')

plot(LCrast) #lets see what our raster looks like!
```




## Buffer sampling locations and extract values

First we need to create "buffers", or circles with a set radius around our sampling locations. We will do this using the *gBuffer* tool in the **rgeos** package. We then can extract the landcover values within each buffer using the *extract* tool in the **raster** package.

```
#Create 1000 m buffers around sampling locations:
SL_buffer1km <- gBuffer(spgeom = SL, byid = TRUE, id = SL$Loc_ID, width = 1000)
plot(LCrast)
plot(SL_buffer1km, add = TRUE) #look at the buffers created
```




```
#Extract number of pixels of each land cover class under each polygon:
LC1km <- raster::extract(x = LCrast, y = SL_buffer1km)

#Find frequency of each land class within each polygon:
LC1km.fq <- lapply(LC1km, table)

#Calculate proportion of land cover within each polygon:
LC1km.pr <- lapply(LC1km.fq, FUN = function(x){x/sum(x)})

LC1km.pr
```

```
## [[1]]
##
##         20         32        210
## 0.04516129 0.84516129 0.10967742
##
## [[2]]
##
##        32        50       110       210
## 0.1270358 0.4690554 0.1107492 0.2931596
##
## [[3]]
##
##        32        50       110       210
## 0.1205212 0.2508143 0.1628664 0.4657980
##
```

```
## [[4]]
##
##         32         50        110        210
## 0.42948718 0.10576923 0.04487179 0.41987179
##
## [[5]]
##
##        32        50       210
## 0.6612903 0.1774194 0.1612903
##
## [[6]]
##
##          32          50         210
## 0.690322581 0.009677419 0.300000000
##
## [[7]]
##
##         32         50        110        210
## 0.30000000 0.21612903 0.07096774 0.41290323
##
## [[8]]
##
## 210
##   1
##
## [[9]]
##
## 32
##  1
##
## [[10]]
##
##         32        110        210
## 0.91082803 0.01273885 0.07643312
##
## [[11]]
##
##         20         32         34        110        210
## 0.01290323 0.09354839 0.05806452 0.04193548 0.79354839
##
## [[12]]
##
##        32        50       110       210
## 0.3677419 0.2870968 0.1193548 0.2258065
```