GIS

GIS

1. Why GIS in economics?
2. Coordinate systems
3. Spatial data types: vector (points, lines, polygons), raster.
4. Merging spatial data
   (spatial + non-spatial, points and raster to polygons).
5. Drawing maps.
6. Geometric manipulations (buffers, distances, intersections, etc.)

1. Why GIS in economics?

- Expand the set of researchable questions
  - Use satellite data
  - Merge different data spatially
- Improve identification
  - More covariates to mitigate omitted variable bias
  - Instruments
  - RD-design
  - Estimate the spillover effect on the control group in RCTs .

Examples

- Measurement
  - Deforestation (Burgess et al, 2012)
  - Night lightning
- Instrumentation
  - Propaganda and Conflict: Evidence from the Rwandan Genocide (Yanagizawa-Drott, QJE 2014)
- RD
  - The Historical State, Local Collective Action, and Economic Development in Vietnam (Nathan Lane et al. Econometrica, 2018).
  - The Persistent Effects of Peru's Mining Mita (Dell, EMA 2010)

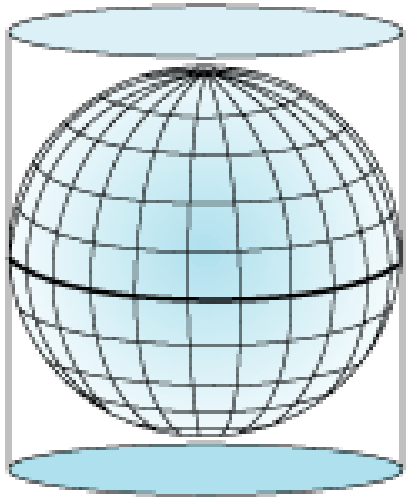
2. Coordinate referencing systems

Earth is a sphere, but we need to represent its surface on a plane.

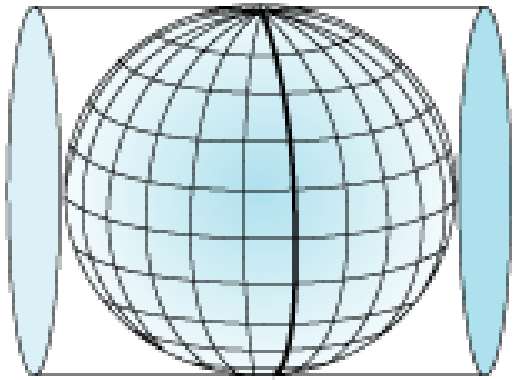
1. Model approximation of the globe (ellipsoid)
   - WGS 1984 is most popular system.
2. Projection: coordinate systems in 2D
   - Latlon
   - UTM

Cylindrical projection

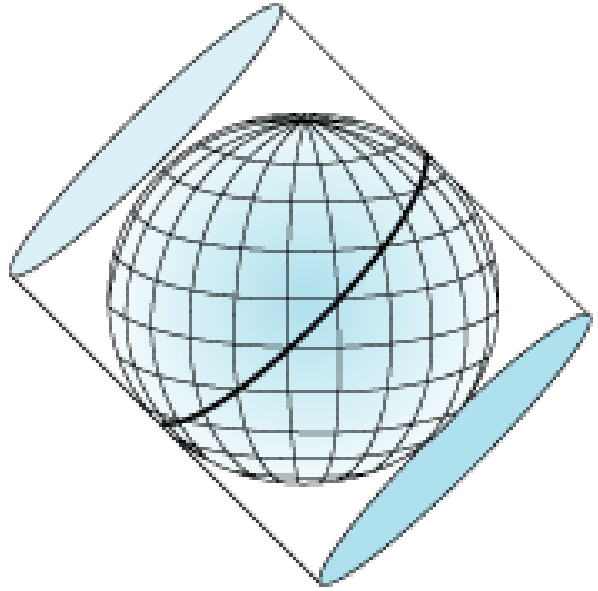
Cylindrical Aspects



Normal



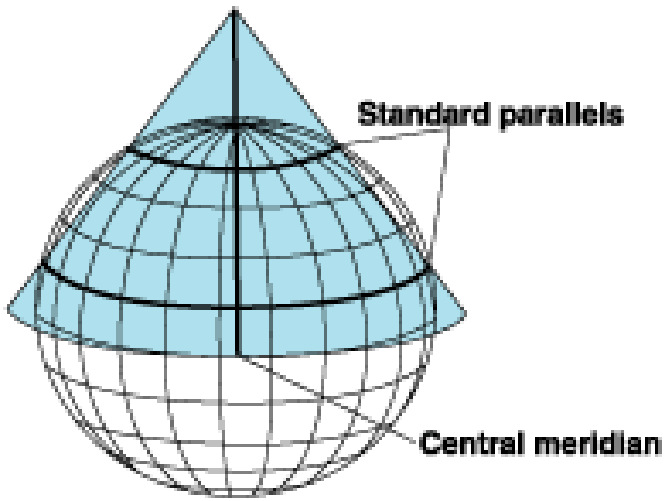
Transverse

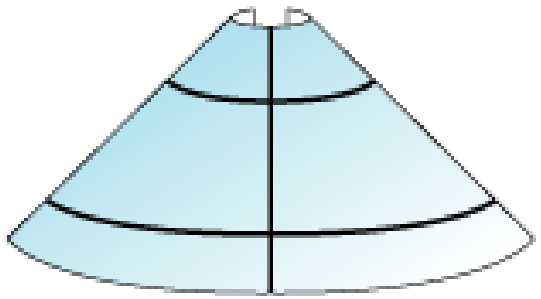


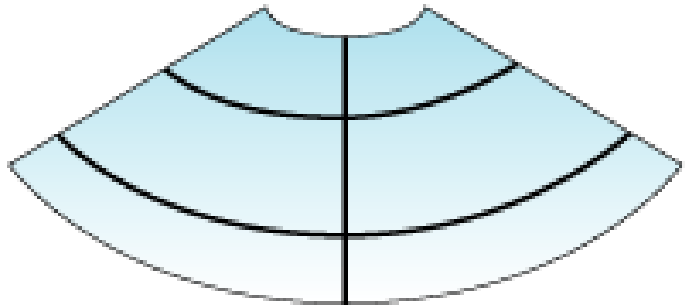
Oblique

Conic projection

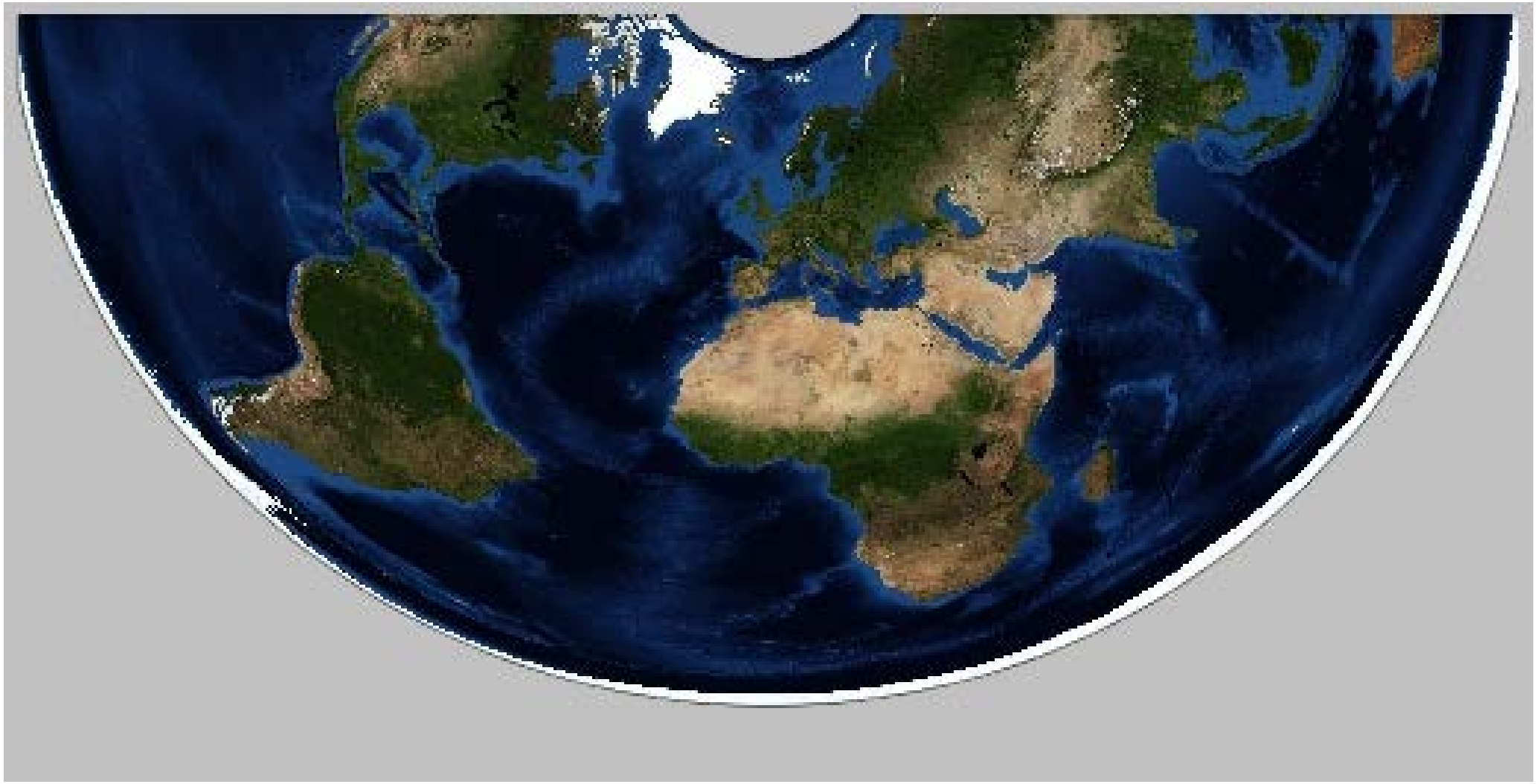
Conic (secant)





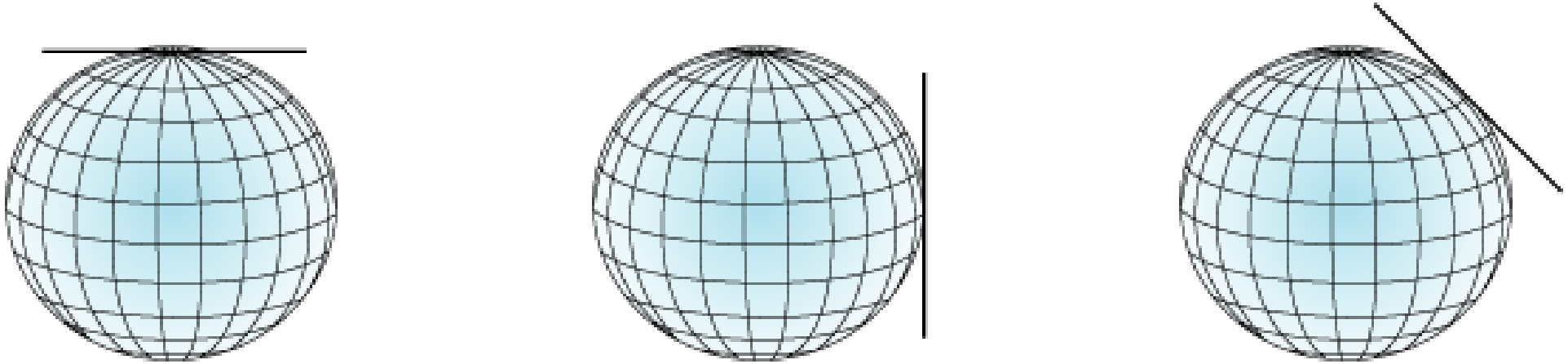


Alberts Equal Area Conic projection



Planar projection

Planar Aspects

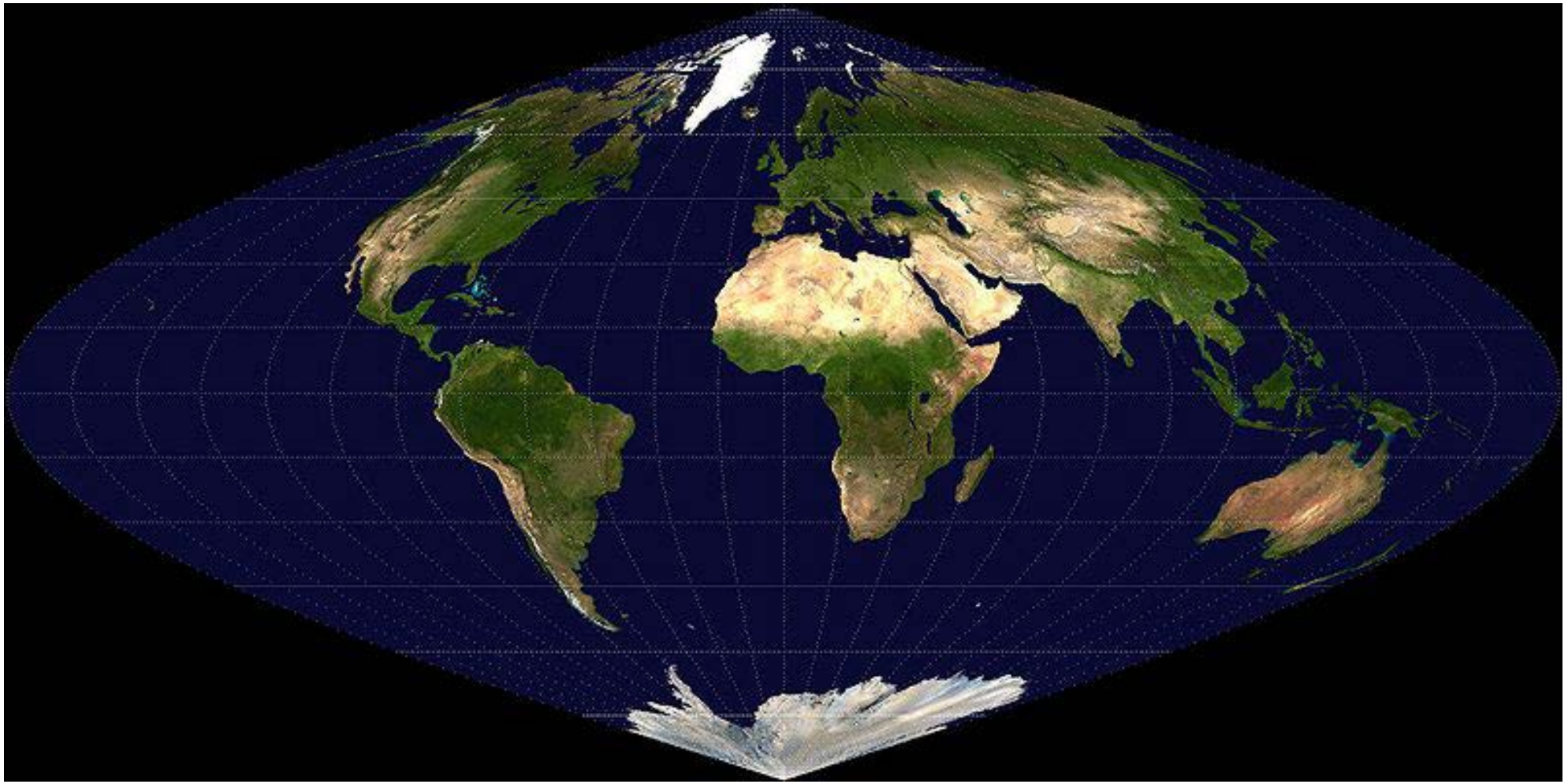


Polar

Equatorial

Oblique

Sinusoidal projection



GIS Software

- **R**
  - Spatial Data Science with R and ”terra”
  - GIS with Terra
  - tidyterra
- ArcGIS
  - Comprehensive, but expensive and buggy.
- Python
  - ArcGIS, QGIS, geopandas

Coordinate Reference Systems (CRS) in R

- CRS("+proj=utm +zone=33 +ellps=WGS84 +datum=WGS84 +units=m+ towgs84=0,0,0+no_defs")

- Earth Model
  - **The Ellipse:** Describes the generalized shape of the Earth. The ellipsoid is an approximation and does not fit the Earth perfectly. The Earth is almost spherical, however there is a tiny bulge at the equator that makes it ~0.33% larger than at the poles. There are different ellipsoids in use, some are designed to fit the whole Earth (WGS84, GRS80) and some are designed to fit a local region (NAD27).

  - **The Datum**: Defines origin and orientation of the coordinate axes (as well the ellipsoid. Datums are based on specific ellipsoids and sometimes have the same name as the ellipsoid.)
    - towgs84: conversion to a global datum. WGS84 itself has "+towgs84=0,0,0".
      - proj=+proj=latlong +ellps=WGS84 +towgs84=0,0,0 is equivalent to
      - +latlong +datum=WGS84

Coordinate Reference Systems (CRS) in R

- CRS("+proj=utm +zone=33 +ellps=WGS84 +datum=WGS84 +units=m +no_defs")
- **Projection**: Project the globe onto a 2D surface.
  - **Universal Transverse Mercator (UTM)**
    - The UTM projection is commonly used in research because it tends to be more locally accurate.
    - The mercator projection preserves angles and direction, but distorts distance. To minimize this distortion, the UTM divides the Earth into sixty zones.
  - **longlat**

UTM zone numbers

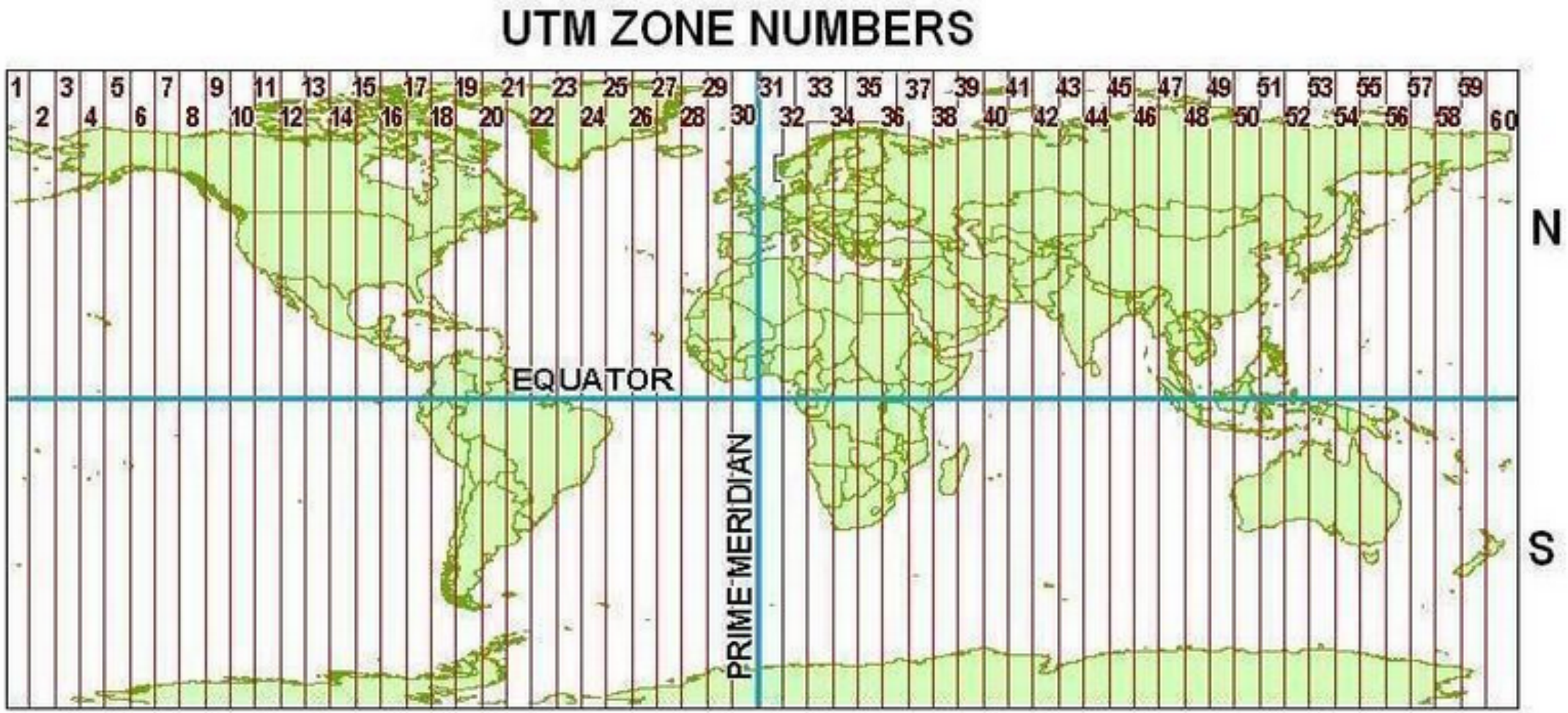


Figure 1. 60 UTM Zones divided North and South. Top↑

EPSG codes

- A particular CRS can be referenced by its EPSG code.

```
> CRS("+init=EPSG:32633")
CRS arguments:
 +init=EPSG:32633 +proj=utm +zone=33 +datum=WGS84 +units=m +no_defs +ellps=WGS84 +towgs84=0,0,0
```

3. Spatial data types

- Vector data
  - Types (feature classes)
    - Point
    - Line
    - Polygon
  - Format
    - Shapefiles (.shp)
  - Examples
    - Countries, sub-national districts, roads, cities.
- Raster data
  - Divides the earth surface into many “square” cells (or pixels)
  - Each cell contains one value
  - Often created from satellite images
  - Elevation, forest coverage, suitabilty for agriculture, night light, ndvi, pollution.

GIS in R

- Packages
  - sp, sf, terra, (tidyterra).

S4 objects

- sp package uses S4 objects to represent spatial data.
- Information stored in slots, accessed with @
- Foundational class is Spatial, with 10 subclasses

```
# Load the sp package
library(sp)

# Spatial classes
getClass("Spatial")
```

```
## Class "Spatial" [package "sp"]
##
## Slots:
##
## Name:         bbox proj4string
## Class:      matrix         CRS
##
## Known Subclasses:
## Class "SpatialPoints", directly
## Class "SpatialMultiPoints", directly
## Class "SpatialGrid", directly
## Class "SpatialLines", directly
## Class "SpatialPolygons", directly
## Class "SpatialPointsDataFrame", by class "SpatialPoints", distance 2
## Class "SpatialPixels", by class "SpatialPoints", distance 2
## Class "SpatialMultiPointsDataFrame", by class "SpatialMultiPoints", distance 2
## Class "SpatialGridDataFrame", by class "SpatialGrid", distance 2
## Class "SpatialLinesDataFrame", by class "SpatialLines", distance 2
## Class "SpatialPixelsDataFrame", by class "SpatialPoints", distance 3
## Class "SpatialPolygonsDataFrame", by class "SpatialPolygons", distance 2
```

Load point data

- Read table into R
- Convert the table into a spatial object using the coordinates command and passing the names of the columns in the table that correspond to longitude and latitude:

```
sf.df=import('sf_restaurant_inspections.csv')
View(sf.df)
```

	V1	business_id	Score	name	longitude	latitude
1	1	19	94	Nrgize Lifestyle Cafe	-122.4215	37.78685
2	2	24	96	OMNI S.F. Hotel - 2nd Floor Pantry	-122.4031	37.79289
3	3	31	100	Norman's Ice Cream and Freezes	-122.4190	37.80716
4	4	45	94	CHARLIE'S DELI CAFE	-122.4136	37.74711

```
> class(sf.df)
[1] "data.frame"
> coordinates(sf.df) <- c("longitude", "latitude")
> class(sf.df)
[1] "SpatialPointsDataFrame"
```

Slots in SpatialPointsDataFrame

```
# slots for SpatialPointsDataFrame
slotNames("SpatialPointsDataFrame")
```

```
## [1] "data"        "coords.nrs"  "coords"      "bbox"        "proj4string"
```

Global Environment

Data

```
sf.df                    Formal class SpatialPointsDataFrame
   ..@ data :'data.frame': 2697 obs. of 4 variables:
   .. ..$ V1 : int [1:2697] 1 2 3 4 5 6 7 8 9 10 ...
   .. ..$ business_id: int [1:2697] 19 24 31 45 48 54 56 61 66 67 ...
   .. ..$ Score : int [1:2697] 94 96 100 94 92 100 98 92 100 90 ...
   .. ..$ name : chr [1:2697] "Nrgize Lifestyle Cafe" "OMNI S.F. Hotel - 2nd Floor Pantr…
   ..@ coords.nrs : int [1:2] 5 6
   ..@ coords : num [1:2697, 1:2] -122 -122 -122 -122 -122 ...
   .. ..- attr(*, "dimnames")=List of 2
   .. .. ..$ : chr [1:2697] "1" "2" "3" "4" ...
   .. .. ..$ : chr [1:2] "longitude" "latitude"
   ..@ bbox : num [1:2, 1:2] -122.5 37.7 -122.3 37.8
   .. ..- attr(*, "dimnames")=List of 2
   .. .. ..$ : chr [1:2] "longitude" "latitude"
   .. .. ..$ : chr [1:2] "min" "max"
   ..@ proj4string:Formal class 'CRS' [package "sp"] with 1 slot
   .. .. ..@ projargs: chr NA
```

```
sf.df.coords<-sf.df@coords
```

Filter

	longitude	latitude
1	-122.4215	37.78685
2	-122.4031	37.79289
3	-122.4190	37.80716
4	-122.4136	37.74711
5	-122.4657	37.76401
6	-122.4377	37.78463

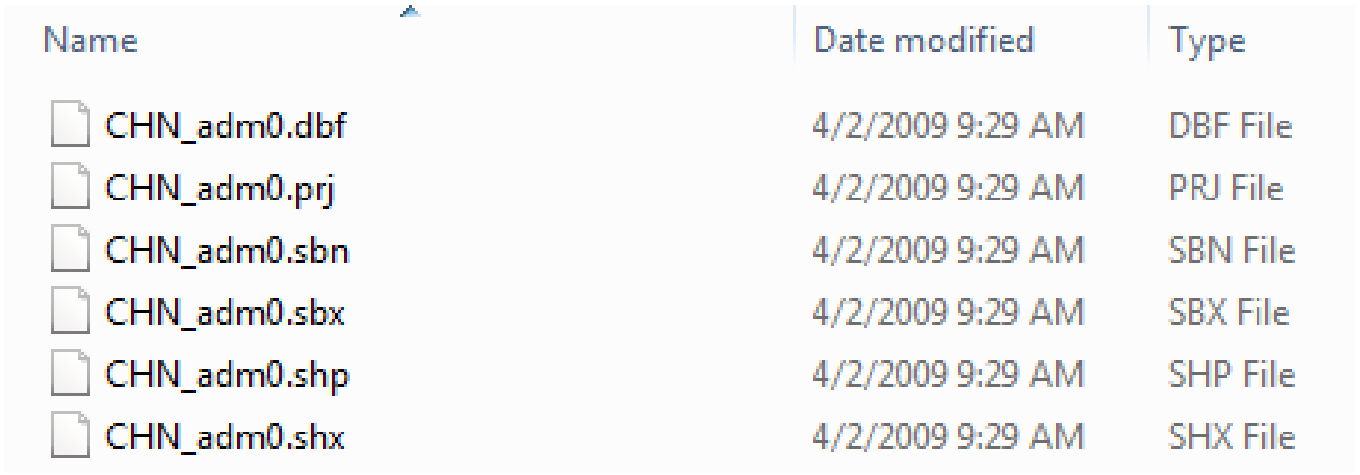
```
sf.df.bbox<-sf.df@bbox
```

Filter

	min	max
longitude	-122.51005	-122.26417
latitude	37.66882	37.83463

Polygon data

- .shp main shapefile info
- .prj coordinate system and projection
- .dbf attributes



Name	Date modified	Type
CHN_adm0.dbf	4/2/2009 9:29 AM	DBF File
CHN_adm0.prj	4/2/2009 9:29 AM	PRJ File
CHN_adm0.sbn	4/2/2009 9:29 AM	SBN File
CHN_adm0.sbx	4/2/2009 9:29 AM	SBX File
CHN_adm0.shp	4/2/2009 9:29 AM	SHP File
CHN_adm0.shx	4/2/2009 9:29 AM	SHX File

CHN_adm0.prj

```
1  GEOGCS["GCS_WGS_1984",DATUM["D_WGS_1984",SPHEROID
   ["WGS_1984",6378137.0,298.257223563]],PRIMEM["Gre
   enwich",0.0],UNIT["Degree",0.0174532925199433]]
```

Load polygon data

- # We can read in and write out spatial data using: terra::vect(filename.shp)

```
> CHN_adm2<-vect("../GISdata/CHN_adm/CHN_adm2.shp")
> CHN_adm2
 class       : SpatVector
 geometry    : polygons
 dimensions  : 345, 18  (geometries, attributes)
 extent      : 73.5577, 134.7739, 15.78, 53.56086  (
 source      : CHN_adm2.shp
 coord. ref. : lon/lat WGS 84 (EPSG:4326)
 names       :  ID_0   ISO NAME_0  ID_1      NAME_1
 type        : <int> <chr>  <chr> <int>       <chr> <
 values      :    45   CHN  China   589 Nei Mongol
                  45   CHN  China   589 Nei Mongol
                  45   CHN  China   589 Nei Mongol
```

The associated data is read from the .dbf file

```
> cn1<-as.data.frame(CHN_adm2)
```

rgdal.R* | cn1 | cn0 | spatial.r* | sf.df

Filter

	ID_0	ISO	NAME_0	ID_1	NAME_1	Shape_Leng	Shape_Area
0	45	CHN	China	571	Anhui	30.519431	1.336874e+01
1	45	CHN	China	572	Beijing	8.644659	1.734891e+00
2	45	CHN	China	573	Chongqing	26.880007	7.713129e+00
3	45	CHN	China	574	Fujian	76.743236	1.103551e+01
4	45	CHN	China	575	Gansu	78.238365	4.150390e+01

Raster data

A raster dataset has three primary components:

1. A grid, which consists of:
   - dimensions (number of rows and columns),
   - resolution (size of sides of each cell),
   - and extent (where the edges of the grid “are”)
2. A set of values associated with each cell in the grid
3. Projection data about how the grid relates to the physical world

Load raster data

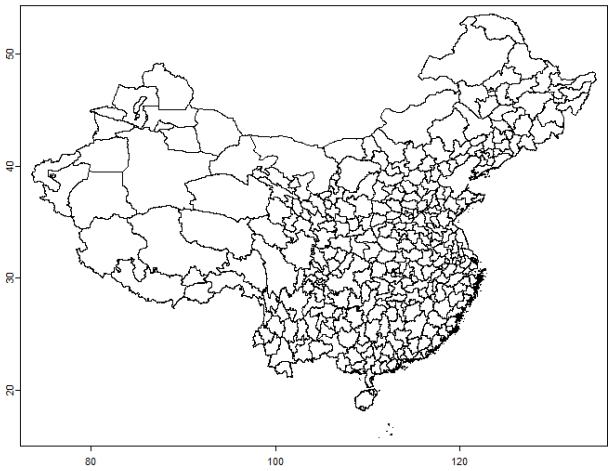
```
> pollution <- rast("R_Workshop/RGIS2_Data/pollution.tif")
> crs(pollution)
[1] "GEOGCRS[\"WGS 84\",\n    ENSEMBLE[\"World Geodetic System 1984
```

4. Merging spatial data

- Spatial + nonspatial (data)
- Spatial + spatial (and computing averages/statistics)
  - Point -> polygon
  - Raster -> polygon

Spatial joins (merge): Spatial* + Non-Spatial

```
# Terra command vect
CHN_adm2<-vect("../GISdata/CHN_adm/CHN_adm2.shp")
CHN_adm2
plot(CHN_adm2)
```



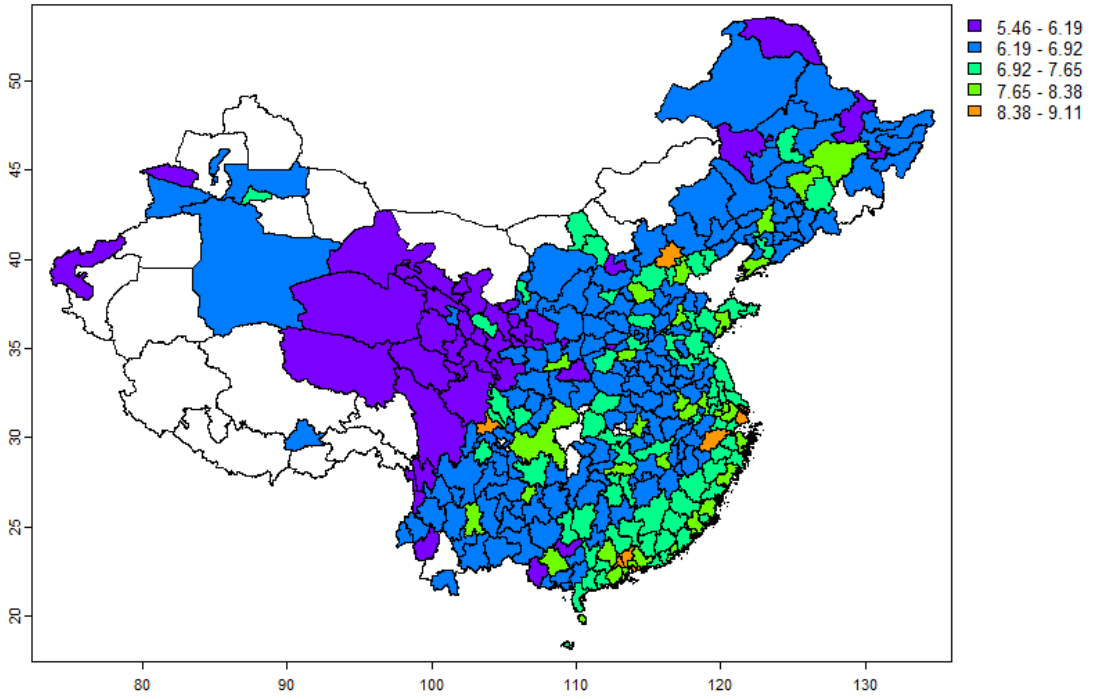
+

```
weibo<-import("../totalposts_p.dta")
```

count	provname	prefname	ID_2
1.192e+09	Shanghai	Shanghai	10025
1414652	Yunnan	Lincang	10090
5146081	Yunnan	Lijiang	10089
2146879	Yunnan	Baoshan	10082
6235636	Yunnan	Dali Bai	10084

=

```
CHN_adm2 <- merge(CHN_adm2, weibo, by.x = "ID_2", by.y = "ID_2")
plot(CHN_adm2, "lncount")
```



Merged spatial objects must have the same projection

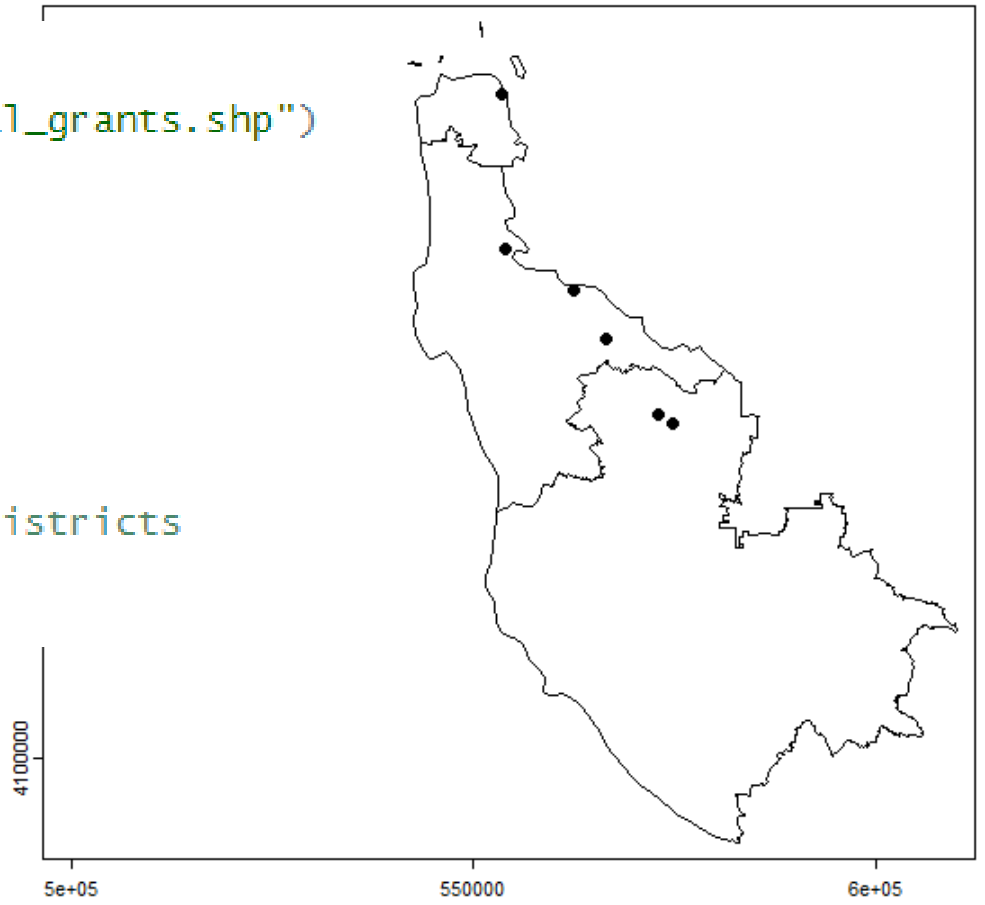
- Re-projecting vector data requires two tools from the sp and rgdal packages:
  - a Coordinate Reference System CRS object with the new CRS you wish to apply
  - the spTransform() method
- A CRS object includes all the information needed to project a spatial object,
  - a Geographic Coordinate System (the model of the Earth used to create the data) and
  - a projection (a way of converting points on the three-dimensional Earth onto a two-dimensional plane).
- Through package terra, the crs() function has access to a large library of coordinate systems and transformations,
  - you just need to get the code for the CRS you want.
  - Codes – often called a “projection strings” – can be found at http://www.spatialreference.org/.

Spatial joins (merge): Spatial* + Spatial*

- sp
  - spTransform re-projects objects.
  - We have an object called MyCity and we want to reproject this into a new CRS:
    - MyNewCRS <- CRS("+init=EPSG:32633")
    - MyCity.reprojected <- spTransform(MyCity, MyNewCRS)
- terra
  - MyNewCRS <- crs("+init=EPSG:32633")
  - MyCity.reprojected <- terra::project(MyCity, MyNewCRS)

Example: re-projecting point to fit polygon

```r
# Federal project grants (Spatial points)
grants<-vect("R_Workshop/RGIS2_Data/shapefiles/federal_grants.shp")

# Check projection of grants and districts.
crs(grants)
crs(districts)
# Put grant on same projection as districts.
new.crs <- crs(districts)
grants.newproj <- terra::project(grants, new.crs)

# Check that the reprojected grants fall inside the districts
plot(districts)
plot(grants.newproj, add=TRUE)
```



What points lies in what polygon: extract {terra}

extract: returns a data frame that for each row in the second argument (grants, id.y) merges in information from the first (districts)

```
grants.districts<-terra::extract(districts,grants.newproj)
```

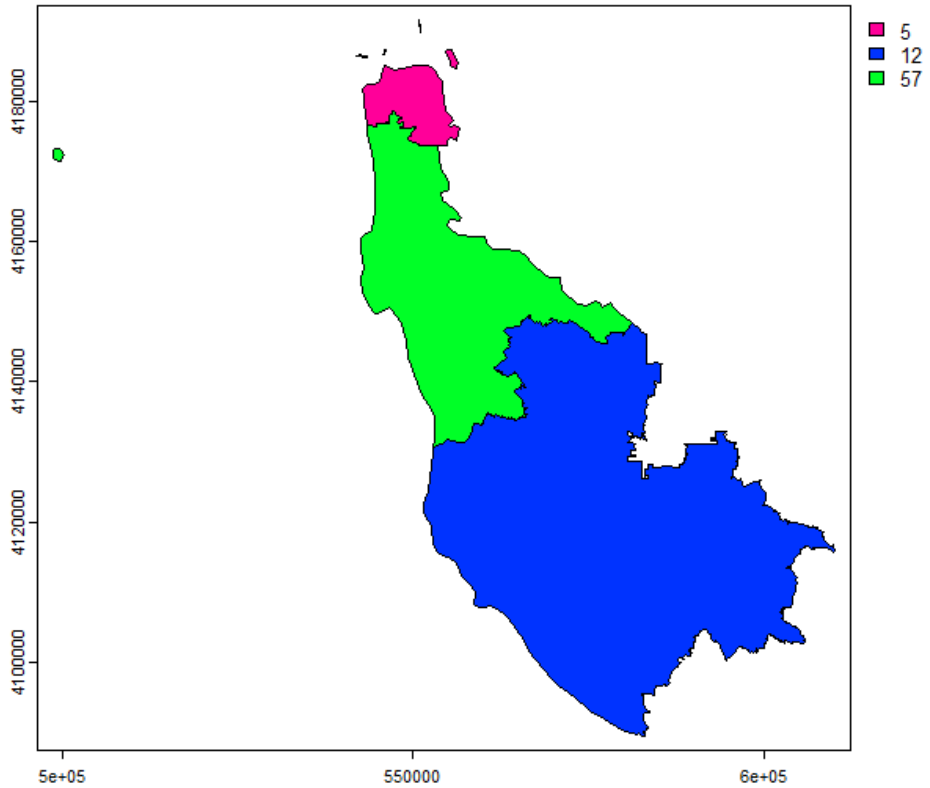
Class.r* × grants.districts ×

Filter

	id.y	DISTRICT	NAME	Shape_Leng	Shape_Area	dem_vote_share
1	1	12	San Francisco	70727.76	105358679	88.3
2	2	14	Peninsula	200487.79	706830475	76.7
3	3	14	Peninsula	200487.79	706830475	76.7
4	4	14	Peninsula	200487.79	706830475	76.7
5	5	18	Palo Alto	308447.51	1815914163	67.8
6	6	18	Palo Alto	308447.51	1815914163	67.8

Total jobs per district

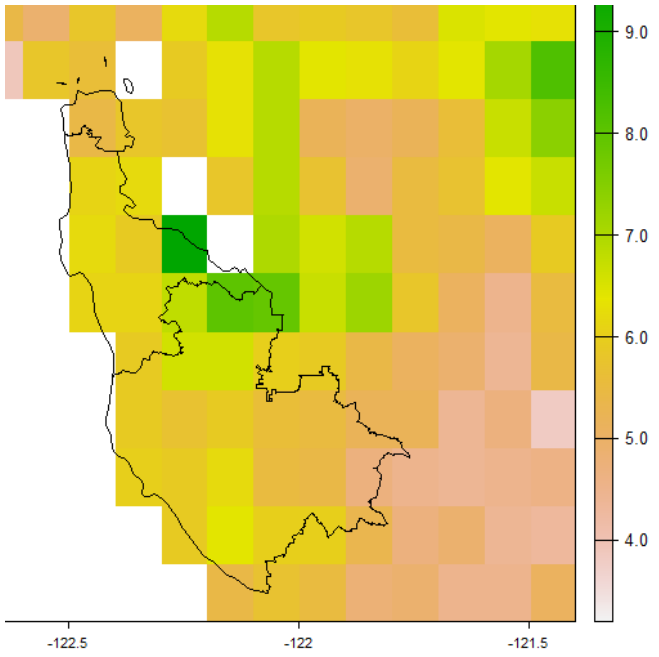
```
# Merge in district data to grants map.
grants.newproj2 <- merge(grants.newproj, grants.districts, by="row.names")

# Merge in total jobs created per district
d<- as.data.frame(grants.newproj2) %>%
  group_by(DISTRICT)  %>%
  summarise(JobsCreate=sum(JobsCreate))

districts <- merge(districts, d, by="DISTRICT")
plot(districts,"JobsCreate")
```



Rasters + SpatialPolygons

- Common task: compute average of raster data (light, pollution, NDVI) within geographic areas.
- Reproject polygon, not raster!
  - Rectangular raster grid is CRS dependent. Reprojected grid points will average values from multiple original grid points: time consuming.

```
# Read in raster data on pollution.

pollution <- rast("R_Workshop/RGIS2_Data/pollution.tif")
crs(pollution)
crs(districts)

# Put grant on same projection as districts.
new.crs <- crs(pollution)
districts.newproj <- terra::project(districts, new.crs)

plot(pollution)
plot(districts.newproj, add=TRUE)
```



What raster point lies in what polygon

```
extracted.values1 <- terra::extract(pollution, districts.newproj)
extracted.values2 <- terra::extract(pollution, districts.newproj, weights=TRUE)
extracted.values3 <- terra::extract(pollution, districts.newproj, fun=mean)
```

Class.r* × extracted.values1

Filter

	ID	pollution
1	1	6.8
2	1	8.0
3	1	6.6
4	1	6.6
5	1	5.9
6	1	5.7
7	1	5.9
8	1	5.6
9	1	5.5
10	1	6.0

Class.r* × extracted.values2 × extracted.v

Filter

	ID	pollution	weight
1	1	6.1	0.0013409962
2	1	6.8	0.6609195545
3	1	8.0	0.6858237696
4	1	7.9	0.3408046051
5	1	NA	0.0099616860
6	1	5.9	0.4005747213
7	1	6.6	0.8365900564
8	1	6.6	1.0000000216
9	1	6.0	0.3249042216

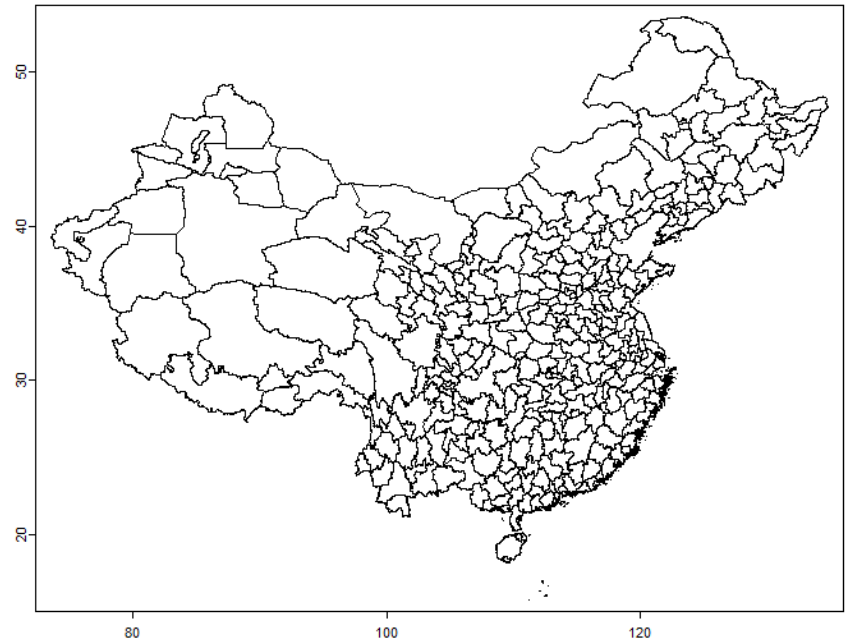
.r* × extracted.values3

Filter

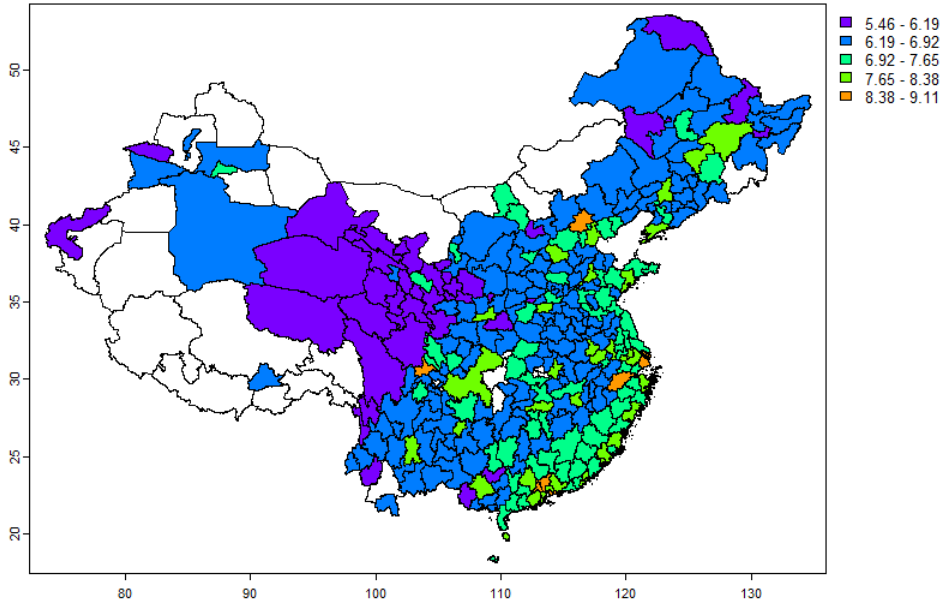
	ID	pollution
1	1	6.041176
2	2	5.400000
3	3	6.616667

5. Drawing maps

- plot(CHN_adm2)

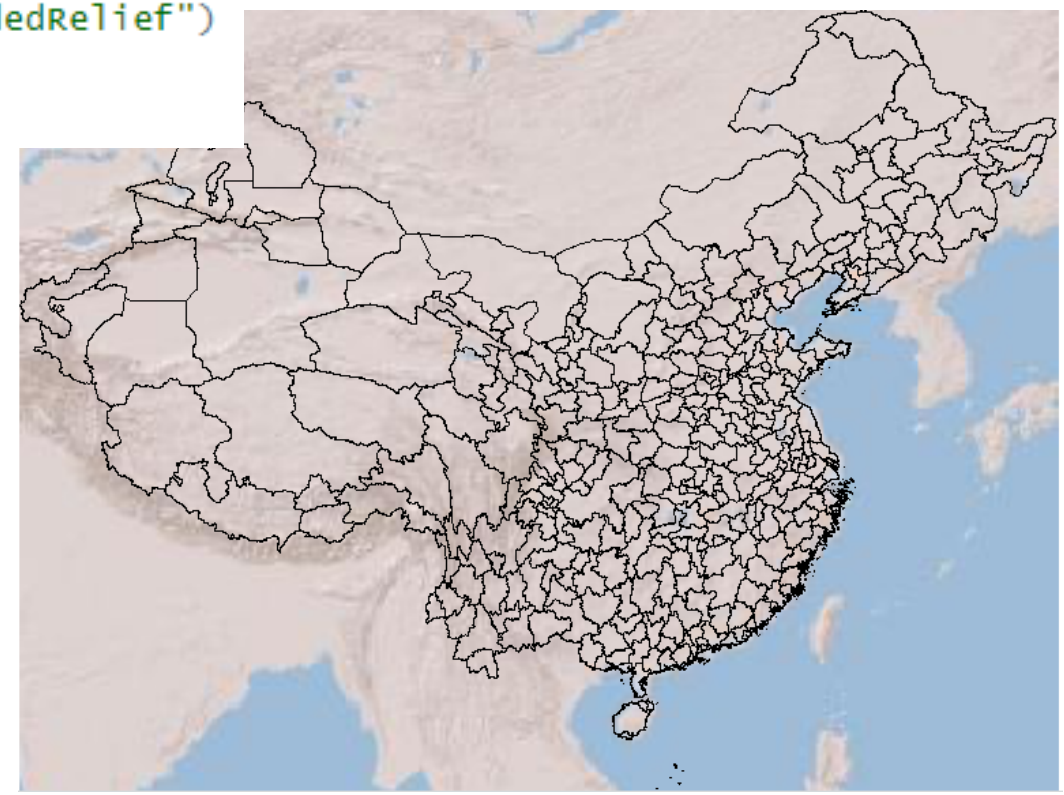


- plot(CHN_adm2, “lncount”)



Drawing maps

```
chn_file = "../GISdata/CHN_adm/CHN_adm2.shp"

x<-vect(district_file)
x<-vect(kommun_file)
x<-vect(chn_file)
bg <- get_tiles(x, crop=TRUE , provider = "Esri.WorldShadedRelief")
plotRGB(bg)
plot(x, add = TRUE, col = "transparent")
```



Selecting by clicks

```
# Find name of one element by clicking
click(CHN_adm2, n=1)

# Extract one element
s <- sel(CHN_adm2) # now click on the map twice
s$NAME_2
plot(s)

# Select records by variable info
i <- which(CHN_adm2$NAME_2 == 'Chongqing')
s<- CHN_adm2[i,]
plot(s)
```

6. Geometric Manipulations: Distances, Neighbors, Buffers, Intersections, etc.

- How many cities are within 10km of a drone strike?
  Or how many people live close to a government project?

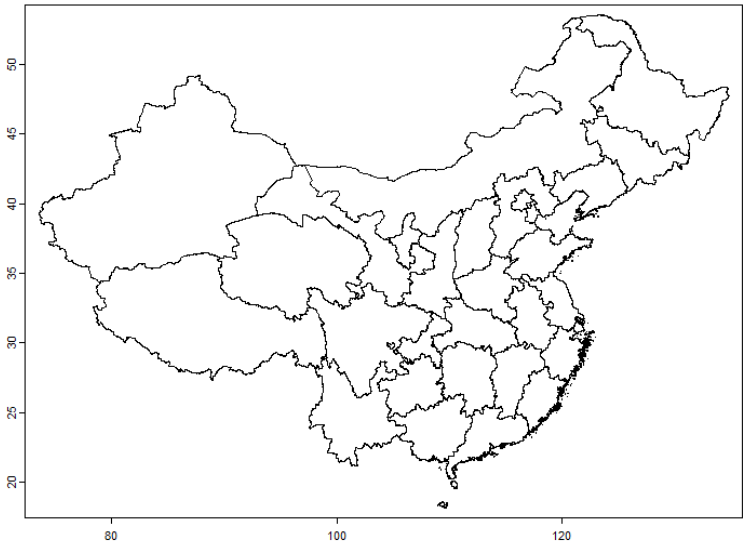
```
# Distance between centroids.
CHN_adm2_centroids <- centroids(CHN_adm2)
crs(CHN_adm2_centroids)
dist<-distance(CHN_adm2_centroids)

# Identifying neighboring areas
neighbors<-adjacent(CHN_adm2, type="rook", pairs=TRUE, symmetrical=TRUE)
#One of "rook", "touches", or "intersects".
# "rook" exclude polygons that touch at a single node only.
# "intersects" includes polygons that touch or overlap
```

lass.r* × neighbors ×

Filter

	from	to
1	1	7
2	1	111
3	1	114
4	1	118
5	2	5
6	2	9
7	2	10
8	2	218
9	2	219
10	2	220
11	2	222
12	2	223
13	2	242
14	2	269

Aggregate




```
# Aggregate to province level
CHN_province <- aggregate(CHN_adm2, by='NAME_1')
plot(CHN_province)

# Means of variables are computed by default
CHN_province_data<-as.data.frame(CHN_province)

# You can set it to other functions: "mean", "max", "min", "median", "sum", etc.
CHN_province <- aggregate(CHN_adm2, by='NAME_1', fun="sum")
CHN_province_data<-as.data.frame(CHN_province)
```

Class.r* × | CHN_province_data1 × | CHN_provin

Filter

	NAME_1	mean_ID_2	mean_ID_0
1	Anhui	9772.00	45
2	Beijing	9781.00	45
3	Chongqing	9782.00	45
4	Fujian	9787.00	45
5	Gansu	9798.50	45

Class.r* × | CHN_province_data1 × | CHN_province_data2 ×

Filter

	NAME_1	sum_ID_2	sum_ID_0	sum_ID_1
1	Anhui	166124	765	9707
2	Beijing	9781	45	572
3	Chongqing	9782	45	573
4	Fujian	88083	405	5166
5	Gansu	137179	630	8050

Some useful terra functions

- Calculating Properties
  - perim: calculates length of line / perimeter of polygons
  - expanse: calculate area of a polygon
  - relate: geometric relationships such as "intersects", "overlaps", and "touches“
  - adjacent: Identify cells that are adjacent to a set of raster cells or identify adjacent polygons
  - distance: distance between items
- Making New Shapes
  - buffer: Expand points into circles of given radius
  - centroid: Collapse polygons to their centroids
  - union, intersect: execute set operations on polygons
  - aggregate: dissolves a collection of shapes into a single shape.
- Testing Geometric Relationships
  - gIntersects: test if shapes intersect. Primarily useful for testing whether two polygons intersect, since this is not somethingover can do.
  - gContains: Is one spatial object entirely within another?
  - gIsValid: Very useful – make sure your geometries aren’t corrupt!

Task 7b

1. The polygons of Swedish municipal borders “Kommun_RT90_region” (from # https://www.scb.se/sv_/Hitta-statistik/Regional-statistik-och-kartor/Regionala-indelningar/Digitala-granser/) are on Athena.
   Read them into R. What type of object is it? What is the CRS? Plot the borders.
2. The Swedish railway lines ”jl_riks” (from https://www.lantmateriet.se/sv/Kartor-och-geografisk-information/Kartor/oppna-data/hamta-oppna-geodata) are on Athena.
   Load jl_riks into R. What type of object is it? What is the CRS. Plot the railways together with the municipal borders.