Geospatial Analysis with R

Michael Harper, Luke Blunden

August 29, 2017

University of Southampton

Uses of Spatial Data in Science

- spatial analysis: using location or spatial relationships as an **explanatory** or **predictive** variable
- examining the effects of **scale**
- backdrops to show **context**

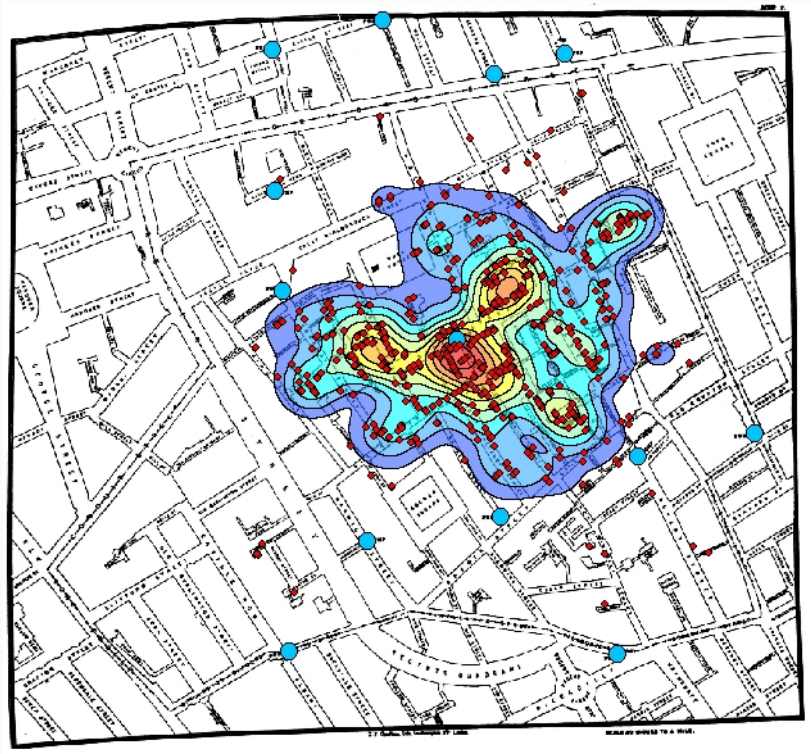


Figure 1: Cholera Map of 1854 London Outbreak

GIS vs. R

- Many tools options also available

- Other tools avaialble (QGIS, Python)

Which tool is best to use?

Well, it depends on the task!

GIS vs. R

GIS

- Visual interaction
- Data management
- Geometric operations
- Standard workflows
- Single map production
- Click, click, click, click
- Speed of execution

R

- Data and model focused
- Analysis
- Attributes as important
- Creativity and innovation
- Many (simpler) maps
- Repeatability (script)
- Speed of development

R Examples (1)

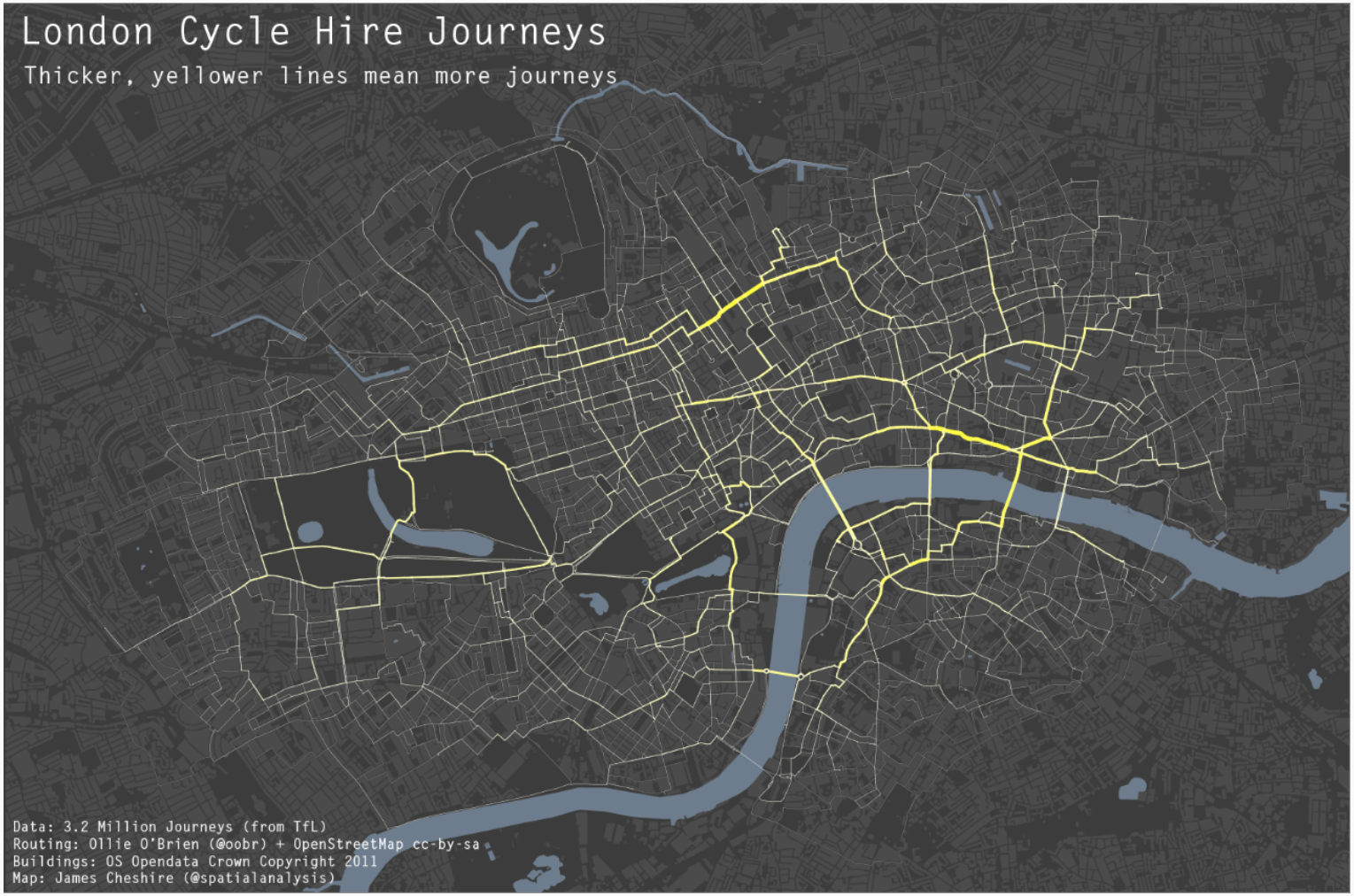


Figure 2: London Cycle Hire Journeys

http://spatial.ly/2012/02/great-maps-ggplot2/

R Examples (2)

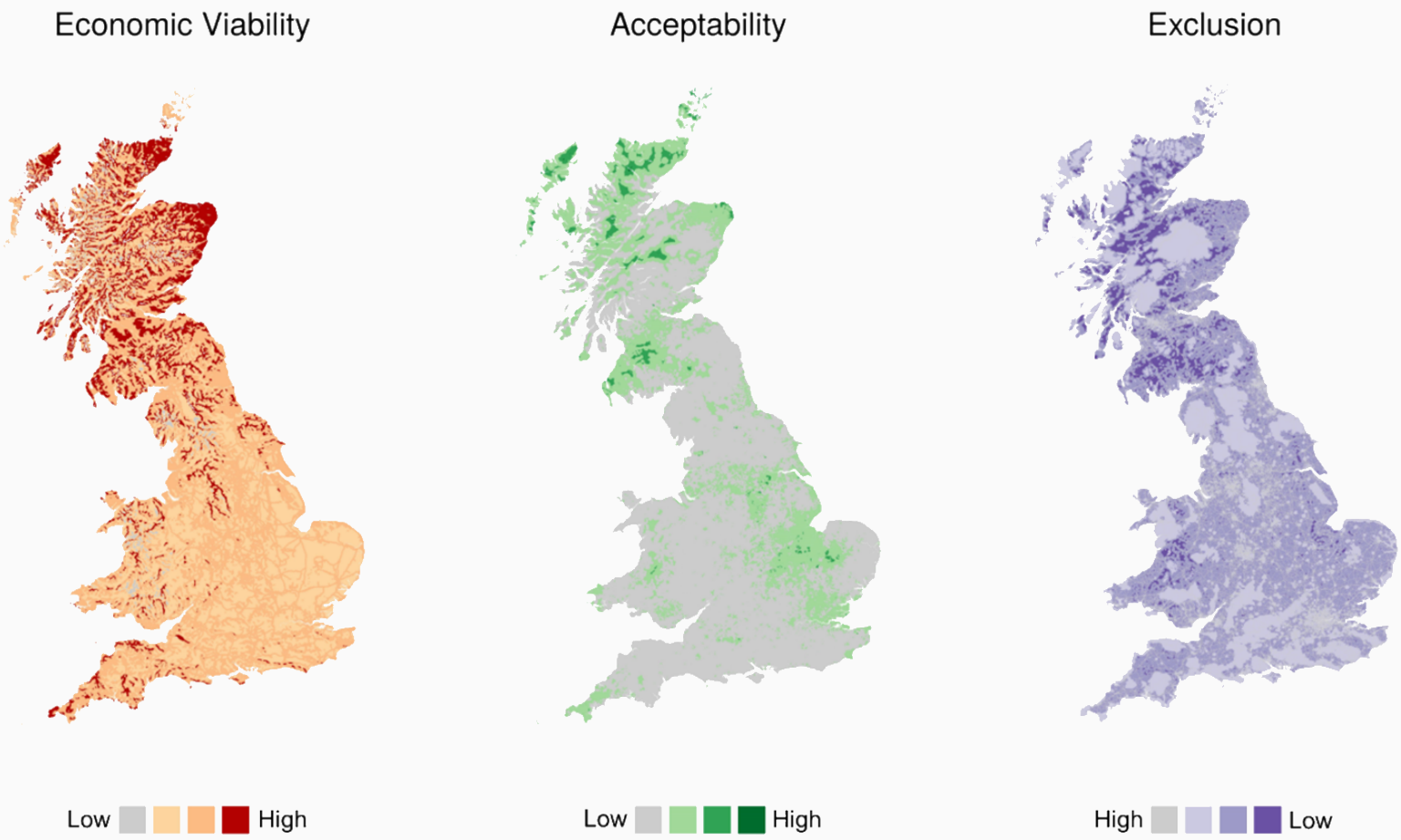


Figure 3: Wind Turbine Analysis of Great Britain

Challenges with Spatial Data in R

Understanding and manipulating spatial data in R can be a real challenge!

- Many types and formats of spatial data
- R is not the easiest program to learn
- There are a many packages from a very diverse user community

Presentation Goals

1. Help you get to the point where you can do basic analysis or visualization
2. Point out some of the commonly-used packages for spatial data manipulation and analysis
3. Be a resource you can come back to
4. Provide Guidance on useful resources to use after the course

Beware of the learning curve!

Takes patience to master.

Outline

1. Creating Spatial Objects
2. Data Table Operations
3. Coordinate Systems and Reprojection
4. Geoprocessing
5. Presenting Results

Subjects Omitted

More advanced visualization, Spatial statistics, Geodatabases, Network analysis, Many other...

Before We Start...

- Many packages are used within the analysis.

```
library(rgdal)  # loads geospatial libraries
library(rgeos)  # dealing with geospatial files
library(sp)  # spatial projections
library(raster)  # loading rasters and geoprocessing
library(ggplot2)  # Creating maps
library(spatstat)  # Used for spatial statistics
```

- Don't worry for now. These will be explained in more detail later.

PART I: Creating Spatial Objects

Representing Physical Features

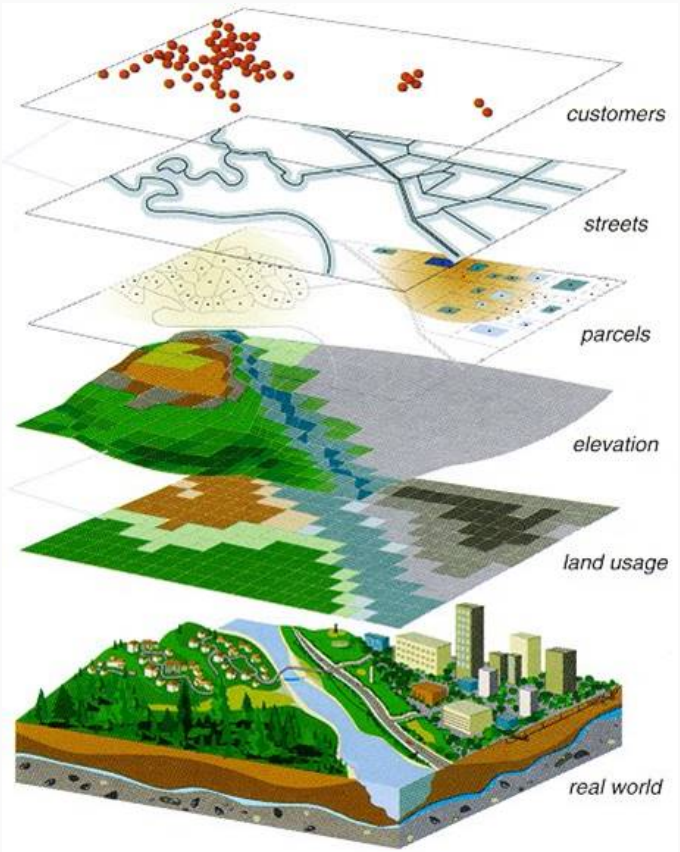


Figure 4: Representation of Figure Layers

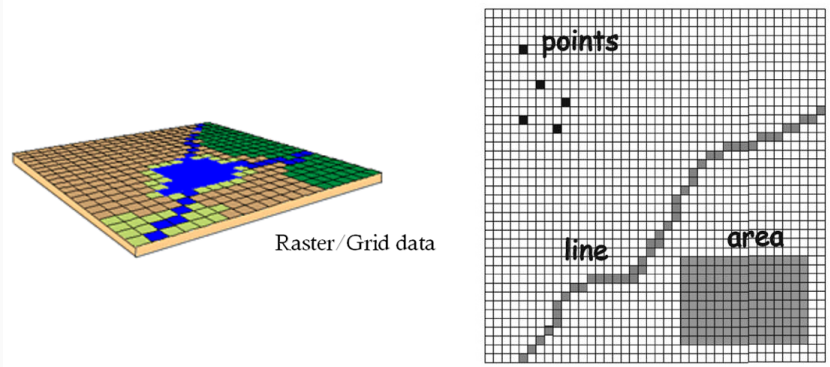


Figure 5: Raster Features

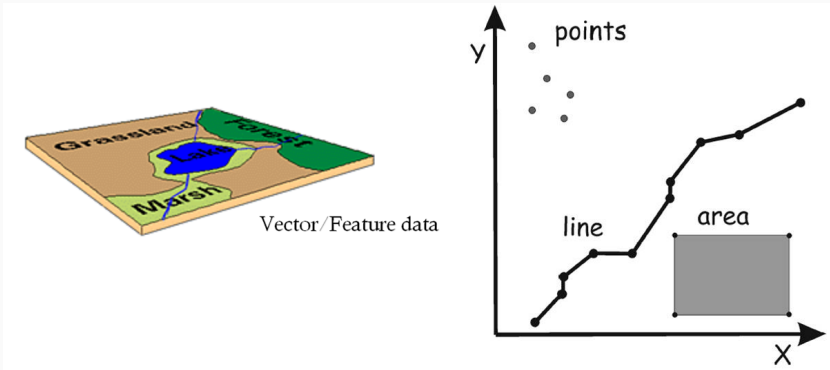


Figure 6: Vector Features

Creating SpatialPoints Objects

- The package sp is primarily used for creating spatial data types
- The code `getClass('Spatial')` can be run to see all types

Three options to create spatial data in R:

1. Create from scratch
2. Promote a data frame with X and Y columns
3. Import a GIS file

- Methods 2 and 3 are most commonly used!
- Tutorial will focus on creation of Points

Method 1: Create a SpatialPoints Object from Scratch

- Let's create an example dataset!

```
library(sp)
# create a set of random coordinates
xy <- matrix(data = runif(n = 100), ncol = 2)
head(xy, n = 3)  # show first three rows
```

```
##             [,1]      [,2]
## [1,] 0.03463934 0.6675732
## [2,] 0.15650207 0.1940877
## [3,] 0.44065106 0.1272320
```

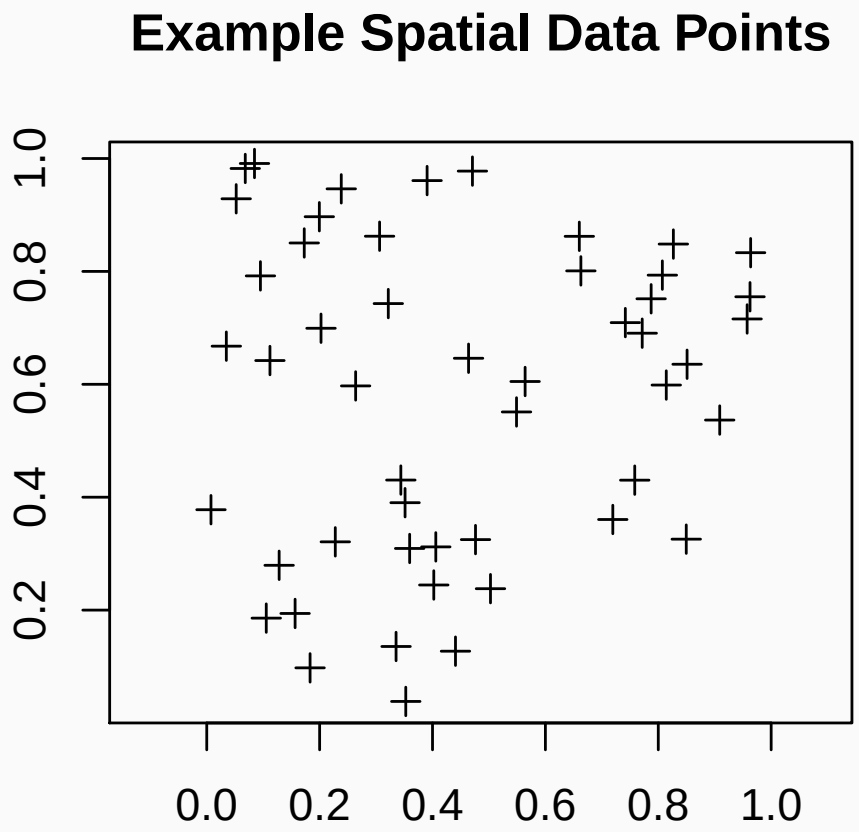
- SpatialPoints function can be used to create a dataset from XY coordinates

```
xySp <- SpatialPoints(xy)
```

Method 1: Create a SpatialPoints Object from Scratch

SpatialPoints can be plotted as typical XY data using the `plot()` function

```
plot(xySp, axes = TRUE, main = "Example Spatial Data Points")
```



Method 2: Promote a DataFrame to a SpatialPointsDataFrame

- It is unlikely you will use Method 1 often
- You will more likely have an existing dataset with XY coordinates.
- For the tutorial, We will used the canada.cities dataset from the maps package.

```
library(maps)
head(canada.cities, n = 3)
```

```
##             name country.etc    pop   lat    long capital
## 1 Abbotsford BC           BC 157795 49.06 -122.30       0
## 2      Acton ON           ON   8308 43.63  -80.03       0
## 3 Acton Vale QC           QC   5153 45.63  -72.57       0
```

- Data already has X (long) and Y (lat) references.

Method 2: Promote a DataFrame to a SpatialPointsDataFrame

- We must let R know which columns are the X and Y coordinates

```
# Create a new variable to edit the data
cities <- canada.cities

# State which columns contain the x and y
# coordinates
coordinates(cities) <- ~long + lat
```

- By assigning coordinates we create a SpatialPointsDataFrame:

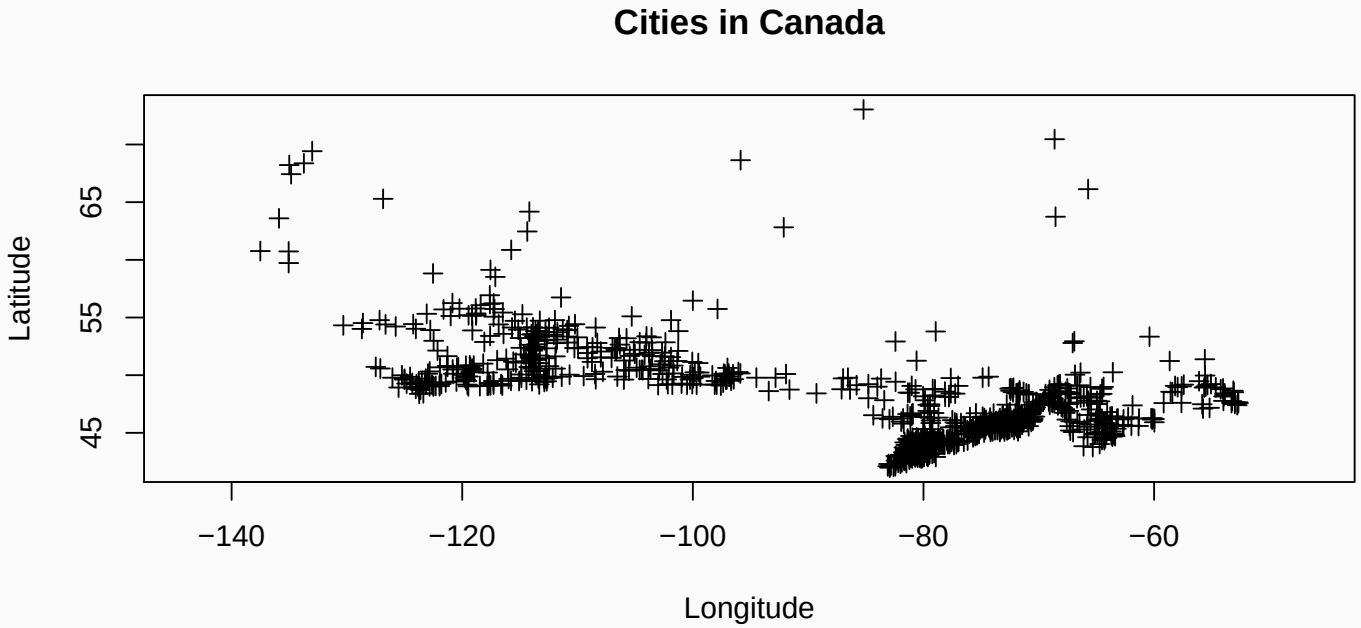
```
class(cities)
```

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

Method 2: Promote a DataFrame to a SpatialPointsDataFrame

We can map the results of the SpatialDataFrame as before:

```
plot(cities, axes = TRUE, main = "Cities in Canada",
    xlab = "Longitude", ylab = "Latitude")
```



- This map is not very useful yet! We will add to this map later!

Method 3: Import a GIS File

- Package rgdal has a large set of import and export filters for both vector and raster GIS data (Shapefiles, KML, GPX etc.).
- For the complete list of vector import filters, use the function `ogrDrivers()`.
- The tutorial will focus on **Shapefiles**, a commonly used format in GIS software.
- We will use the "scot_BNG" file.
- `readOGR()` is the function you use to import a vector file.

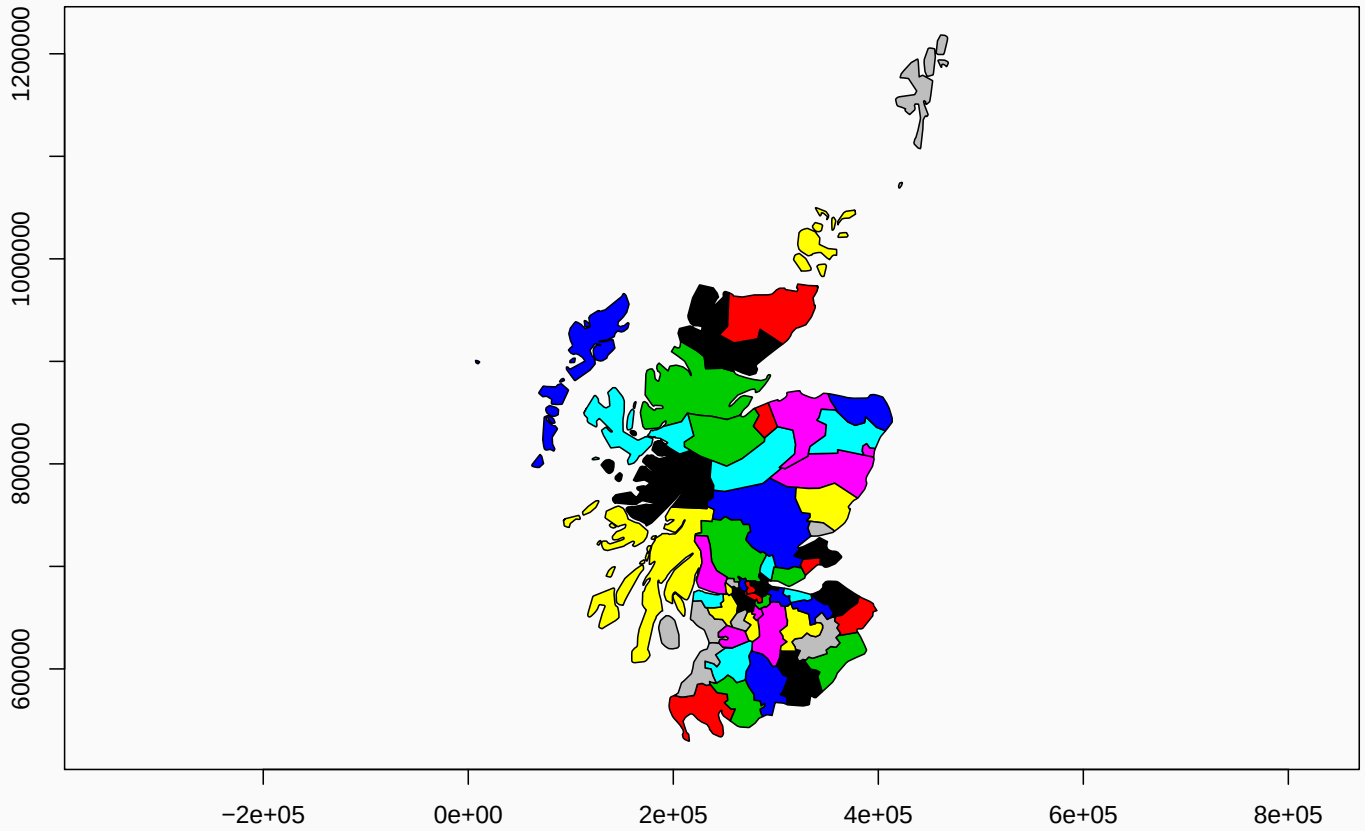
```
# Find the file path of the vectors
vectorDir <- system.file("vectors", package = "rgdal")
scotland <- readOGR(dsn = vectorDir, layer = "scot_BNG",
    verbose = FALSE)
```

- Note: you don't need to include the file extension when loading the object

Method 3: Import a GIS File

- Shapefiles are typically easy to plot straight away

```
plot(scotland, axes = T, asp = 1, col = palette())
```



- We will come back to plotting again later!

PART II: Working With SpatialDataFrames

Working with the Data Frame

- You can work with the data frame as you would with any data frame
- To use the attribute table of a SpatialPointsDataFrame, use the **data** slot (with '@' character):

```
nrow(cities@data)
```

```
## [1] 916
```

```
# show the column names
names(cities@data)
```

```
## [1] "name"       "country.etc" "pop"
```

- What happened to the coordinates?
- We can read them with the `coordinates()` function

```
coordinates(cities)[1, ]
```

```
##    long     lat
## -122.30   49.06
```

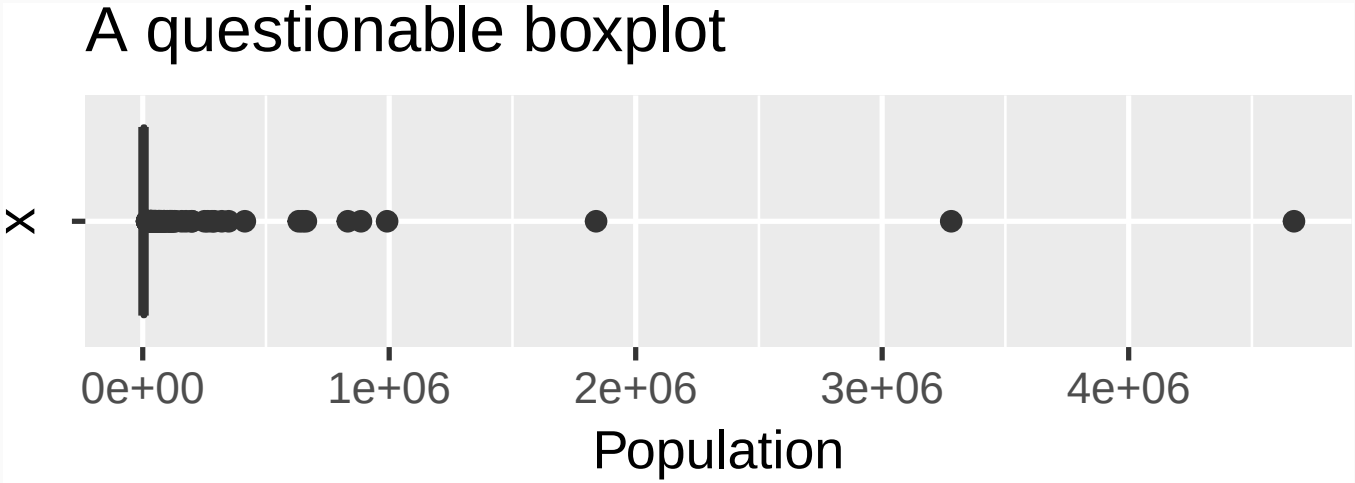
Calculating Summaries

- SpatialDataframes can *mostly* be treated like a normal Dataframe

```
mean(cities$pop)   # Calculate Mean population
```

```
## [1] 27463.94
```

```
ggplot2::qplot(x = "", y = cities$pop, geom = "boxplot",
    main = "A questionable boxplot", ylab = "Population") +
    coord_flip()
```



Adding a Column to the Data Frame of a SPDF Object

- Calculate new columns in the data frame

```
# Calculate the size order of cities
cities$sizerank <- rank(-cities$pop)

head(cities@data, n = 5)
```

```
##             name country.etc    pop capital sizerank
## 1 Abbotsford BC           BC 157795       0       20
## 2      Acton ON           ON   8308       0      220
## 3 Acton Vale QC           QC   5153       0      314
## 4    Airdrie AB           AB  25863       0       86
## 5    Aklavik NT           NT    643       0      915
```

Filter on an Attribute

- Standard R syntax can be used to filter SpatialDataFrames.[1]

```
# Filter cities by those with a population more
# than 100,000
BigCities <- cities[cities$pop > 1e+05, ]

# Count how many cities meet criteria
nrow(BigCities)
```

```
## [1] 29
```

[1] Check out http://www.statmethods.net/management/subset.html

Filter on an Attribute

```
par(mfrow = c(1, 2))  # Plot graphs side by side
plot(cities, axes = T, asp = 1, pch = 16, main = "Original")
plot(cities, axes = T, asp = 1, pch = 16, main = "Highlighing Big Citie
plot(BigCities, pch = 1, col = "red", cex = 3, add = TRUE)
```

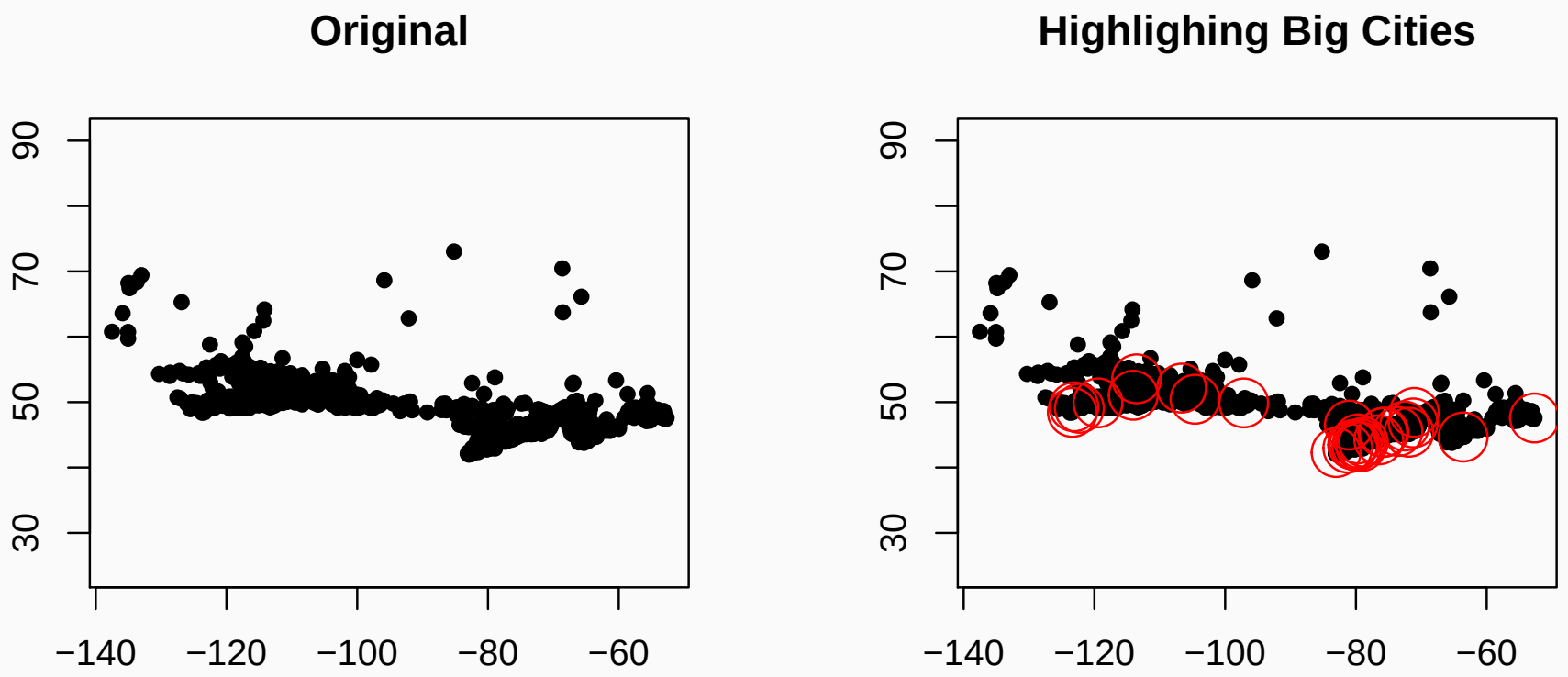


Figure 7: Subsetting of spatial dataframes

Enriching Data

- You may want to link data to your spatial data
- e.g. deomgraphic data to census wards

INPUT

OBJECT ID#	Landuse Code
1	2
2	0
3	1

Join Fields

+

Landuse Code	Landuse Type
0	Unclassified
1	shrub
2	water

OUTPUT

OBJECT ID#	Landuse Code	Join Table Landuse Code	Join Table Landuse Type
1	2	2	water
2	0	0	Unclassified
3	1	1	shrub

Enriching Data: Example

- Let's join some census data with shapefiles for the Southampton region

```
# Load Shapefile
Solent <- readOGR(dsn = "../data/SolentLEP", "Solent_revised",
    verbose = FALSE)
head(Solent[1:4], n = 2)
```

```
##      OA11CD   LAD11CD LAD11NM  AREA_HECT
## 0 E00116566 E07000090  Havant 5.84000000
## 1 E00116564 E07000090  Havant 9.75000000
```

```
nrow(Solent)
```

```
## [1] 3826
```

Enriching Data: Example

- We can load some Census Data

```
# Age data for all of the UK
AgeData <- read.csv("../data/MedianAge.csv",
                    stringsAsFactors = FALSE)
head(AgeData,n=3)
```

```
##   GeographyCode MedianAge
## 1     K04000001        39
## 2     E92000001        39
## 3     W92000004        41
```

```
nrow(AgeData)
```

```
## [1] 223722
```

- The age data has records for the whole of England

Enriching Data: Example

- We can easily merge the age data with the boundary data using the `merge()` function

```
# Merge Datasets
SolentMerged <- merge(Solent, AgeData, by.x = "OA11CD",
    by.y = "GeographyCode")
# Lets focus on Southampton
Southampton <- SolentMerged[SolentMerged$LAD11NM ==
    "Southampton", ]
```

Enriching Data: Example

```
spplot(Southampton, "MedianAge", main = "Median Age (years)")
```

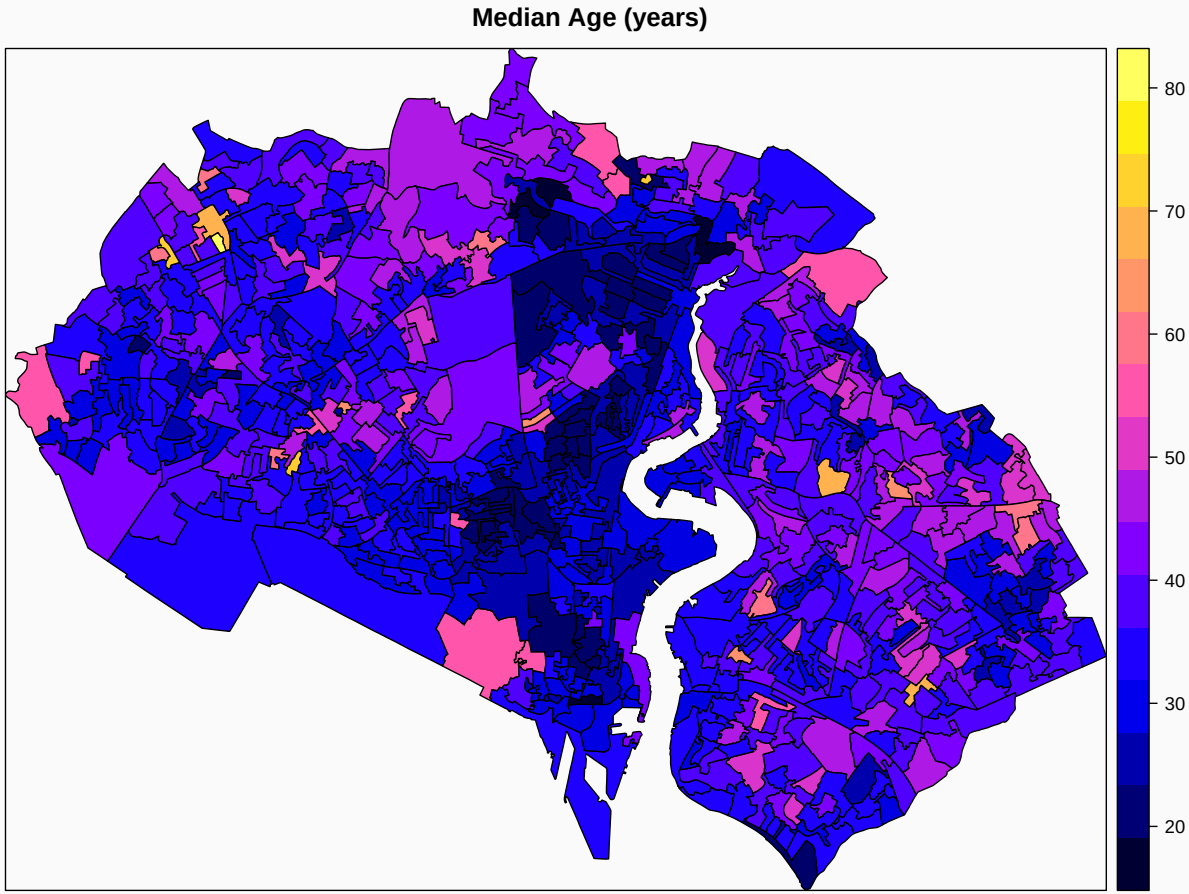


Figure 8: Median Age with Output Areas in Southampton. Data from 2011 Census

PART III: Coordinate Reference Systems

Coordinate Reference Systems (CRS)

- Very important! R won't give you a warning when loading like in ArcGIS.
- You don't always need to give one, but it is recommended to always do.
- If you create spatial data, **you will usually have to specify** the projection.
- packages sp and rgdal are used for managing CRS.

Find out more

A useful guide is provided here about spatial projection in R
https://goo.gl/FAsv4k

Coordinate Systems: Example

- Earlier, we created the SpatialDataFrame of Canadian Cities called `cities`.
- We can check the spatial projection using the function `crs()`

```
crs(cities)
```

```
## CRS arguments: NA
```

- Currently the data has no CRS! We need to assign one
- Dataset has WGS84 projection, which is the mostly commonly used

```
# Assign Projection to Cities
crs(cities) <- CRS("+init=epsg:4326")
```

Find out spatial projection codes

An online registry of spatial projections is available at http://www.epsg-registry.org for Coordinate Reference Systems

Coordinate Systems: Example

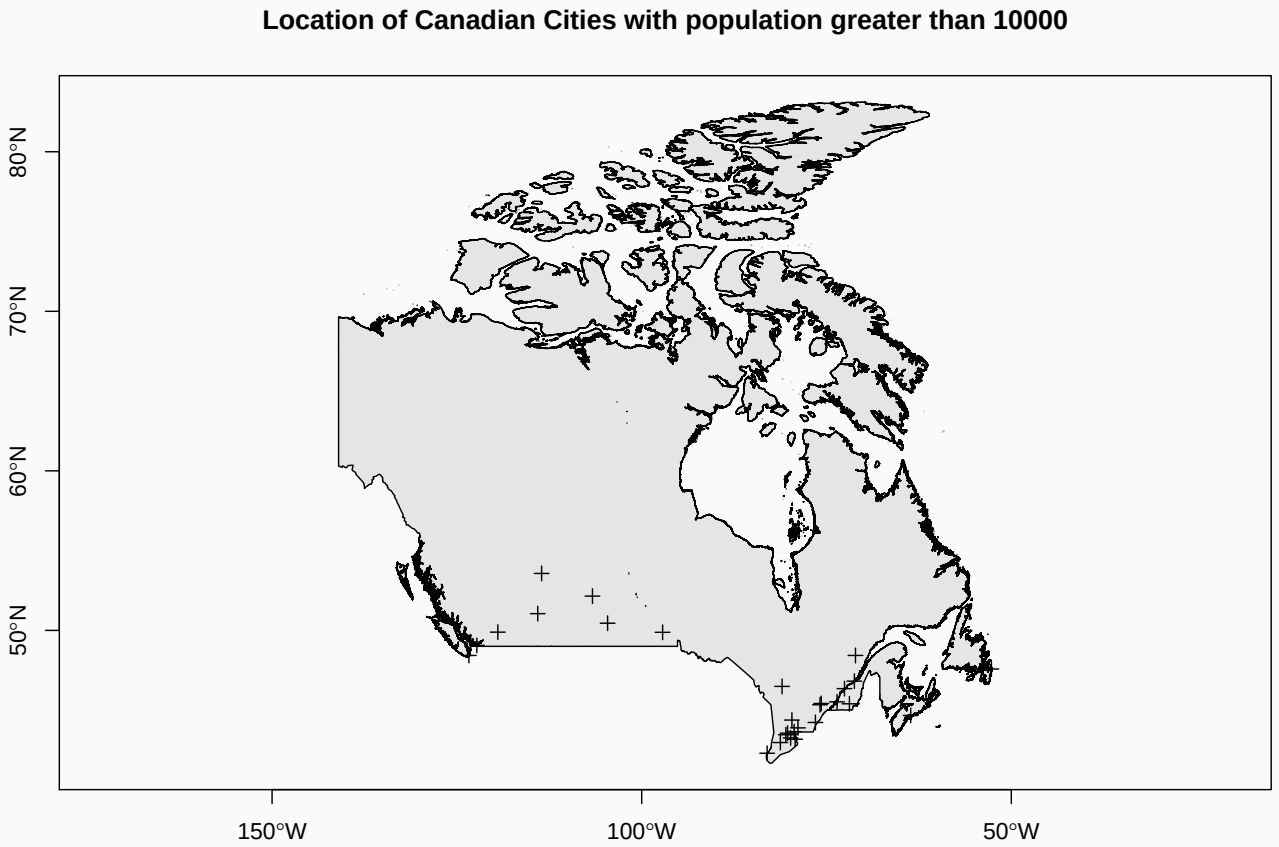
- We will load data for administrative borders in Canada using the function `getData()` from the raster package [2].

```
# Load Canada Data
canada <- raster::getData("GADM", country = "CAN",
    level = 0, download = FALSE, path = "../data")
# Check Coordinate System
crs(canada)
```

```
## CRS arguments:
##  +proj=longlat +datum=WGS84 +no_defs +ellps=WGS84 +towgs84=0,0,0
```

[2] This provides global vector and raster data for countries, administrative boundaries, altitude models and climatic data.

Coordinate Systems: Example



Example: Reprojecting

- The function `spTransform()` can be used to reproject a CRS

```
GB <- raster::getData("GADM", country = "GBR", level = 1,
    path = "../data")
GB_BNG <- spTransform(GB, CRS("+init=epsg:27700"))
```

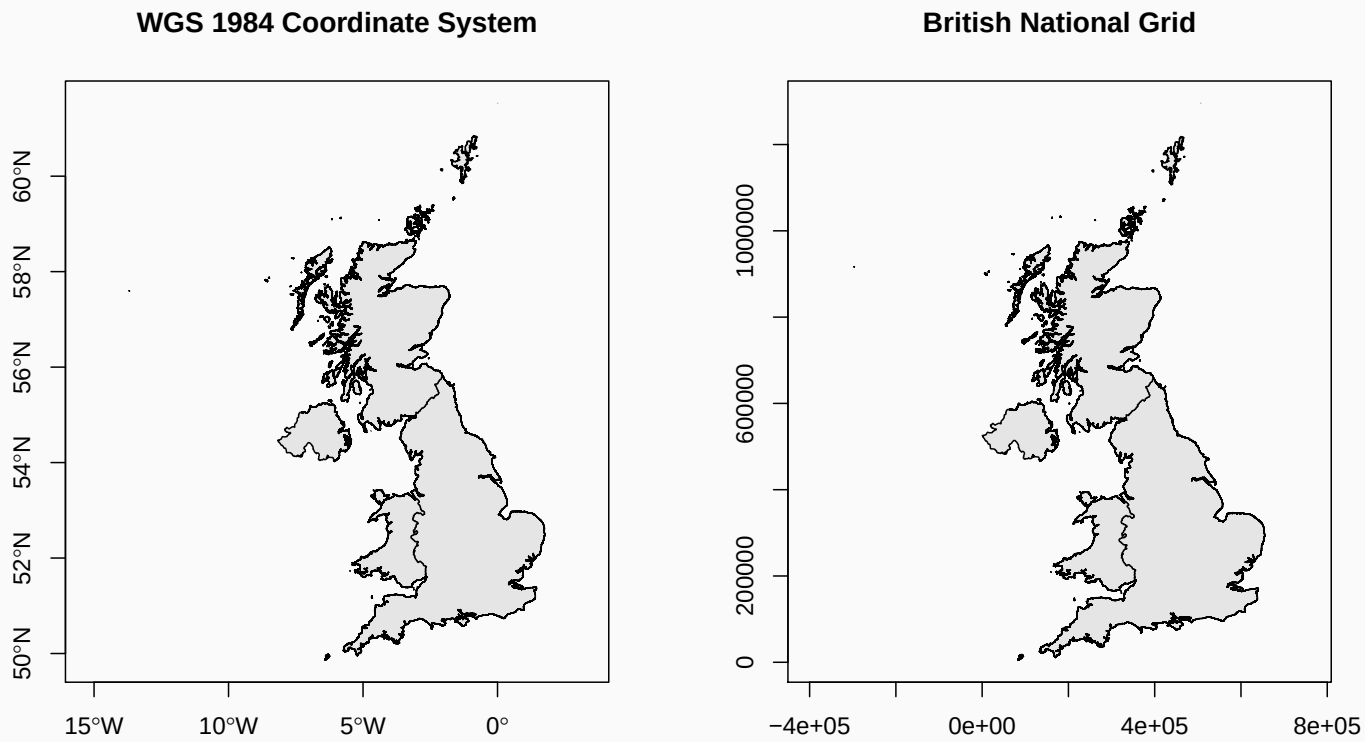


Figure 9: Comparison of coordinate systems

PART IV: Geoprocessing

Geoprocessing: rgeos package

- rgeos is the main package used within R for geospatial analysis
- interface to GEOS (Geometry Engine Open Source) Library

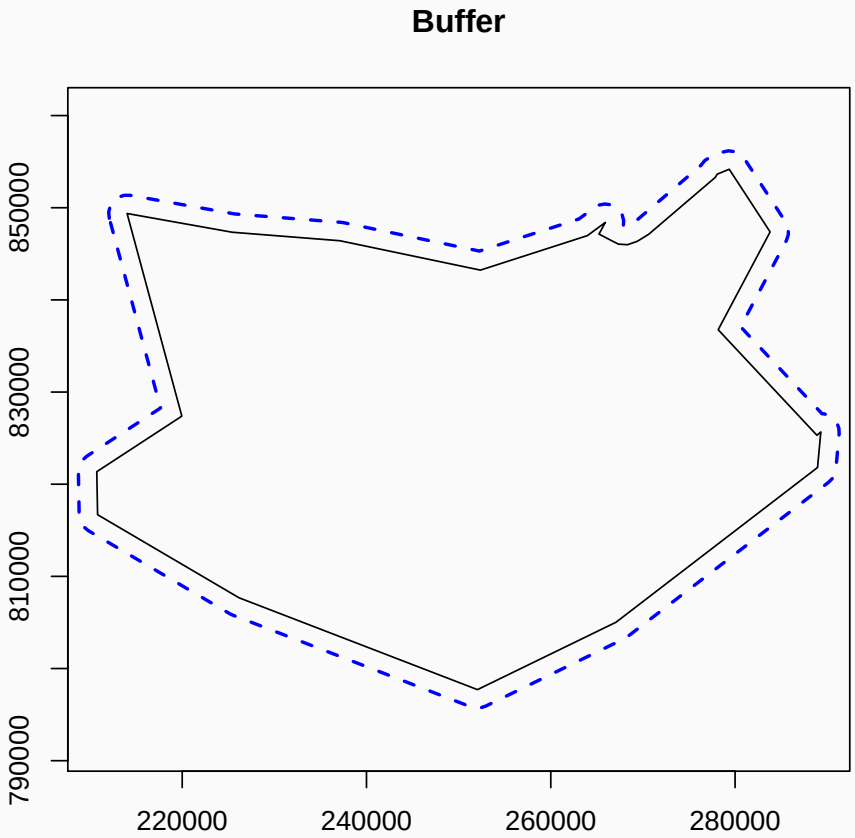
Example Processes

Union, distance, intersection, convex hull, envelope, buffer, simplify, polygon assembly, valid, area, length, intersects, touches, disjoint, crosses, within, contains, overlaps, equals, covers, etc.

Buffer

- apply a 2km buffer to the county of Inverness

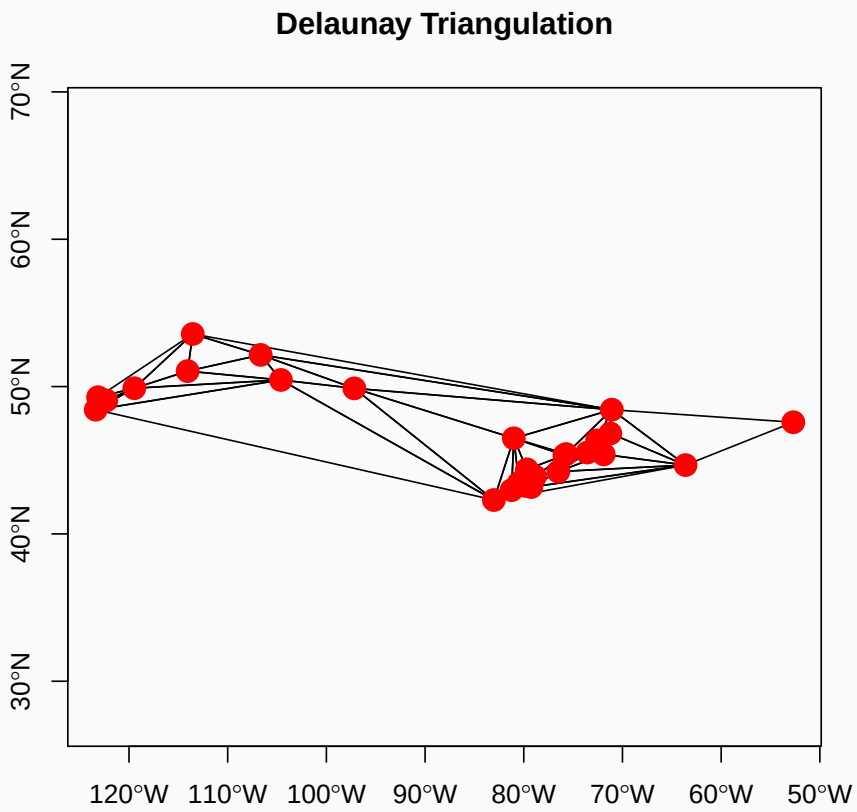
```
inver <- scotland[scotland$NAME == "Inverness", ]
inver_buf <- gBuffer(inver, width = 2000)
```



Delaunay Triangulation

- Form Delaunay triangulation for the cities in Canada

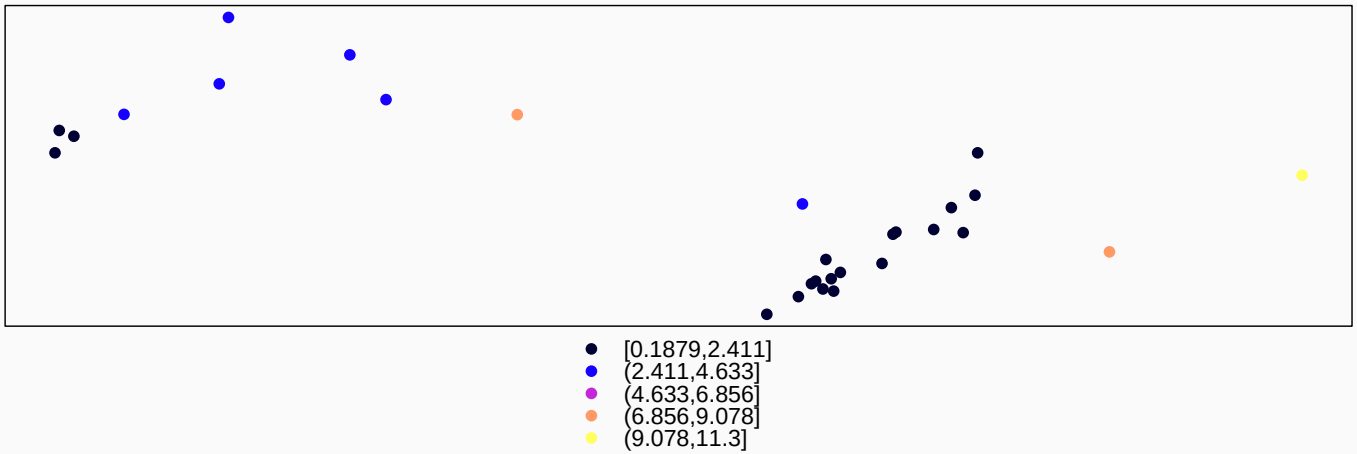
```
delTri <- gDelaunayTriangulation(BigCities)
plot(delTri, axes = TRUE, , main = "Delaunay Triangulation")
plot(BigCities, col = "red", pch = 16, cex = 2, add = TRUE)
```



Spatial Proximity

- How far is it to the nearest city?
- spatstat provides adds a spatial proximity functions.

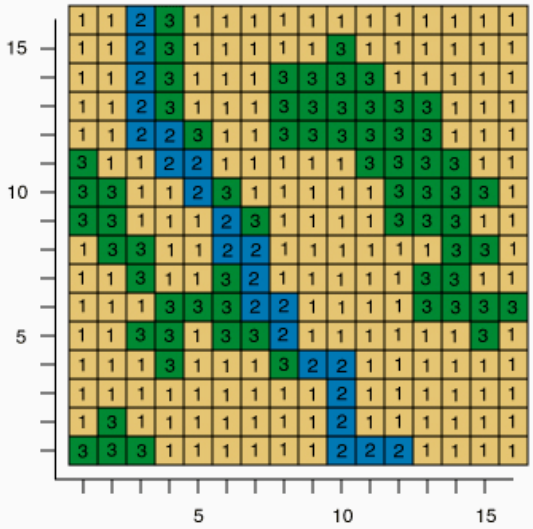
```
library(spatstat)
BigCities$Distance <- nndist(X = BigCities@coords)
spplot(BigCities, "Distance")  # Plot Results
```



PART V: Raster Data

Raster package

- Most control through `raster` package
- Read, write and manipulate gridded spatial data
- Basic and high-level analysis functions
- Processing of large files



Raster Layer: Create from Scratch

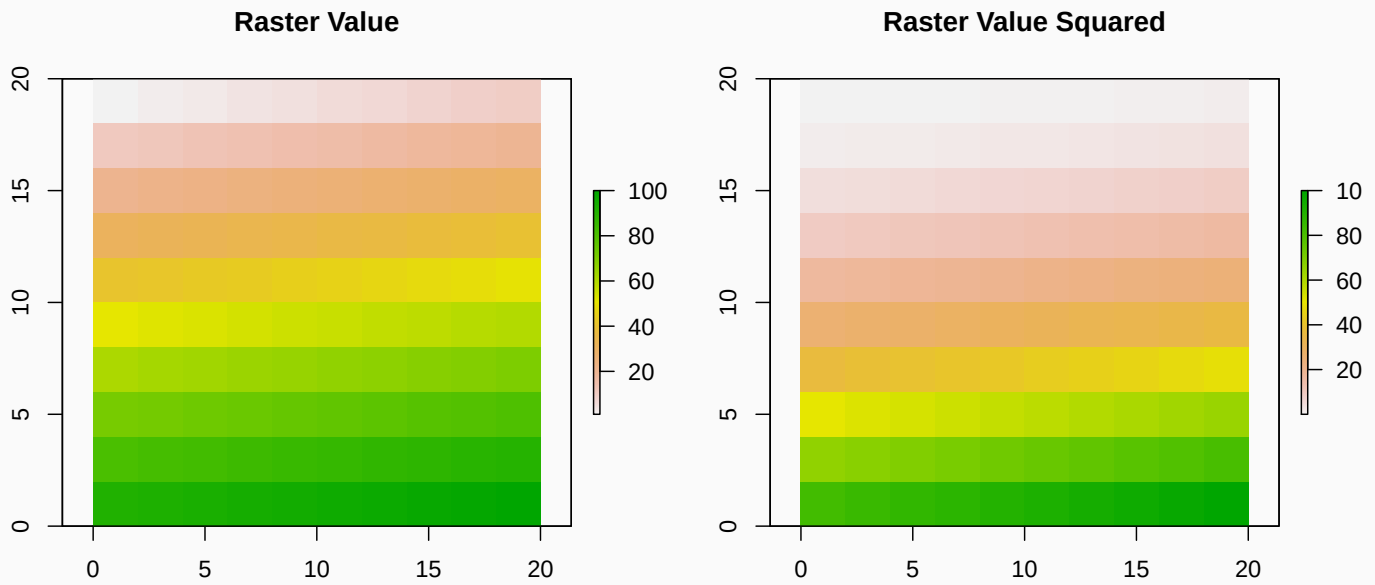
```
myrast <- raster(extent(c(0, 20, 0, 20)), ncol = 10,
    nrow = 10)
myrast[] <- 1:100
myrast
```

```
## class       : RasterLayer
## dimensions  : 10, 10, 100  (nrow, ncol, ncell)
## resolution  : 2, 2  (x, y)
## extent      : 0, 20, 0, 20  (xmin, xmax, ymin, ymax)
## coord. ref. : NA
## data source : in memory
## names       : layer
## values      : 1, 100  (min, max)
```

Raster Layer: Raster Algebra

- Rasters can be added, multiplied etc.

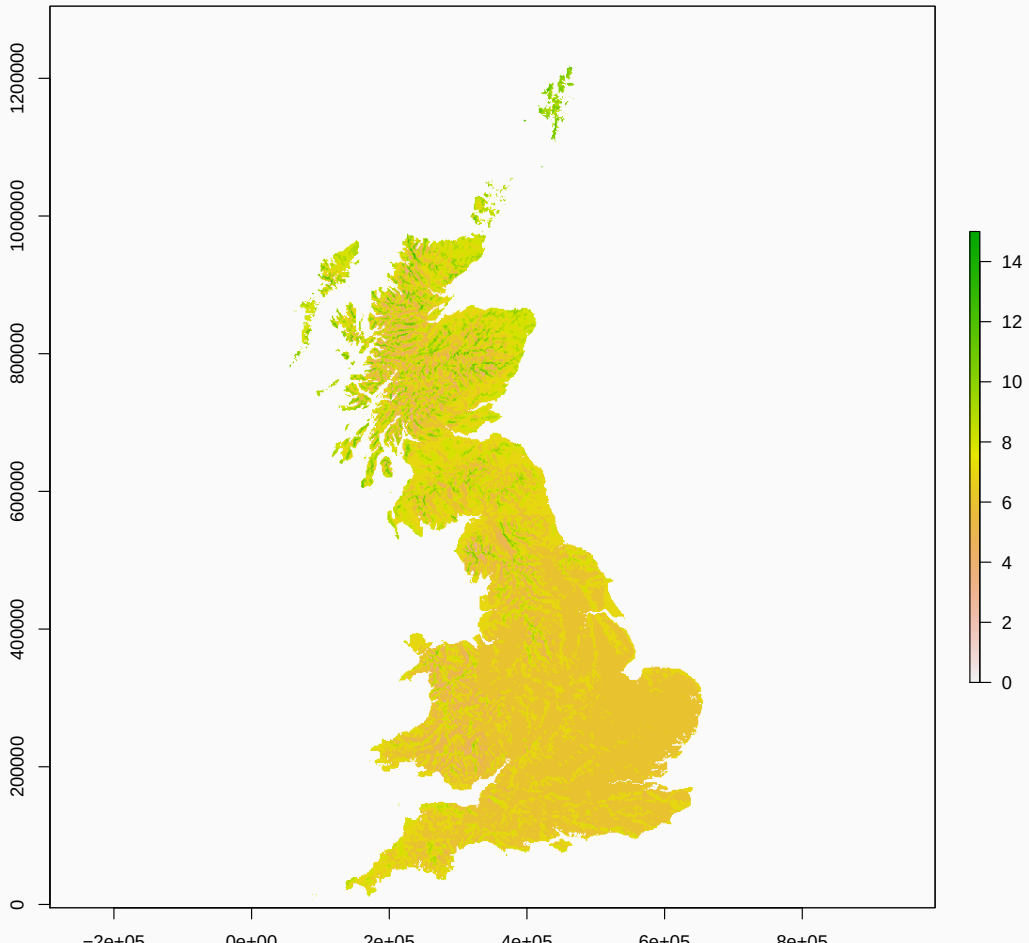
```
par(mfrow = c(1, 2))  # Plot graphs side by side
plot(myrast, main = "Raster Value", asp = 1)
myrastSquared <- myrast^2
plot(myrastSquared, main = "Raster Value Squared",
    asp = 1)
```



Raster Data: Import a GeoTiff

- The `raster()` function from the raster package can be used

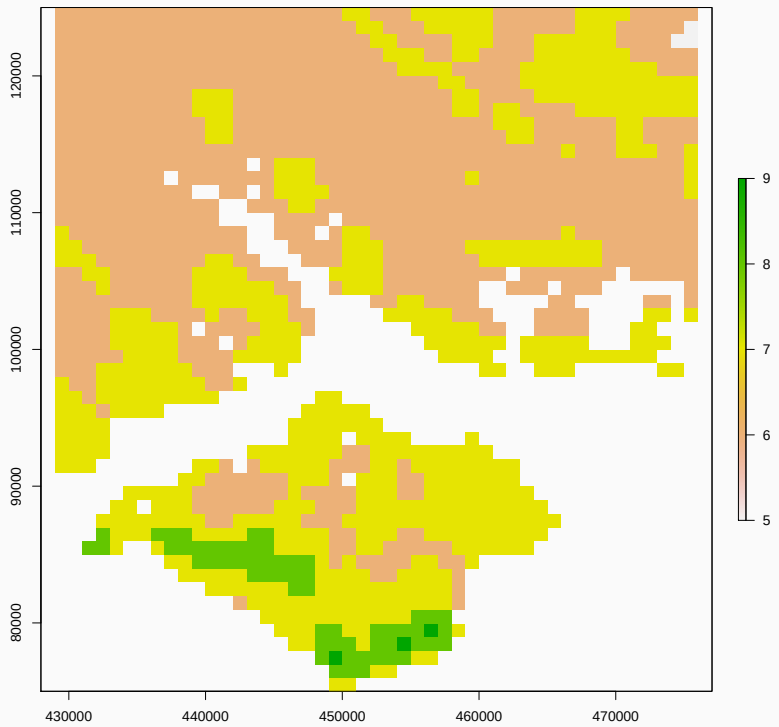
```
WindSpeedRaster <- raster(x = "../data/WindSpeed45.tif")
plot(WindSpeedRaster, "WindSpeed45")
```



Raster Data: Crop a Raster

- `crop()` function can be used to clip a raster to a spatial extent

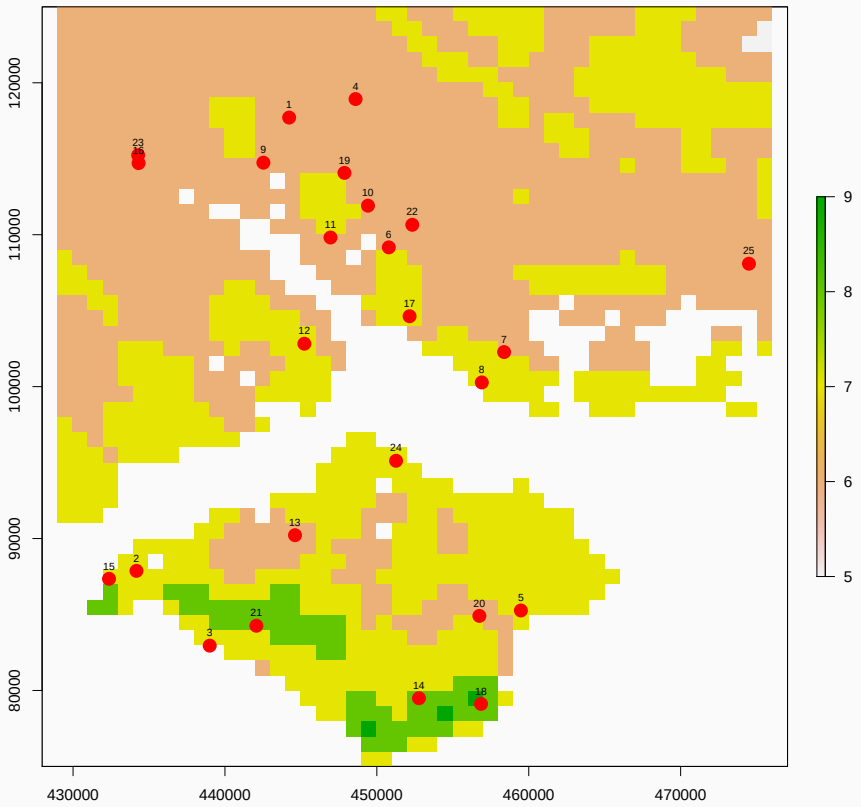
```
WindSpeedSolent <- crop(WindSpeedRaster, Solent)
plot(WindSpeedSolent, asp = 1)
```



Overlay Points and Extract Values

- Generate Some Points Within The Solent Region
- The `spsample()` function from the package sp can to produce a sample within a given extent

```
SolentPoints <- spsample(Solent, n = 25, type = "random")
SolentPoints$ID <- 1:nrow(SolentPoints@coords)
```



Overlay Points and Extract Values

- You can use the `extract()` function to calculate the value at points.

```
extract(WindSpeedSolent, SolentPoints)
```

```
##  [1]  6  7 NA  6  7  6  6  7  6  6  6  7  6  8  7  6  7  9  6  7  8  6  6
## [24]  7  6
```

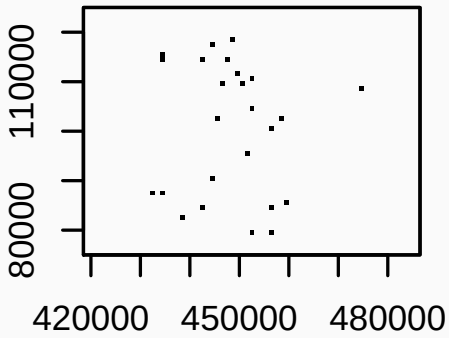
Create a Distance Surface

- The function `distance()` can produce a proximity raster to the nearest spatial point
- We must first `rasterize()` the Vector data (SpatialPoints)

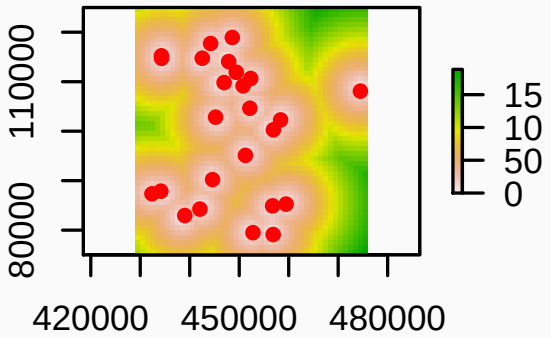
```
SolentPointsRast <- rasterize(SolentPoints, WindSpeedSolent,
    field = 1)
dist2SamplePtsRast <- distance(SolentPointsRast)
```

Original Points

Rasterized Points



Proximity Raster



PART VI: Displaying Results

ggplot: the basics

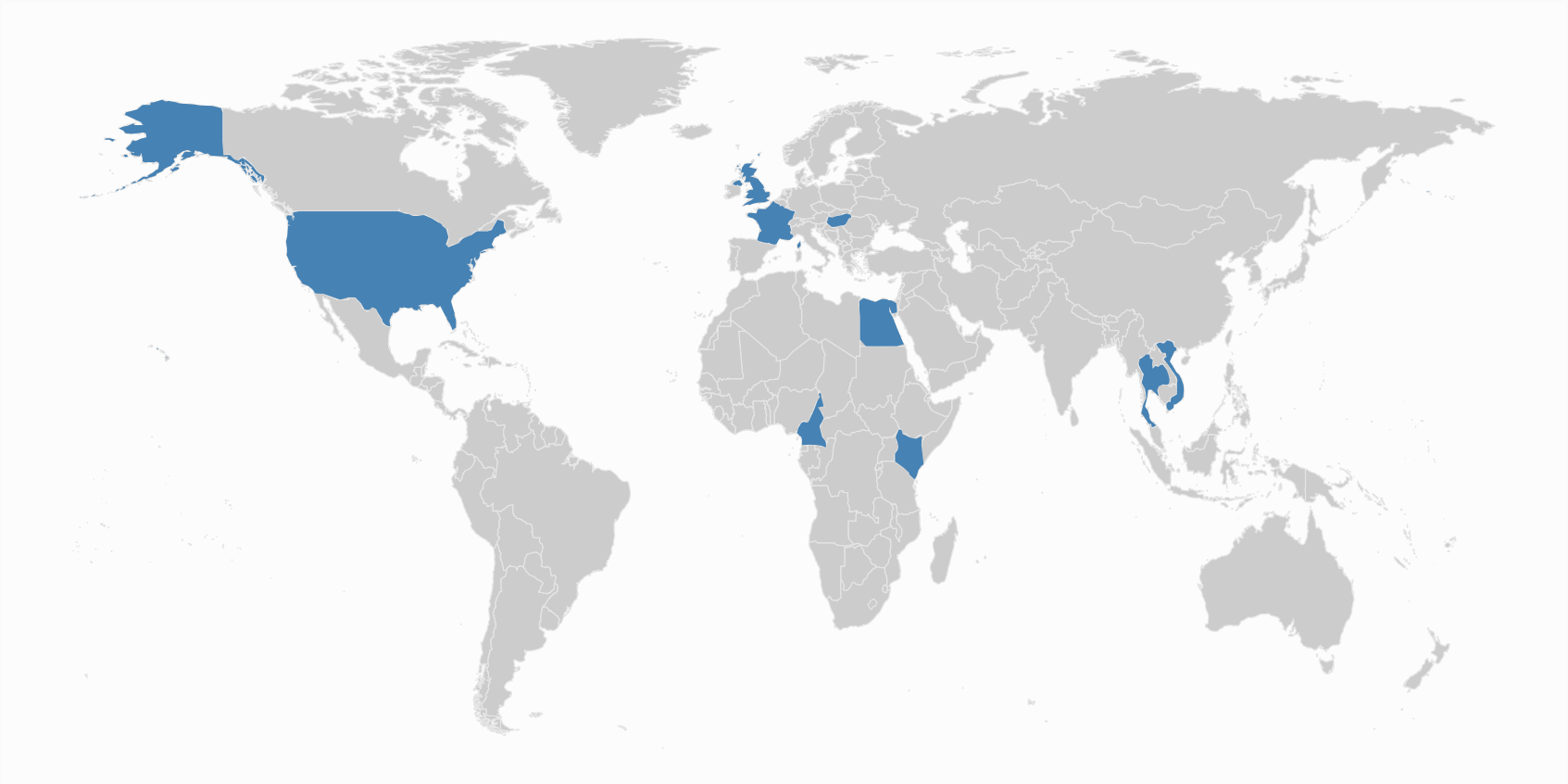
- Maps so far have been quite basic and produced with the functions `plot()` or `spplot()`
- Easy to produce, but not the most attractive
- Time to use ggplot2
- Same procedure as building graphs, with a few minor differences

Warning

Don't worry if you don't understand all the examples. They can be used for future reference.

ggplot Example: A World Map

- Places I have visited.



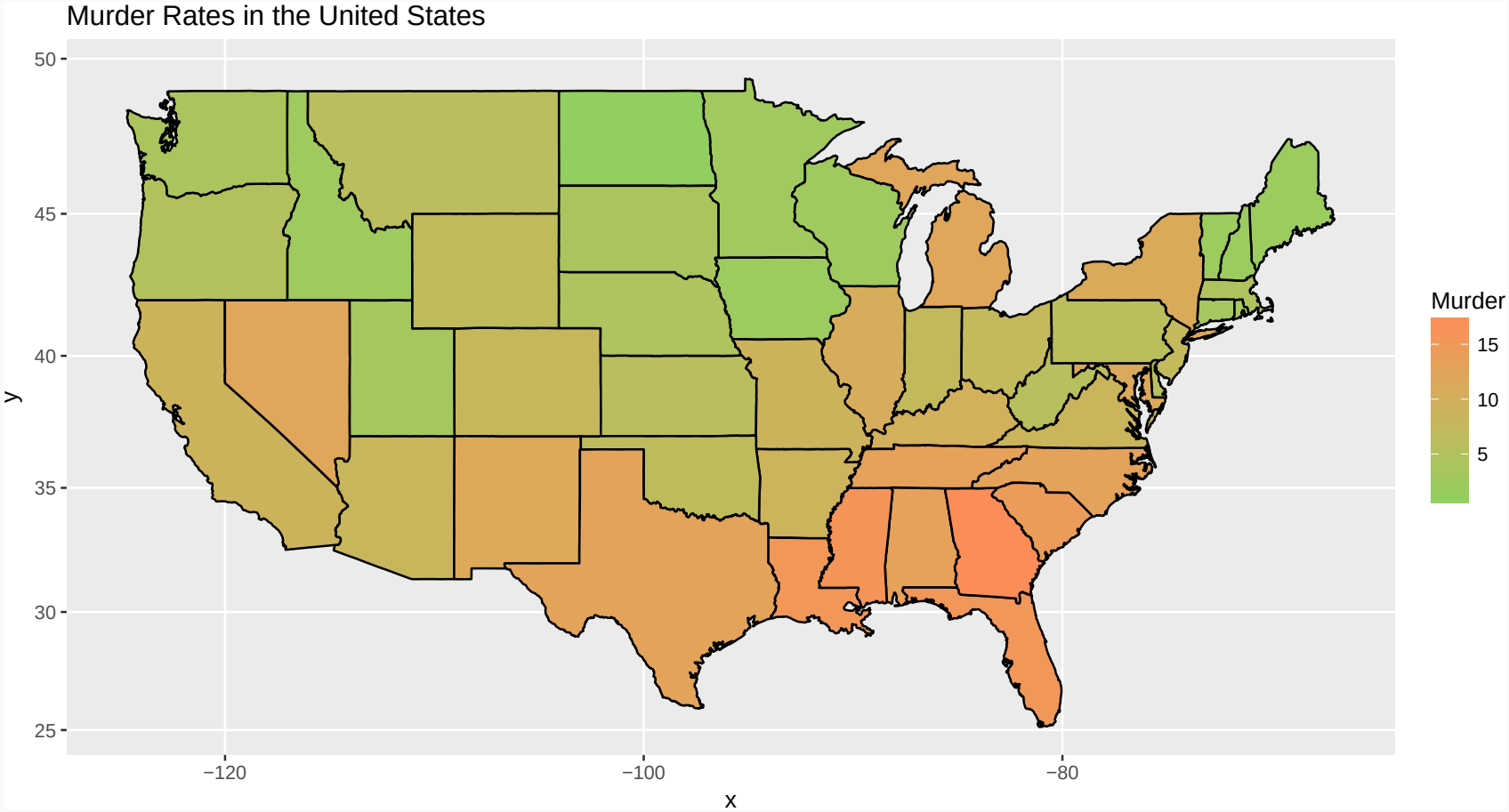
ggplot Example: A World Map

- Map made with less than 15 lines of code.

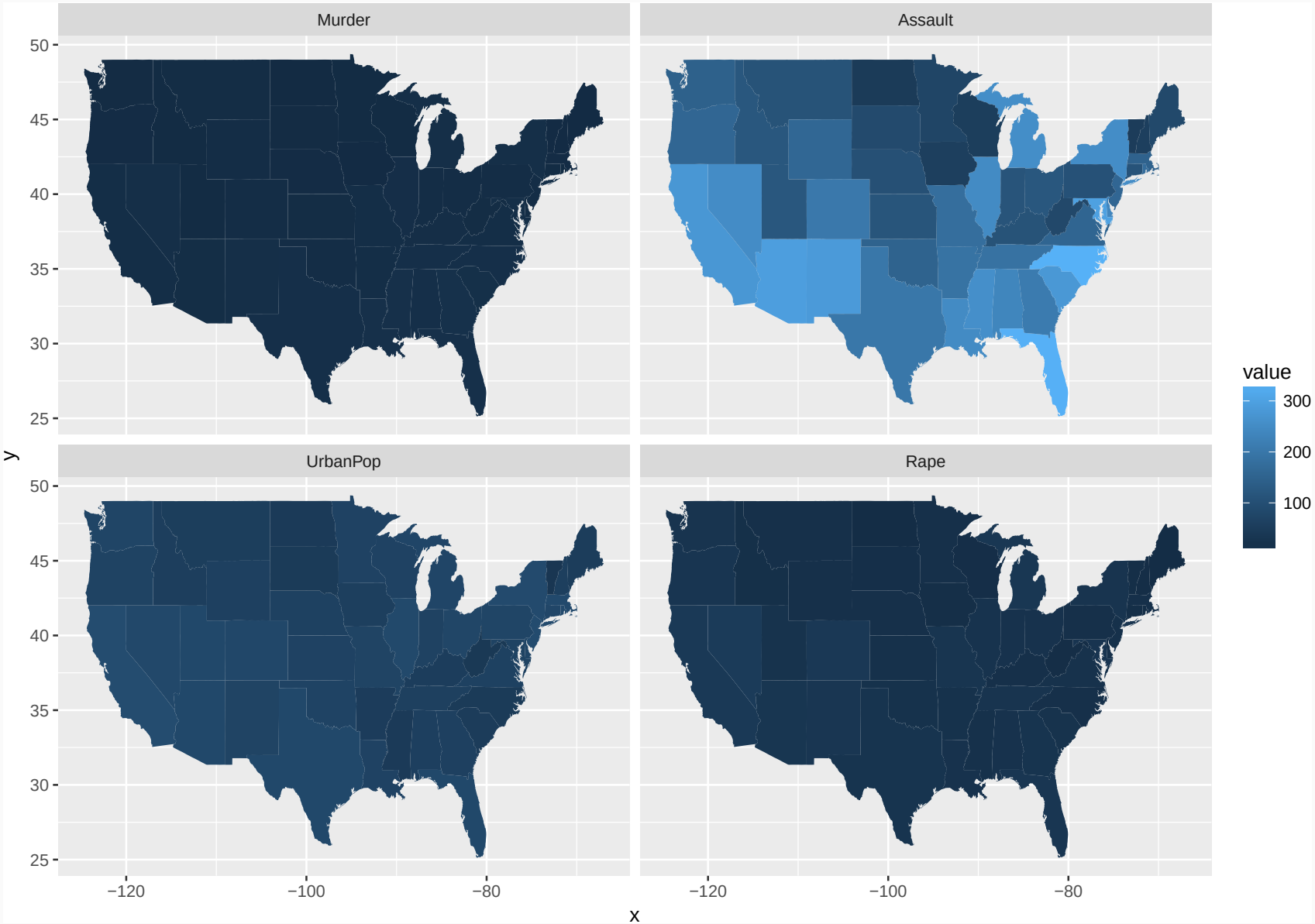
```
library(mapdata)
library(ggmap)

# Load Data and select countries
World <- map_data(map = "world")
World <- World[World$region != "Antarctica", ] # Remove Antartica as it looks bad on the map

# Select Highlight Regions
StudyRegions <- World[World$region == c("UK", "USA", "UAE", "Egypt", "Kenya", "Cameroon",
                                        "Hungary", "France", "Thailand", "Vietnam"),]

ggplot() +
  # Add the basemap
  geom_polygon(data = World, aes(x=long, y = lat, group = group), fill = "grey80", colour = "grey90", size = 0.25) +

  # Add the higlihhted Countries
  geom_polygon(data = StudyRegions, aes(x=long, y = lat, group = group), fill = "steelblue") +

  # Coordinates
    coord_fixed(1.3) +

  # Themes
  theme_nothing() +
  theme(panel.background = element_rect(fill = 'gray99'),
        plot.margin = unit(c(0,0,0,0), "cm"))
```

ggplot Examples: a choropleth map

Another example from: http://docs.ggplot2.org/current/geom_map.html



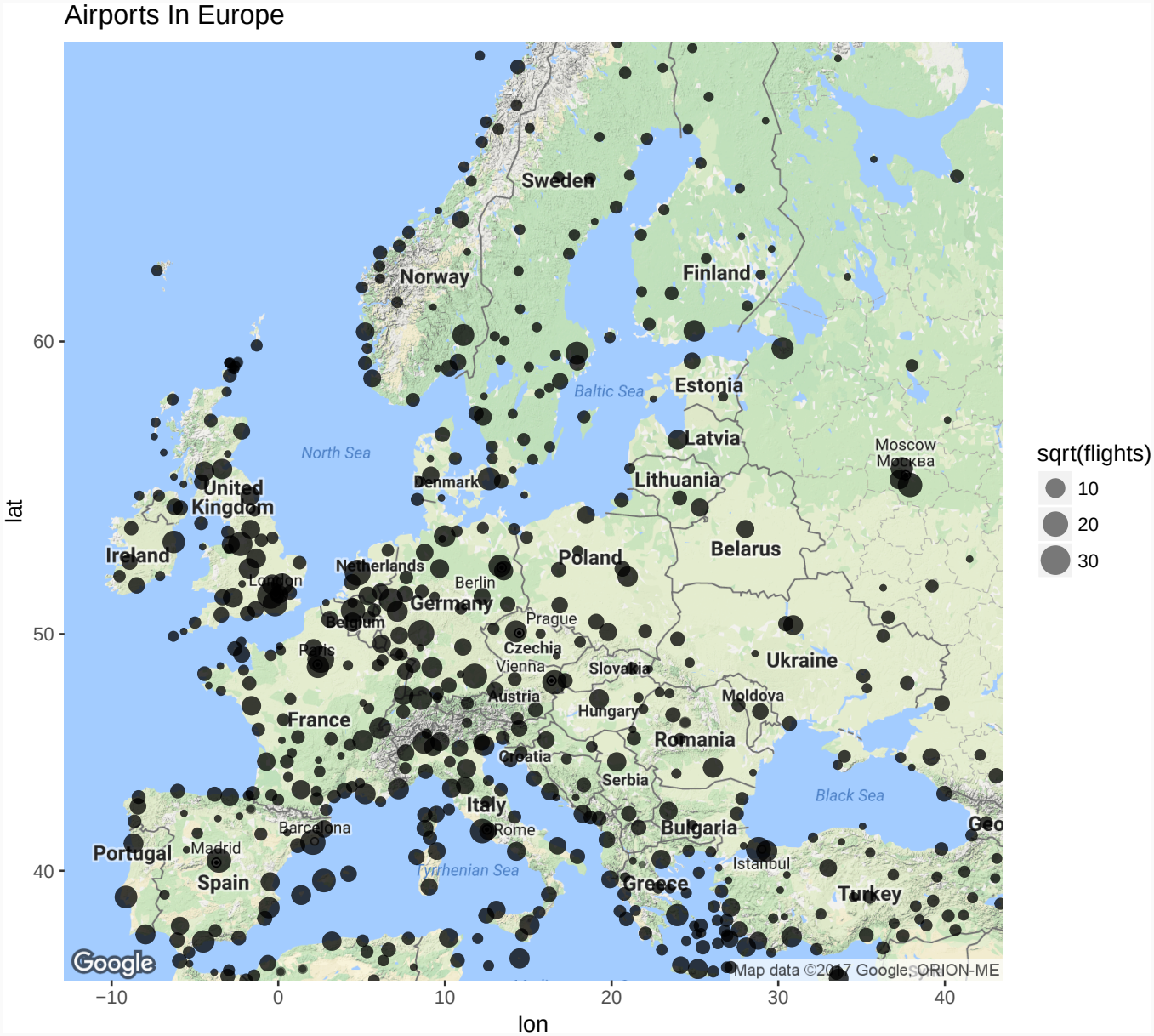
ggplot Examples: a faceted map



ggplot Examples: a choropleth map

```
library(ggplot2)
crimes <- data.frame(state = tolower(rownames(USArrests)),
    USArrests)
library(reshape2)  # for melt
crimesm <- melt(crimes, id = 1)
library(maps)

states_map <- map_data("state")

# Murder Map

# Data & Aesthetics
ggplot(crimes, aes(map_id = state, fill = Murder)) +

# Geometries
geom_map(aes(fill = Murder), map = states_map, colour = "black") +

scale_fill_gradient(low = "#91cf60", high = "#fc8d59") +

# Coordinates
expand_limits(x = states_map$long, y = states_map$lat) +
    coord_map() +
# Theme
ggtitle("Murder Rates in the United States")


# Faceted Map
ggplot(crimesm, aes(map_id = state)) + geom_map(aes(fill = value),
    map = states_map) + expand_limits(x = states_map$long,
    y = states_map$lat) + facet_wrap(~variable)
```

ggplot Examples: spatial Points



ggplot Examples: spatial Points

```r
#data
airports <- read.csv("https://raw.githubusercontent.com/jpatokal/openflights/master/data/airports.dat", header = FALSE)
colnames(airports) <- c("ID", "name", "city", "country", "IATA_FAA", "ICAO", "lat", "lon", "altitude", "timezone", "DST")
routes <- read.csv("https://raw.githubusercontent.com/jpatokal/openflights/master/data/routes.dat", header=F)
colnames(routes) <- c("airline", "airlineID", "sourceAirport", "sourceAirportID", "destinationAirport",
                      "destinationAirportID", "codeshare", "stops", "equipment")

library(plyr)
departures <- ddply(routes, .(sourceAirportID), "nrow")
names(departures)[2] <- "flights"
arrivals <- ddply(routes, .(destinationAirportID), "nrow")
names(arrivals)[2] <- "flights"

airportD <- merge(airports, departures, by.x = "ID", by.y = "sourceAirportID")
airportA <- merge(airports, arrivals, by.x = "ID", by.y = "destinationAirportID")

map <- get_map(location = 'Europe', zoom = 4, messaging = FALSE)

# create the data set containing both departures and arrivals
airportD$type <- "departures"
airportA$type <- "arrivals"
airportDA <- rbind(airportD, airportA)

ggmap(map) +
  geom_point(aes(x = lon, y = lat, size = sqrt(flights)), data = airportDA, alpha = .5) +
# panels according to type (departure/arrival) +
  ggtitle("Airports In Europe") +
  guides(fill=guide_legend(title="New Legend Title"))
```

Conclusions

When to Use GIS vs. R?

GIS

- no mapping experience
- one-off maps
- serious cartography
- your data live in a geodatabase

R

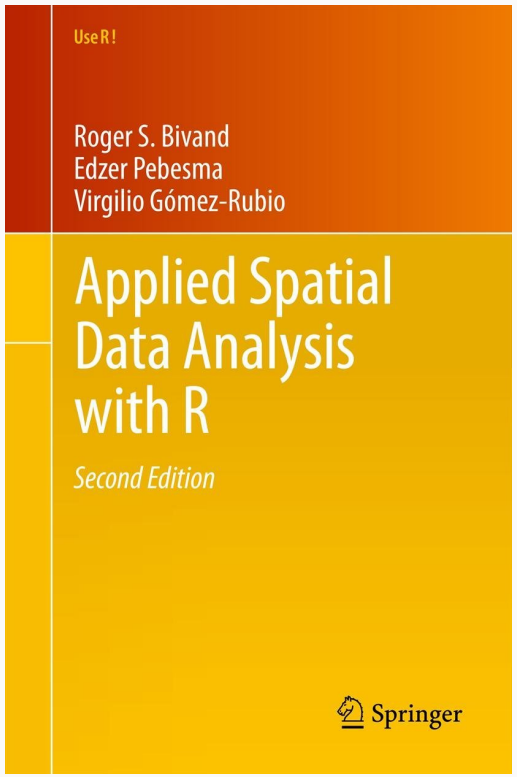
- no money for commercial software
- need repeated analysis
- integration with other analyses
- development of new methods
- working with large raster files

Additional Resources

Overview of Spatial Packages

- http://cran.r-project.org/web/views/Spatial.html

Books



Additional Resources

Tutorials

- http://rpubs.com/RobinLovelace/11869
- http://science.nature.nps.gov/im/datamgmt/statistics/r/advanced/spatial.cfm
- http://spatial.ly/r/
- http://gis.stackexchange.com/questions/45327/tutorials-to-handle-spatial-data-in-r
- https://pakillo.github.io/R-GIS-tutorial/

R Spatial Cheatsheet

- http://www.maths.lancs.ac.uk/~rowlings/Teaching/UseR2012/cheatsheet.html
- https://github.com/wegmann/RSdocs/releases/download/May2014_spatial/AniMove_refcard.pdf

Summary

1. Help you get to the point where you can do basic analysis or visualization
2. Point out some of the commonly-used packages for spatial data manipulation and analysis
3. Be a resource you can come back to
4. Provide Guidance on useful resources to use after the course

Hopefully you have ticked a few of these boxes!

Thank You for Listening!