Geospatial Data and Mapping in R

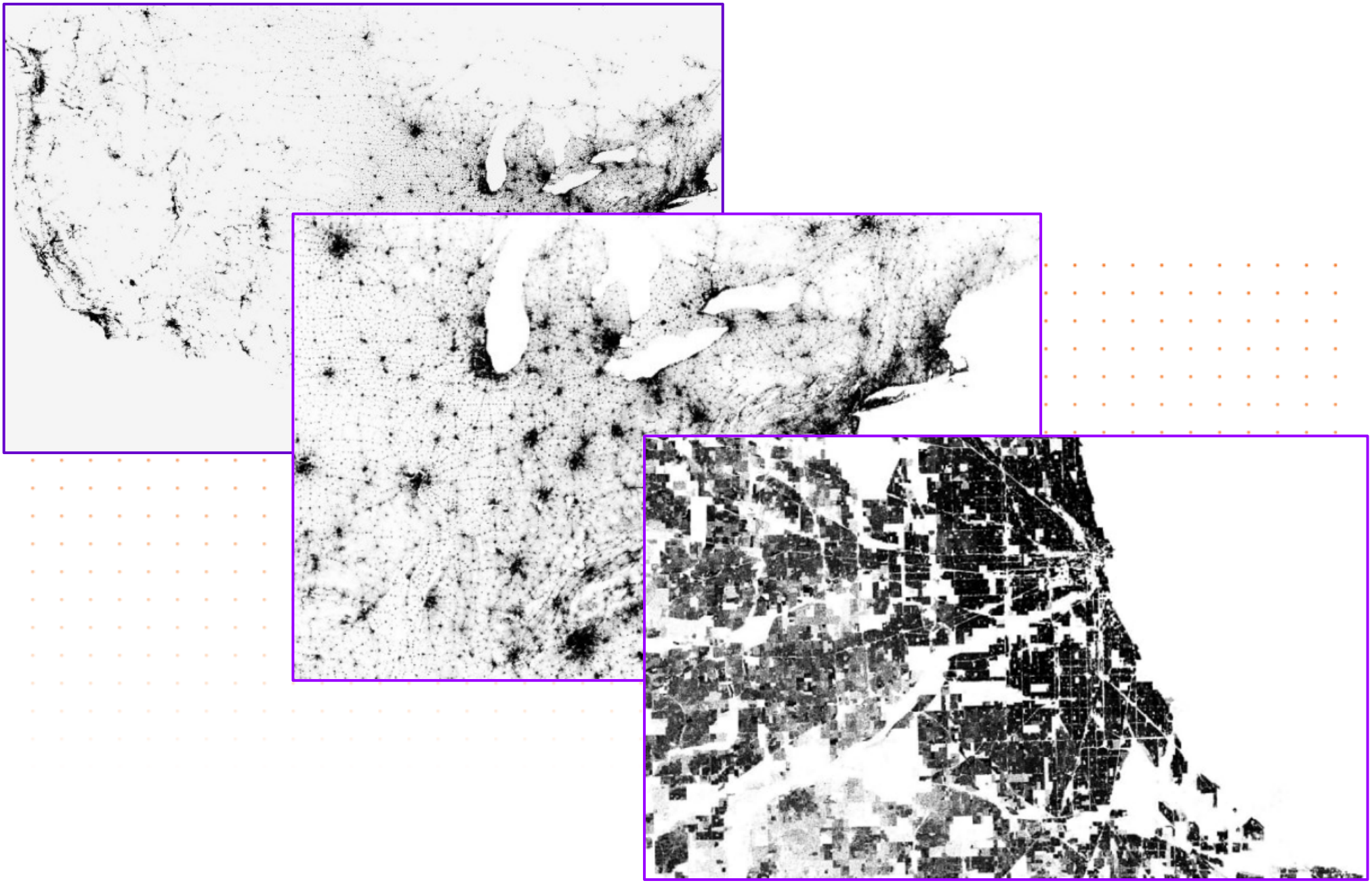
DohaR MeetUp

19-May-2017

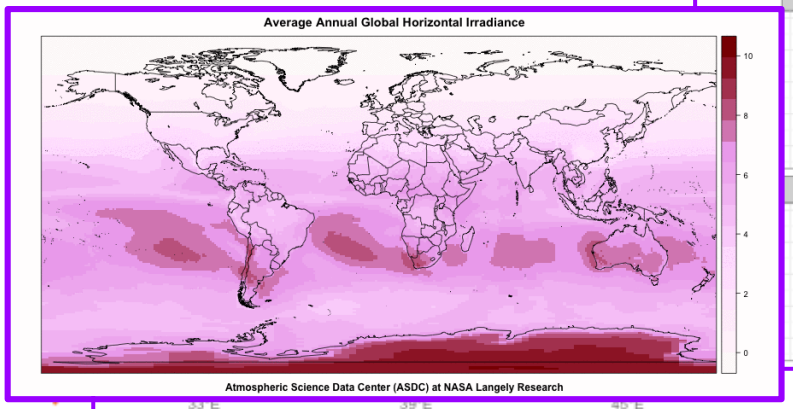
Bradley J Horn

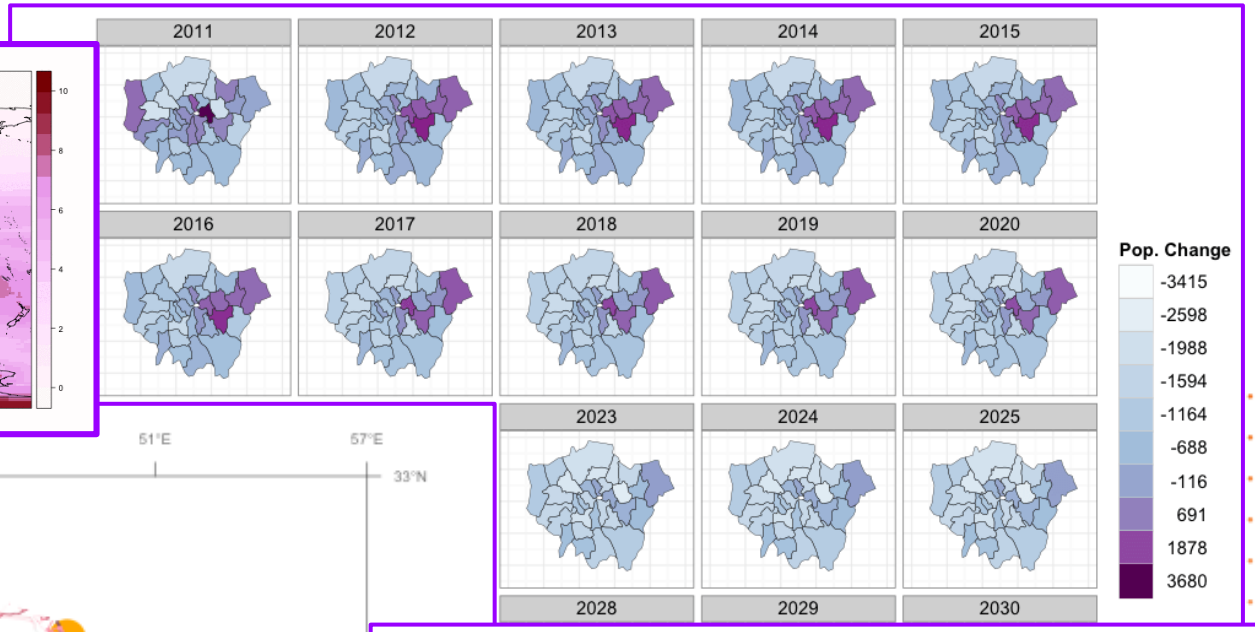
bxhorn@gmail.com

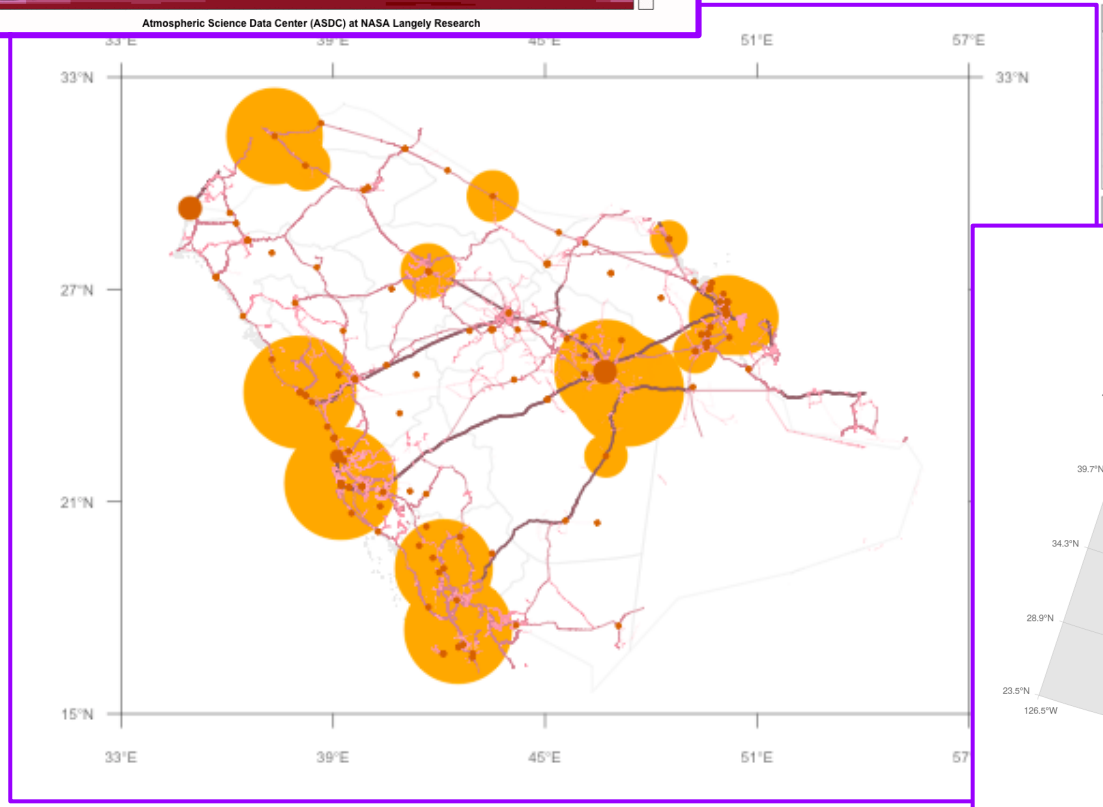
Dot Maps in R

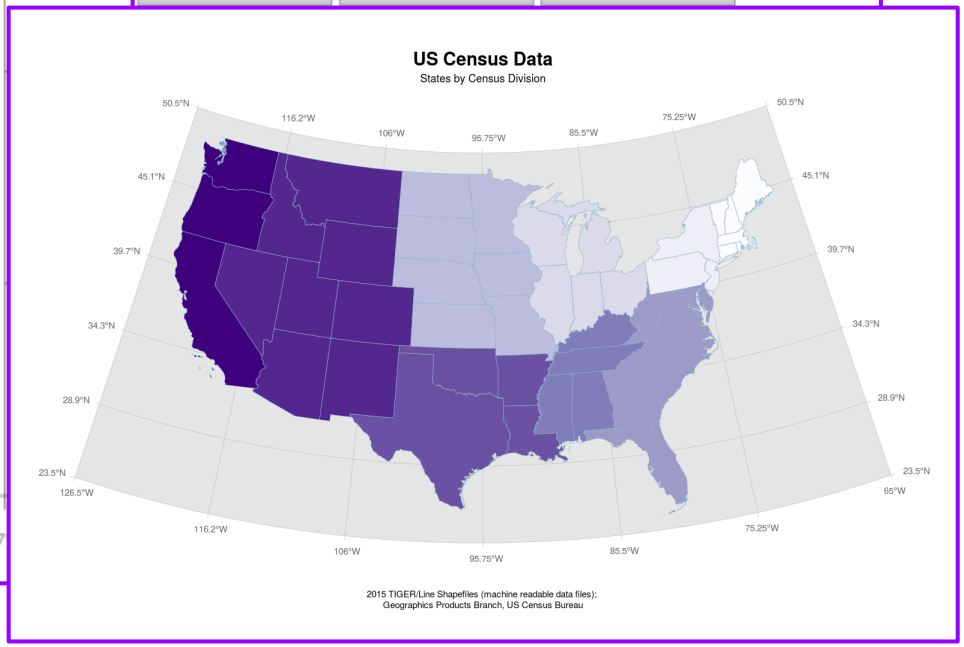


Shapefile Maps in R

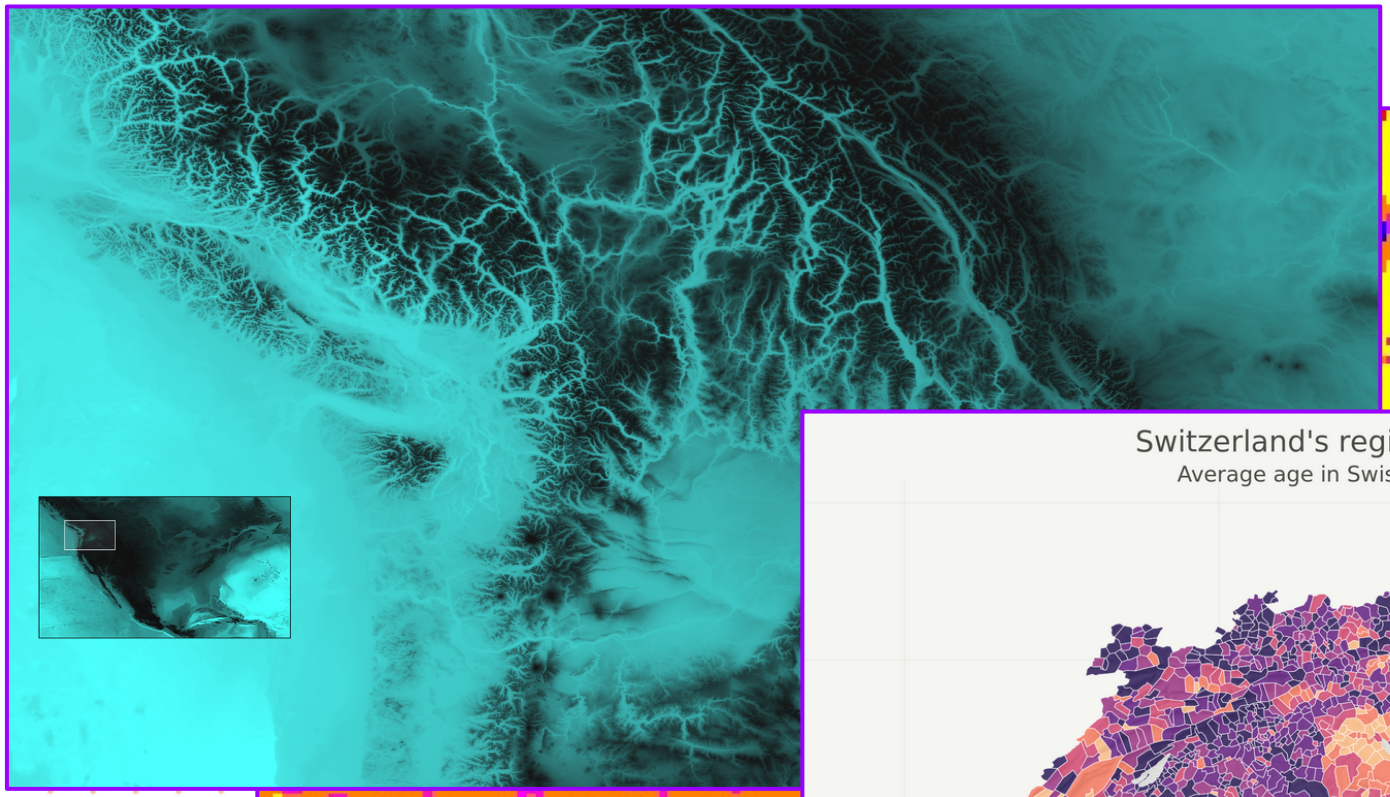


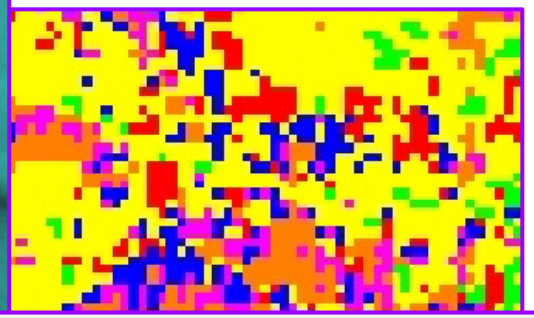


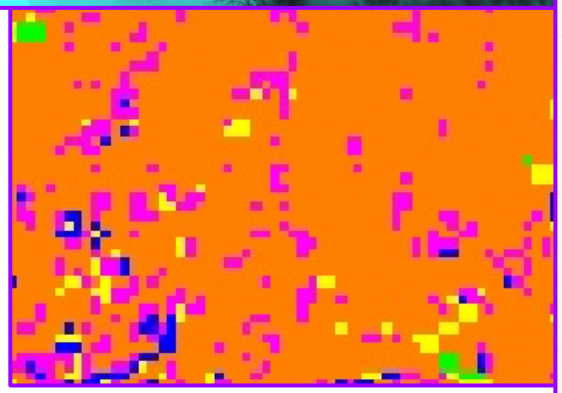


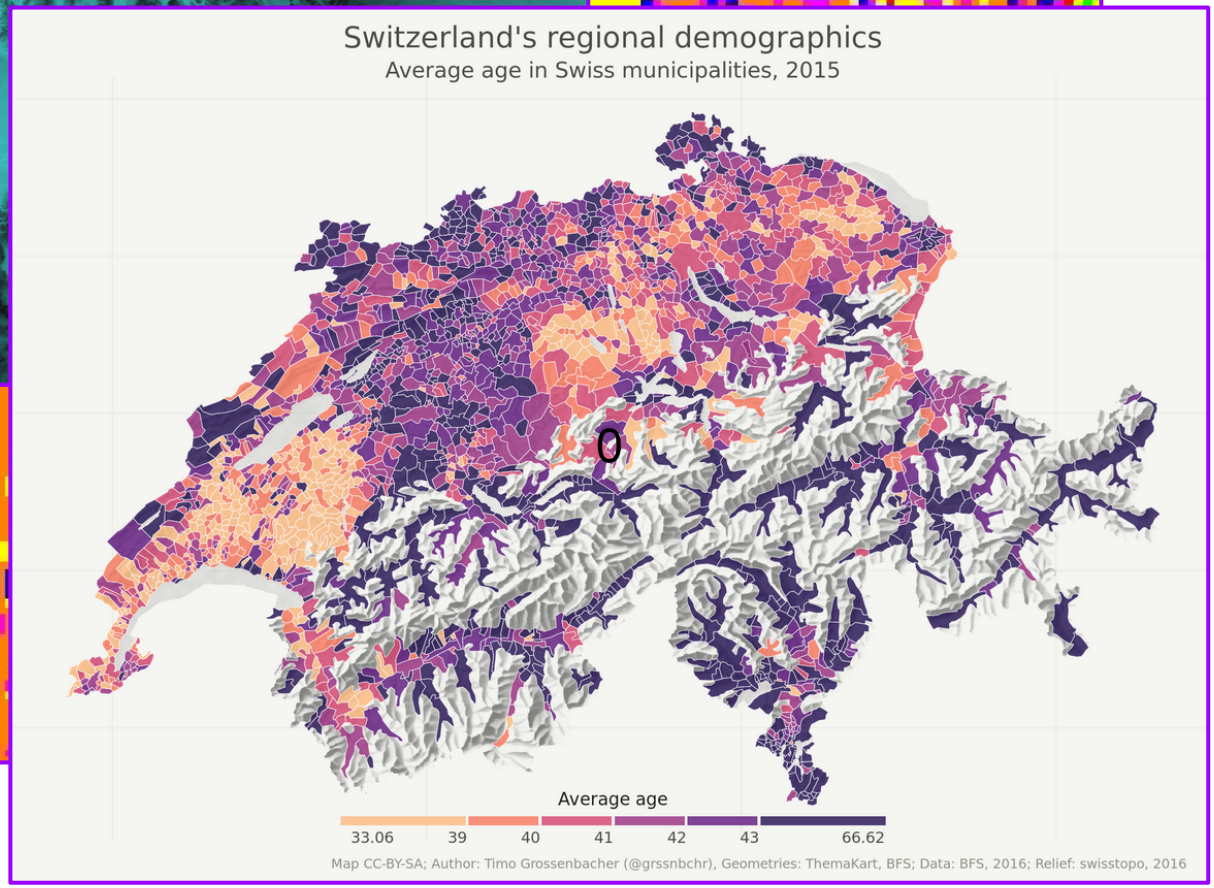


Raster Maps in R

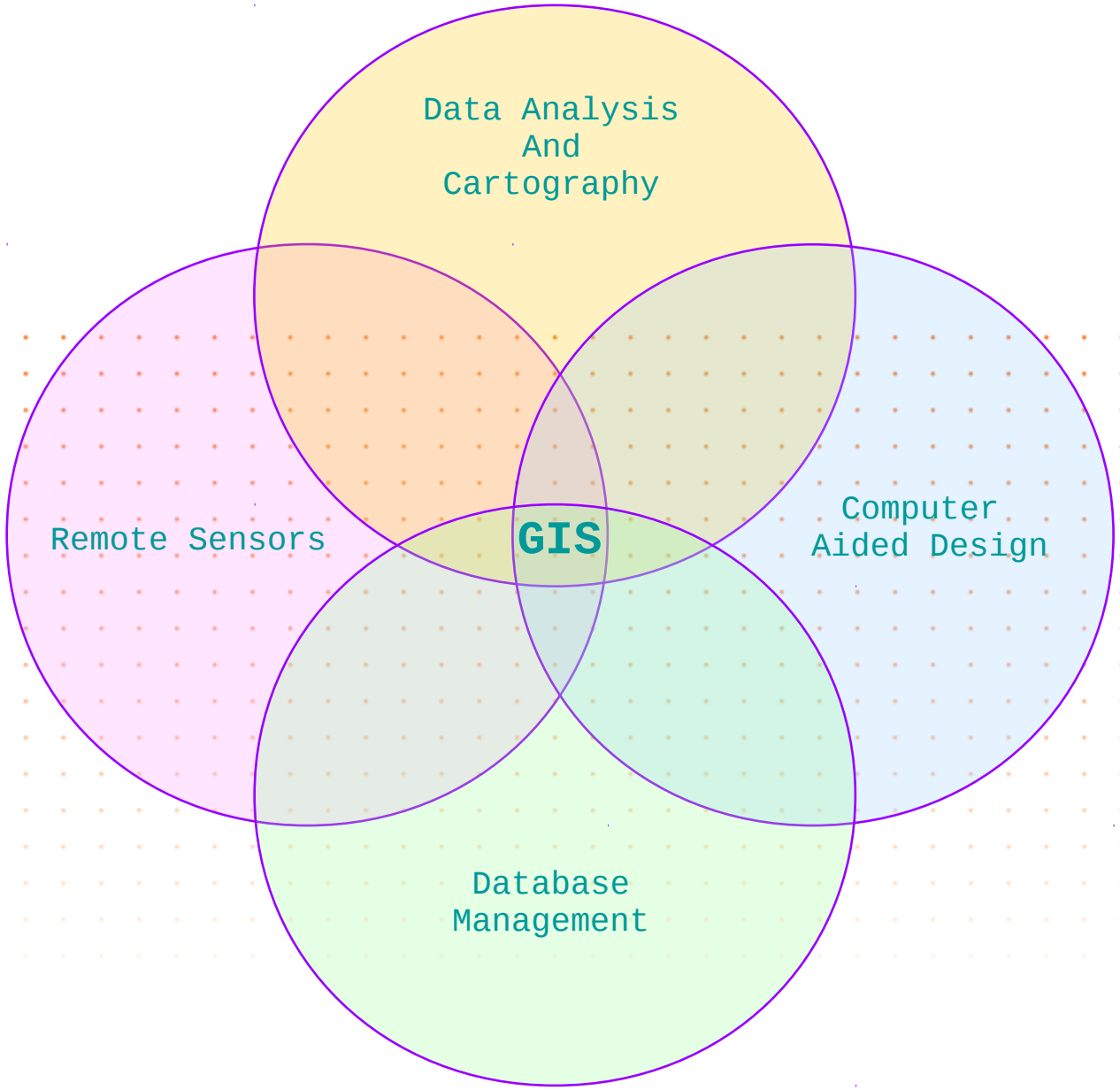








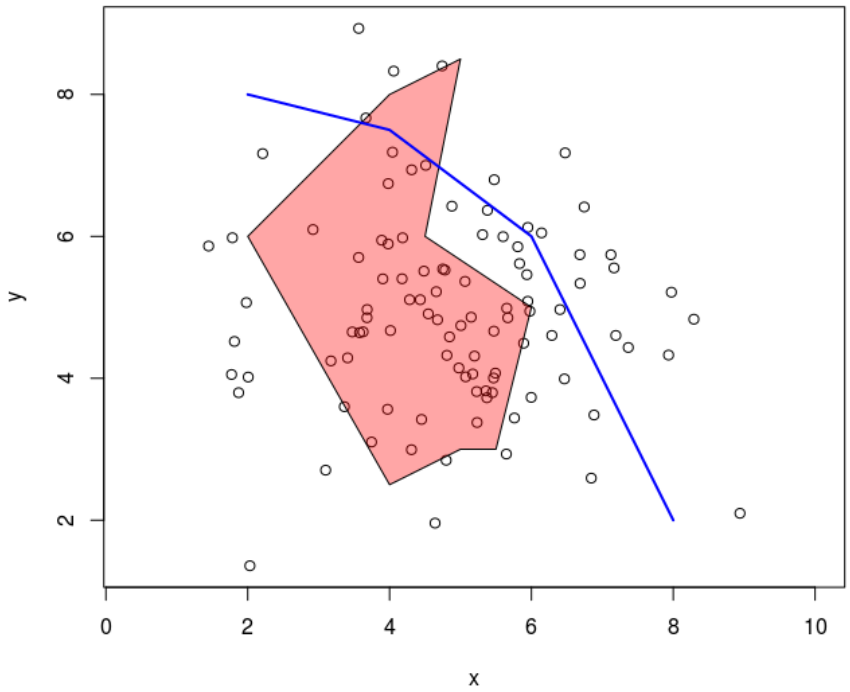
What is GIS?


Data Analysis
And
Cartography
Remote Sensors
GIS
Computer
Aided Design
Database
Management

Geospatial Analysis

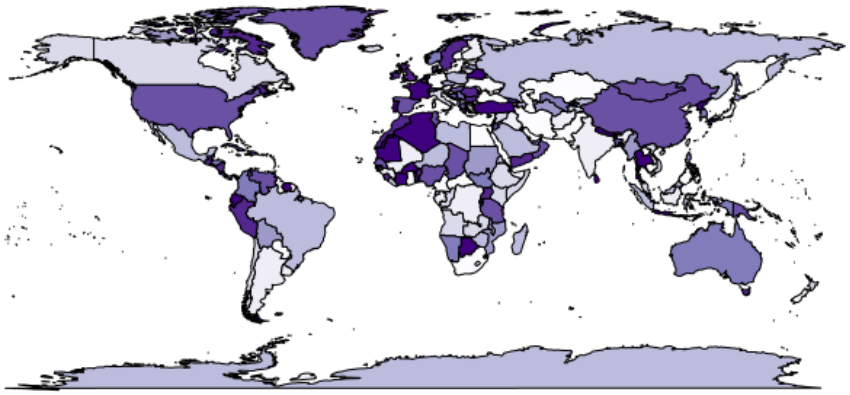
- Location: where is … ?
- Condition: what is at … ?
- Trend: what changed at … ?
- Routing: which way … ?
- Classification: what is the pattern … ?
- Modelling: why and what if … ?
- Engineering: where and how … ?

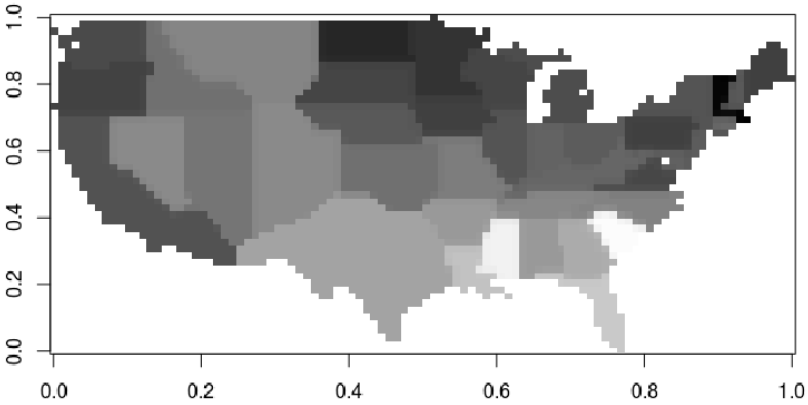
Map Elements in Base R




```
# Points
x <- rnorm(100, 5, 1.5)
y <- rnorm(100, 5, 1.5)
xy <- tibble(x = x, y = y)
plot(xy, asp = 1)

# Lines
x <- c(2, 4, 6, 8)
y <- c(8, 7.5, 6, 2)
xy <- tibble(x = x, y = y)
lines(xy, lwd = 2, col = "blue", add = TRUE)

# Polygons
x <- c(2, 4, 5, 4.5, 6, 5.5, 5, 4, 2)
y <- c(6, 8, 8.5, 6, 5, 3, 3, 2.5, 6)
xy <- tibble(x = x, y = y)
polygon(xy,  col = rgb(1,0,0,0.35), add = TRUE)
```

Maps package (1995)

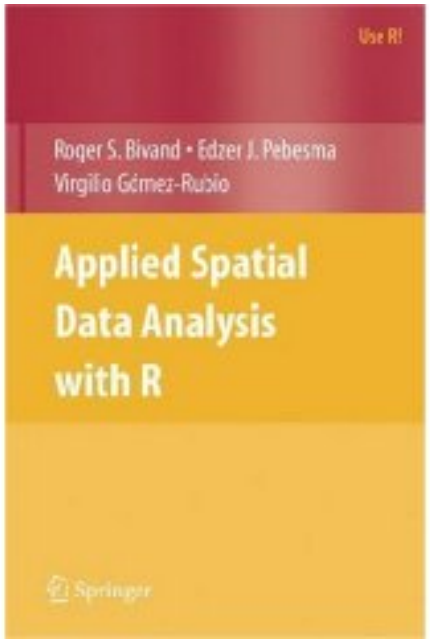





```
library(maps)

# World map
cols <- brewer.pal(9, "Purples")
map("world", fill = TRUE, col = cols)

# US states with a demographic matrix
data(state, package = "datasets")
data(votes.repub)
Z <- votes.repub[, "1900"]
M <- map("state", fill = TRUE, plot = FALSE)

Fit <- smooth.map(m, z, span = 1/100, merge = TRUE,
                  ave = TRUE)
Mat <- tapply(fit$z, fit[1:2], mean)
gray.colors <- function(n) gray(rev(0:(n - 1))/n)

# plot integrated map and geo data
image(mat, col = gray.colors(100))
```

YUCK!

Vector Data – the `sp` classes (2004)



OGC "Simple Features" applied

- Spatial Points
- SpatialLines
- SpatialPolygons
- SpatialGrids
- SpatialPixels

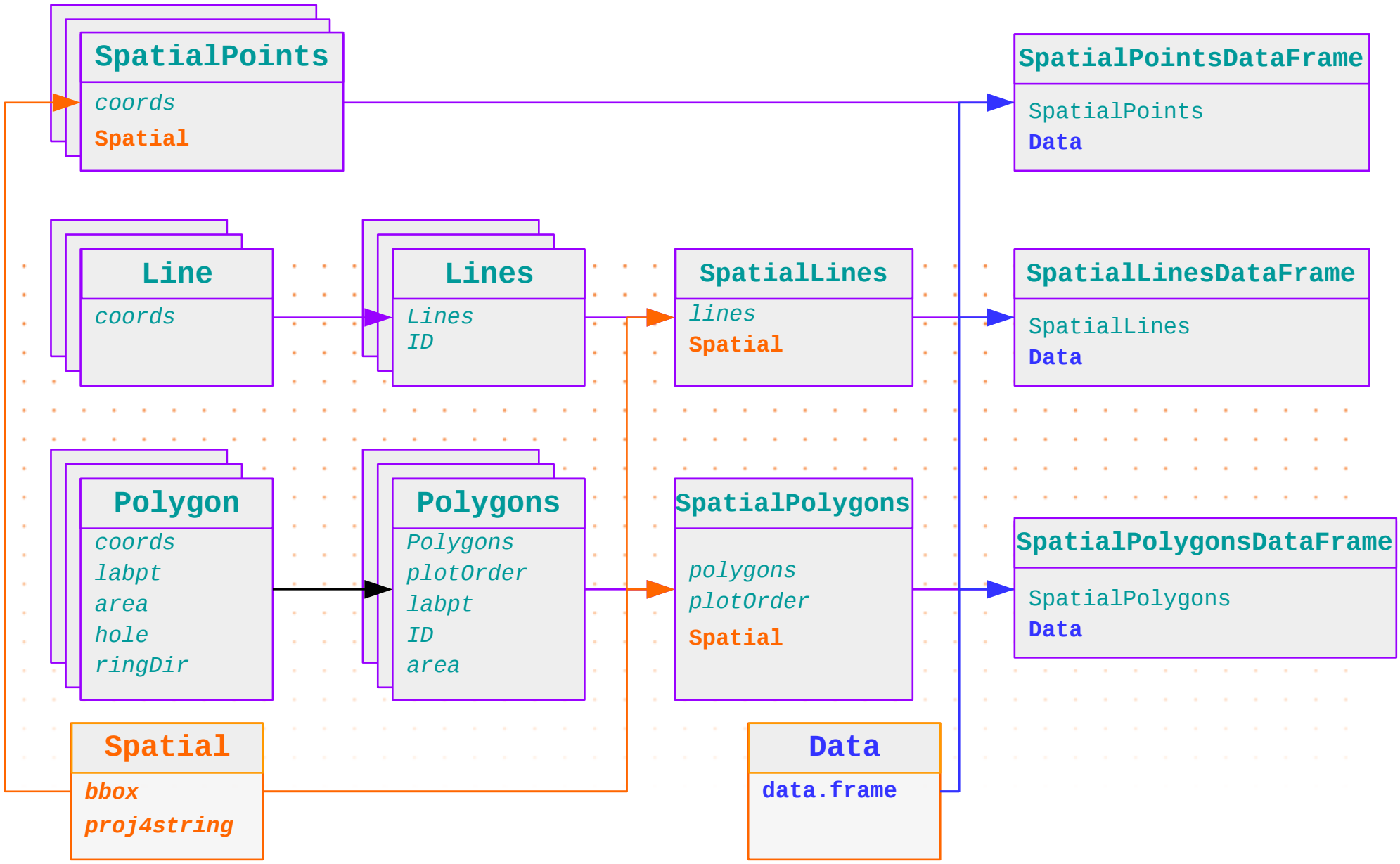
Data object classes and methods for spatial features

`data.frame` extensions for feature data

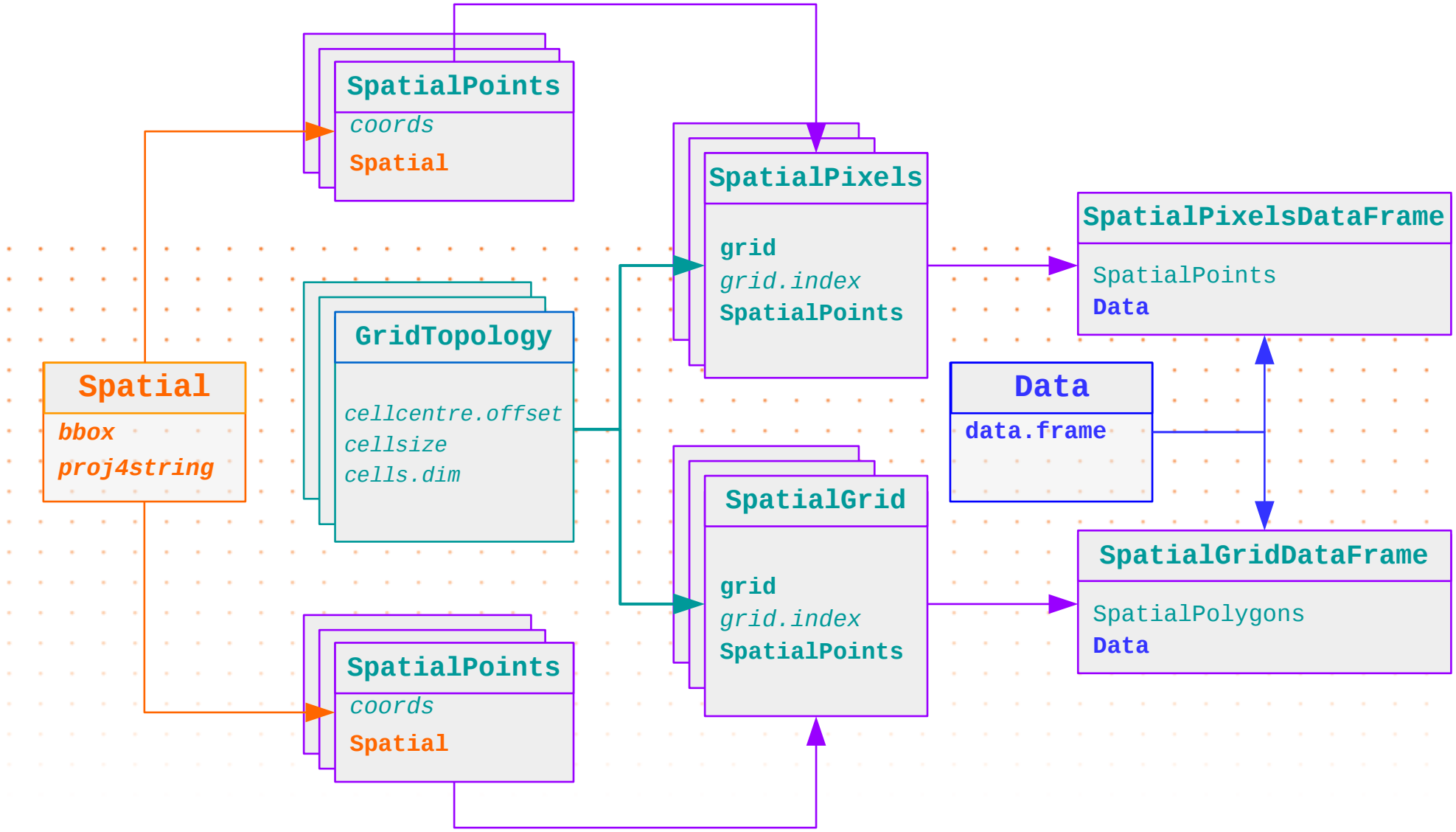
Standard PROJ4 datum and map projections

```
OGC = Open Geospatial Consortium
```

Classes and Slots – Shapefile Objects



Classes and Slots - Raster Objects



R Packages by Need

GDAL/OGR

- Read vector and raster datasets
- Wrapped by **`rgdal`** package
- Used by **`sp`** and **`raster`** packages

PROJ4: Cartographic Projections

- Set and change coordinate reference systems
- **`sp:CRS`** and **`sp:spTransform`**

Spatial Data Classes and Methods for R

- **`sp`** and **`raster`** packages

GEOS: Geometry Engine Open Source

- GIS data manipulation functions
- Wrapped by **`rgeos`**

```
GDAL = Geospatial Data Abstraction Library
OGR = OGC OpenGIS Simple Features Reference
```

Packages

```
# spatial libraries
library(rgdal)                  # bindings for the Geospatial Data Abstraction Library
library(sp)                     # classes and methods for spatial data
library(rgeos)                  # interface to GEOS: geometry engine open source
library(raster)                 # geographic data analysis and modelling
library(maptools)               # tools for reading and handling spatial objects
library(spdplyr)                # data manipulation for spatial classes
library(spbabel)                # convert sp data to tidy tables

# spatial visualization
library(gridExtra)              # misc graphics functions
library(ggplot2)                # Grammar of Graphics
library(ggthemes)               # plot themes for ggplot
library(ggmap)                  # spatial visualization with ggplot2
library(rasterVis)              # visualization methods for raster data
library(graticule)              # meridian and parallel lines for maps
library(RColorBrewer)           # colour ramps
library(viridis)                # gradient colours

# spatial data
library(OpenStreetMap)          # access OSM raster images
library(tigris)                 # api for US Census TIGER database
```

Create Points

```
# define point coordinates
coords <- cbind(x, y)

# go spatial / create SpatialPoints object
sp <- SpatialPoints(coords)

# create SpatialPointsDataFrame
spdf <- SpatialPointsDataFrame(sp, data)
spdf <- SpatialPointsDataFrame(coords, data)

# alternatively, coerce a data.frame to spatial
coordinates(data) <- cbind(x, y)
coordinates(data) <- ~lon + lat

# coerce back to data.frame
as.data.frame(data)
data <- spdf@data

# interogate spatial data objects
class(spdf)
slotNames(spdf)
str(spdf, max.level = 3)
bbox(spdf)
head(spdf@data)
plot(spdf)
```

Create Lines

```
# define line coordinates
c1 <- cbind(x, y)
c2 <- cbind(x, y)
c2 <- cbind(x, y)

# create line arcs through the designated points
L1 <- Line(c1)
L2 <- Line(c2)
L2 <- Line(c3)

# create lists or clusters or lines
Ls1 <- Lines(list(L1), ID = “a”)
Ls2 <- Lines(list(L2, L3), ID = “b”)

# go spatial / convert to sp class
SL1 <- SpatialLines(list(Ls1))
SL12 <- SpatialLines(list(Ls1, Ls2))

# extend to spatial data.frame
SLDF <- SpatialLinesDataFrame(SL12, data.frame(Z = c(“Road”, “River”),
                                row.names = c(“a”, “b”)))

# interrogate spatial data objects
slotNames(SLDF)
str(SLDF, max.level = 2)
head(SLDF@data)

# get line lengths and IDs
SpatialLinesLength(SLDF)
sapplyslot(SLDF, "lines"), function(x) slot(x, "ID"))
```

Create Polygons

```
# create a polygon ring and convert to spatial class
c1 <- cbind(x, y)
r1 <- rbind(c1, c1[1, ])
P1 <- Polygon(r1)
Ps1 <- Polygons(list(P1), ID = “a”)

# create a double ring and convert
c2a <- cbind(x2a, y2a)
r2a <- rbind(c2a, c2a[1, ])
c2b <- cbind(x2b, y2b)
r2b <- rbind(c2b, c2b[1, ])

P2a <- Polygon(r2a)
P2b <- Polygon(r2b)
Ps2 <- Polygons(list(P2a, P2b), ID = “b”)

# go spatial / convert to sp class
SPs <- SpatialPolygons(list(Ps1, Ps2))

# extend to spatial data.frame
SPDF <- SpatialPolygonsDataFrame(SPs, data.frame(N = c(“one”, “two”),
                                 row.names = c(“a”, “b”)))

# create ring polygon with hole
hc1 <- cbind(x, y)
hr1 <- rbind(hc1, hc1[1, ])
H1 <- Polygon(r1, hole = TRUE)
P1h <- Polygons(list(P1, H1, ID = “c”)
SP1h <- SpatialPolygons(list(P1h))
```

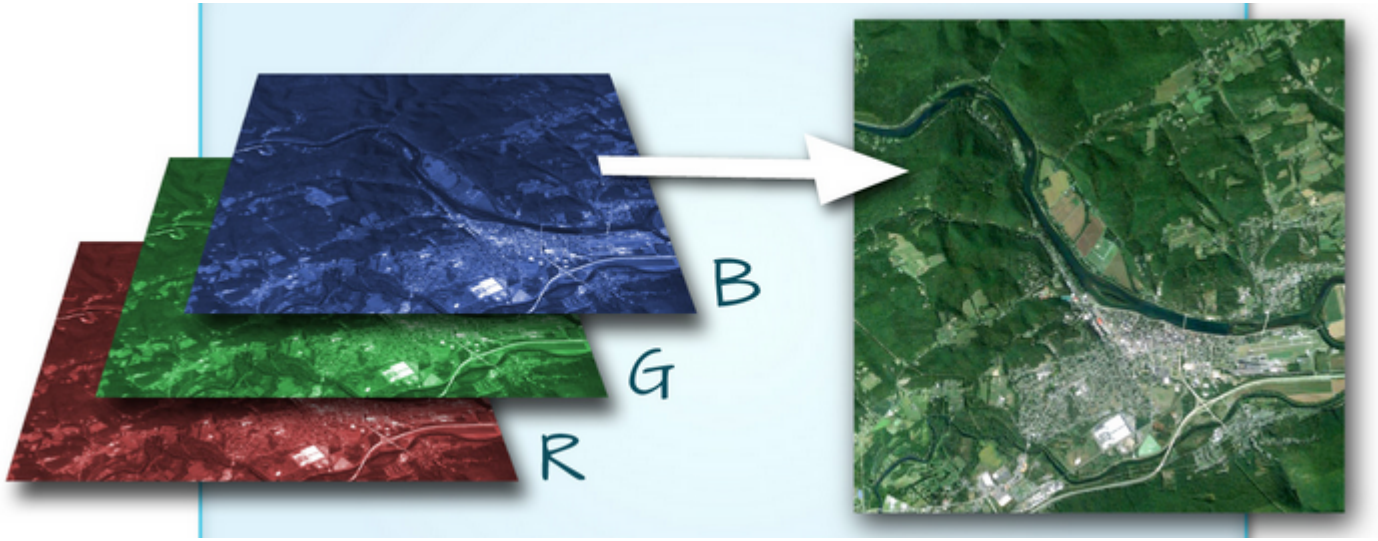
Create Raster Grid

```
# requires 3D matrix or aaray values
r1 <- raster(list(x = x, y = y, z = z)

# get row and column values using standard extraction
r1[3, ]
r1[ ,2]

# interrogate raster extent
extent(r1)

     ## class       : Extent
     ## xmin        : 22.95
     ## xmax        : 29.45
     ## ymin        : 44
     ## ymax        : 46.3

# create empty raster and then fill with z-values
r2 <- raster(nrows = nrows, ncols = ncols, xmin = xmin,
             Xmax = xmax, ymin = ymin, ymax = ymax)
r2[] <- runif(nrows * ncols)

# create raster from extent and then set values
r3 <- raster(extent(r2), nrows = nrows, ncols = ncols)
values(r3) <- runif(nrows * ncols)

# create raster stack (same spatial extent and resolution)
rS <- stack(list(r1, r2, r3))

# create raster brick
rB <- brick(list(r1, r2, r3))
```

Raster Stacks vs. Bricks

Raster stack: combine raster layers/multiple files

- High-resolution satellite imagery by color/wavelength bands



Raster brick: combine raster layers/same file

- 4D data: rainfall (z) by location (x,y) every day (time)

Data I/O

```
# initialize data path
data.path <- file.path(getwd(), "data/")

# confirm feasible file formats for spatial I/O
OgrDrivers()

# get shapefile metadata
ogrInfo(data.path, “layer”)

# read shapefiles...same for other formats
shapes = readOGR(data.path, "layer")
sensor.points <- readOGR(data.path, “sensors.klm”)

write shapefiles...same for other formats
writeOGR(shapes, data.path, "layer", "ESRI Shapefile")
writeOGR(shapes, data.path, "towns.kml", "KML")

# read raster files
r = raster("data.tif")

# write raster file
writeFormats()
writeRaster(r1, paste0(data.path, "data2.tif"), "GTiff")
```

Coordinate Reference Systems

proj4strings

```
NAD83 <- CRS("+proj=longlat +datum=NAD83 +no_defs +ellps=GRS80 +towgs84=0,0,0" )
NATATLAS <- CRS("+proj=laea +lat_0=45 +lon_0=-100 +x_0=0 +y_0=0 +a=6370997 +b=6370997 +units=m +no_defs" )
WGS84 <- CRS("+proj=longlat +ellps=WGS84 +datum=WGS84 +no_defs")
ALBERS <- CRS("+proj=aea +lat_1=29.5 +lat_2=45.5 +lat_0=37.5 +lon_0=-96 +x_0=0 +y_0=0 +ellps=GRS80 +datum=NAD83 +units=m +no_defs" )
```

Reference sources for proj4 strings:

- www.spatialreference.org
- www.epsg.io

Which projection is best?

- experiment!
- US Dept of Interior reference paper Map Projections: A Working Manual, 1987
- www.epsg.io

Manipulate CRS Map Projections

```
# interogate shapefile CRS
proj4string(SLDF)

# assign shapefile CRS
proj4string(SLDF) <- CRS(WGS84)
SL <- SpatialLines(list(Ls1), proj4string = CRS(ALBERS))

# change or transform shapefile CRS
sl.NAD83 <- spTransform(SL, CRS(NAD83))

# interogate raster CRS
projection(r1)

# assign raster CRS
projection(r1) <- CRS(NAD83)
r1 <- raster(list(x = x, y = y, z = z), crs = NATATLAS)

# change or transform raster CRS
r1.WGS84 <- projectRaster(r1, crs = WGS84)
```

NOTE: all CRS strings used (e.g. WGS84, ALBERS, etc.) were defined on the prior page and recycled here.

Map Visualizations

Use syntax for base R graphics!

- Plot methods for spatial objects rely on base graphics
- Maintain detailed control of all graph or device elements
- “Old school” methods are still the best for customization of spacial details
- Syntax may be awkward, but the results justify the effort

ggplot2 is fine … *but flawed at the core*

- ggplot2 was designed to work with data.frames only...NOT **`sp`** objects
- Complex **`sp`** object structures must be destroyed / converted to data.frames
- Why do that when base R and **`sp`** plot methods are robust?
- Control of data graph elements is excellent, but no graphical device controls
- Custom mapping capability is very good, but features are missing

Java based tools are best for website design

- See **`mapview`** package for quick web delivery
- See R interface to **`Leaflet`** and **`OpenLayers`** for professonal web delivery
- Explore the R interface for the java heavyweight **`OpenStreetMaps`**
- Save **`sp`** objects in GoogleEarth KLM format (web aligned, not open-source!)

Case Study

A solar project developer has completed “ground-truthing” or data collection for development sites in Qatar. 6 of 20 locations are deemed suitable for future development.

Each site has had remote solar sensors in place for 12 months. The short-term solar resource data spans one seasonal year, but the long-term solar energy potential is still not clear.

Fortunately, solar data collection by satellites has been in place for more than 20 years and covers the entire globe. Correlate the short-term site-specific observations with the appropriate long-term satellite observations, and use the model estimates to “backcast"the long-term solar resource as each site.

Your Mission - complete a GIS land surface analysis in one day to prepare for model development:

- Identify and plot all potential development sites in Qatar
- Download satellite reference or pixel points across the globe and extract only those points that sit on the Qatar land mass. Overlay and plot the satellite reference points on the site map of Qatar
- Create buffer zones of 5KM and 10km around each potential development site and extract the satellite pixal points in the buffer zones that best represent each project site
- Identify the metadata ID for each extracted pixel points to confirm which long-term satellite data to download from the satellite FTP site
- Trigger the FTP download before going home so the satellite data is ready for modelling in the morning

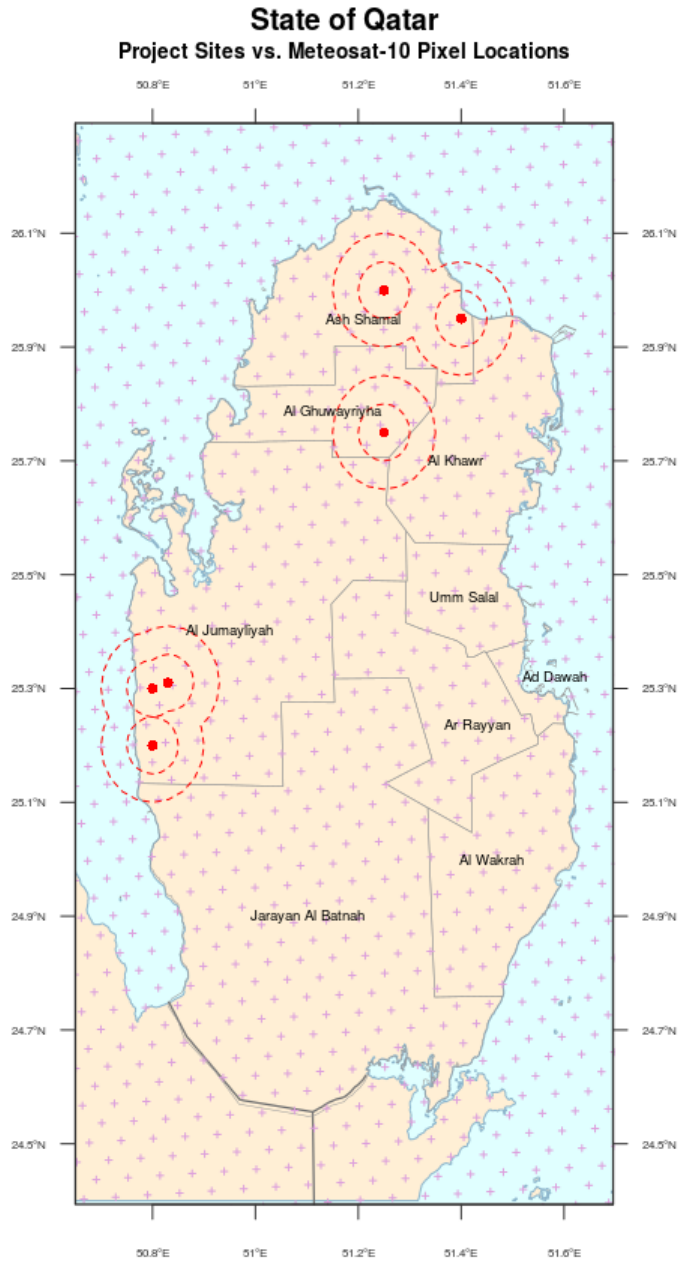
Case Study Answer (1 of 4)

```
# 0.Initialize Code
# Load librariers and DLLs
library(sp)
library(rgdal)
library(maptools

# paths – update and point to input map vector files
data.path <- "/home/bxhorn/Documents/DohaR/data/"

# map projection and graph devices
map.proj <- CRS("+proj=longlat +ellps=WGS84")
op <<- par()
##-------------------------------------------------------------------------------------------##

# 1.Define spatial points for 6 potential development sites
x <- c(51.25, 51.40, 51.25, 51.05, 51.07, 51.05)
y <- c(26.00, 25.95, 25.75, 25.30, 25.31, 25.20)
coords <- cbind(x, y)
site.names <- data.frame(site = LETTERS[1:6])

# go spatial
spdf.sites <- SpatialPointsDataFrame(coords, site.names)
##-------------------------------------------------------------------------------------------##

# 2.Load Qatar map data
qatar.bbox <- matrix(c(50.7, 51.65, 24.4, 26.4), ncol = 2, byrow = TRUE,
                     dimnames = list(c("x", "y"), c("min", "max")))

# GSHHS Shores Region Map (SpatialPolygons)
qatar.shores <- Rgshhs(paste0(data.path, "gshhs_f.b"), xlim = c(50.65, 51.7), ylim = c(24.4, 26.4))
q.shoreGSHHS <- qatar.shores$SP

# NOAA Borders (SpatialLinesDataFrame)
WDBIIinfo <- ogrInfo(data.path, "WDBII_border_f_L1")
WDBII_b_L1 <- readOGR(data.path, "WDBII_border_f_L1")
q.borderNOAA <- pruneMap(WDBII_b_L1, xlim = c(50.65, 51.7), ylim = c(24.4, 26.4))

# GADM Amin Boundries
GADMinfo <- ogrInfo(data.path, "QAT_adm1")
q.adminGADM <- readOGR(data.path, "QAT_adm1")

# Convert from SpatialPolygonDataFrame to SpatialLineDataFrame
q.adminGADM <- as(q.adminGADM, "SpatialLinesDataFrame")
##-------------------------------------------------------------------------------------------##

# 3.Load LSA-SAF pixels point and geographical coordinates
lat.mena <- readGDAL(paste0(data.path, "HDF5_LSASAF_MSG_LAT_NAfr_4bytesPrecision"))
lon.mena <- readGDAL(paste0(data.path, "HDF5_LSASAF_MSG_LON_NAfr_4bytesPrecision"))

# clean-up missing data
is.na(lat.mena@data$band1) <- lat.mena@data$band1 == 900000
is.na(lon.mena@data$band1) <- lon.mena@data$band1 == 900000

# define spatial points for satellite pixels over Qatar land mass
lon.sp <- lon.mena@data$band1/10000
lat.sp <- lat.mena@data$band1/10000
mat.sp <- cbind(lon.sp, lat.sp)
row.names(mat.sp) <- 1:nrow(mat.sp)
sp.pixels <- SpatialPoints(na.omit(mat.sp), bbox = qatar.bbox, proj4string = map.proj)
##-------------------------------------------------------------------------------------------##

4. Create 10km and 5km Buffer Zones (width in decimal degrees)
buff10k <- gBuffer(spdf.sites, width = 0.10)
buff5k <- gBuffer(spdf.sites, width = 0.05)
```

Case Study Answer (2 of 4)

```
# 4.Create State of Qatar Map with Satellite Pixels, Proposed Project Sites and Buffer Zones
M0title <- "State of Qatar"
M1title <- "Project Sites vs. Meteosat-10 Pixel Locations"
S0title <- "Pixel Map: MSG Data Servers, Satellite Application Facility, EUMETSAT"
S1title <- "Land Map: GSHHS and WDBII Databases, National Geophysical Data Center, NOAA"

par(op)
par(mar = c(2,2,2,2) + 0.01, pin = c(3.73, 7.5))

plot(q.shoreGSHHS, xaxs = "i", yaxs = "i", axes = FALSE, bg = "lightcyan", col = "papayawhip", border = "lightskyblue3")
plot(q.borderNOAA, lwd = 1.3, col = "grey45", add = TRUE)
plot(q.adminGADM, lwd = 0.5, col = "grey60", add = TRUE)
plot(sp.pixels, col = "plum", cex = 0.45, add = TRUE)
plot(spdf.sites, col = "red", pch = 16, cex = 0.95, add = TRUE)
plot(buff5k, lty = 2, border = "red", Add = TRUE)
plot(buff10k, lty = 2, border = "red", Add = TRUE)
axis(1, at = c(50.8 + 0:5 * 0.2), pos = 24.393, las = 1, cex.axis = 0.5, col = "grey20",
     col.axis = "grey20", labels = parse(text = sp:::degreeLabelsEW(c(50.8 + 0:5 * 0.2))))
axis(2, at = c(24.5 + 0:10 * 0.2), pos = 50.65, las = 2, cex.axis = 0.5, col = "grey20",
     col.axis = "grey20", labels = parse(text = sp:::degreeLabelsNS(c(24.5 + 0:10 * 0.2))))
axis(3, at = c(50.8 + 0:5 * 0.2), pos = 26.2927, las = 1, cex.axis = 0.5, col = "grey20",
     col.axis = "grey20", labels = parse(text = sp:::degreeLabelsEW(c(50.8 + 0:5 * 0.2))))
axis(4, at = c(24.5 + 0:10 * 0.2), pos = 51.695, las = 2, cex.axis = 0.5, col = "grey20",
     col.axis = "grey20", labels = parse(text = sp:::degreeLabelsNS(c(24.5 + 0:10 * 0.2))))
title(main = M0title, font.main = 2, line = 3.25)
title(main = M1title, cex.main = 0.85, font.main = 2, line = 2.25)
title(sub = S0title, cex.sub = 0.80, font.sub = 3, line = 2.0)
title(sub = S1title, cex.sub = 0.80, font.sub = 3, line = 3.00)
box(col = "grey20", lwd = 1.3)
text(x = 51.1, y = 24.9, label = "Jarayan Al Batnah", cex = .6)
text(x = 51.46, y = 25.0, label = "Al Wakrah", cex = .6)
text(x = 50.95, y = 25.4, label = "Al Jumayliyah", cex = .6)
text(x = 51.15, y = 25.785, label = "Al Ghuwayriyha", cex = .6)
text(x = 51.431, y = 25.235, label = "Ar Rayyan", cex = .6)
text(x = 51.58, y = 25.32, label = "Ad Dawah", cex = .6)
text(x = 51.405, y = 25.46, label = "Umm Salal", cex = .6)
text(x = 51.39, y = 25.7, label = "Al Khawr", cex = .6)
text(x = 51.21, y = 25.95, label = "Ash Shamal", cex = .6)
##------------------------------------------------------------------------------##
```

Case Study Answer (3 of 4)



Pixel Map: MSG Data Servers, Satellite Application Facility, EUMETSAT
Land Map: GSHHS and WDBII Databases, National Geophysical Data Center, NOAA

Case Study Answer (4 of 4)

```
# 5. Extract Satellite Pixels Inside the 10km and 5km Buffer Zones
# first ensure the required data objects have the same CRS
proj4string(sp.pixels) <- WGS84
proj4string(buff10k) <- WGS84
proj4string(buff5k) <- WGS84

# Next two lines confirm the beauty of sp methods
# A few dozen satellite pixel points are easily extracted from hundreds of millions!!!
ftplist.10km <- sp.pixels[buff10k]
ftplist.5k <- sp.pixels[buff5k]
##-------------------------------------------------------------------------------##

# 6. Confirm pixel ID# and coordinates for FTP download
ftplist.5k@coords

       lon.sp lat.sp
674139  51.27  26.03
676350  51.24  26.00
678562  51.28  25.97
678564  51.41  25.97
680775  51.38  25.94
680776  51.44  25.94
682987  51.42  25.91
691830  51.24  25.76
694041  51.22  25.73
694042  51.28  25.73
718364  51.05  25.35
720575  51.03  25.32
720576  51.09  25.32
722787  51.07  25.29
727209  51.01  25.22
727210  51.08  25.22
729421  51.05  25.19
```

APPENDIX

Historical Perspective

- **600 BC:** Babylonian world map...flat earth paradigm
- **200 BC:** Hellenistic world map...spherical earth paradigm
- **1154:** Al-Idrisi's Tabula Rogeriana (Arabic:نزهة المشتاق في اختراق الآفاق) "The book of journeys" with multiple continents
- **1569:** Mercator world map...first cylindrical map projection
- **1832:** Charles Picquet's map of cholera epidemiology in Paris with 48 districts and half-tone colour gradients...first known GIS map with a spatial polygons data.frame
- **1940-1960:** The US Army and the US Air Force nuclear command introduce new geographic datum and coordinate reference systems
- **1960-1975:** Early GIS pioneers: US Census, US Geological Survey and Harvard Laboratory for Computer Graphics and Experimental Cartography
- **1967:** Publication of An introduction to the Geo-Information System of the Canada Land Inventory by R. F. Tomlinson
- **1969:** Environmental Systems Research Institute (ESRI) is formed as a land use consulting firm
- **1975-1985:** International governments promote geodata experimentation yielding modest success and stunning failures
- **1976:** S-Plus is born based on the goal "to change the way data is managed and visualized," Bell Labs, AT&T
- **1978:** First GPS satellite is launched
- **1982:** US Army Corps of Engineers begins open-source Linux development of GRASS (Geographic Resource Analysis Support System)
  ESRI releases Arc/Info desktop application with license for Linux
- **1984:** S-Plus released to the public by AT&T with no copyright protection
- **1987:** US Dept of Interior publishes the reference paper Map Projections: A Working Manual
- **1988:** Commercial version of S-Plus is released with a proprietary license issued by StatSci
- **1991:** Seminal textbook Geographic Information Systems by Paul Longly et. al
- **1993:** **`Maps`** package for S-Plus by Becker and Wilks;
  Open-source R programming language (aka GNU SPlus) is released
- **1994:** Open Geospatial Consortium (OGC) formed to promote spatial data standards
  GRASS application migrates from the US government to universities for source code development and management
- **1997:** Arc/Info 6.0 released with a major subsystem for raster (GRID) processing and satellite imagery
- **1998:** Introduction of vector shapefiles and other map file formats that combine spatial and demographic data
- **1999:** Release of industry leading ArcGIS application, replacing Arc/Info
- **2004:** OGC promotes simple GIS data object and feature specifications that are widely embraced
- **2005:** **`sp`** and **`rgdal`** packages for geo-processing in R by Roger Bivand, et.al
  ArcGIS 9.0 introduced with basic geo-processing capabilities