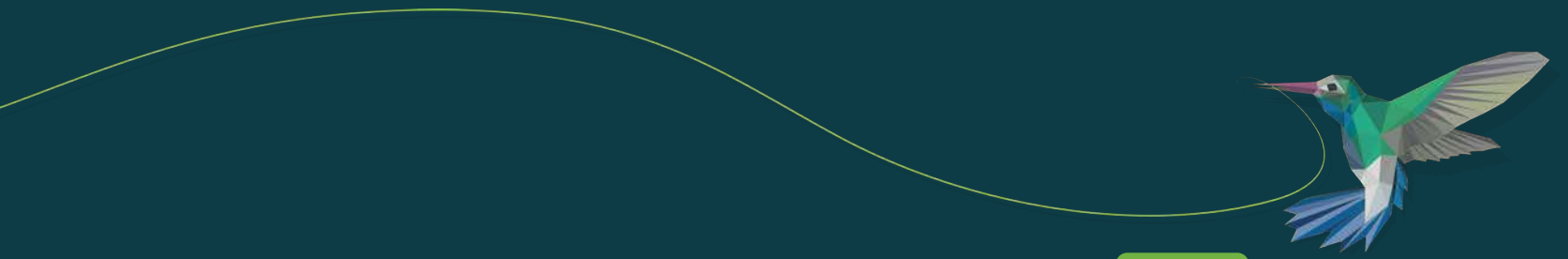
Coordinate Reference Systems (CRS)



ZevRoss
Know Your Data

Can't we just skip this section?

Please!?

You might avoid the pitfalls at first...

But sooner or later...

Problems will arise

- Layers won't match up
- Distances won't be correct
- Circles turn to ellipses for no apparent reason

I guarantee that this section will pay dividends in your future work

Motivating example

Map these together

```
landmarks <- read_sf("landmarks.geojson")
boroughs  <- read_sf("boroughs.shp")
```

Landmarks plot just fine

```
st_geometry(landmarks) %>%
  plot(pch = 16, col = "green4", cex = 2)
```

Boroughs also plot fine

```
st_geometry(boroughs) %>% plot()
```

Plot both layers together

I promise this code is correct...

```
plot(st_geometry(boroughs))
plot(st_geometry(landmarks), add = TRUE)
```

Dude where are my points!

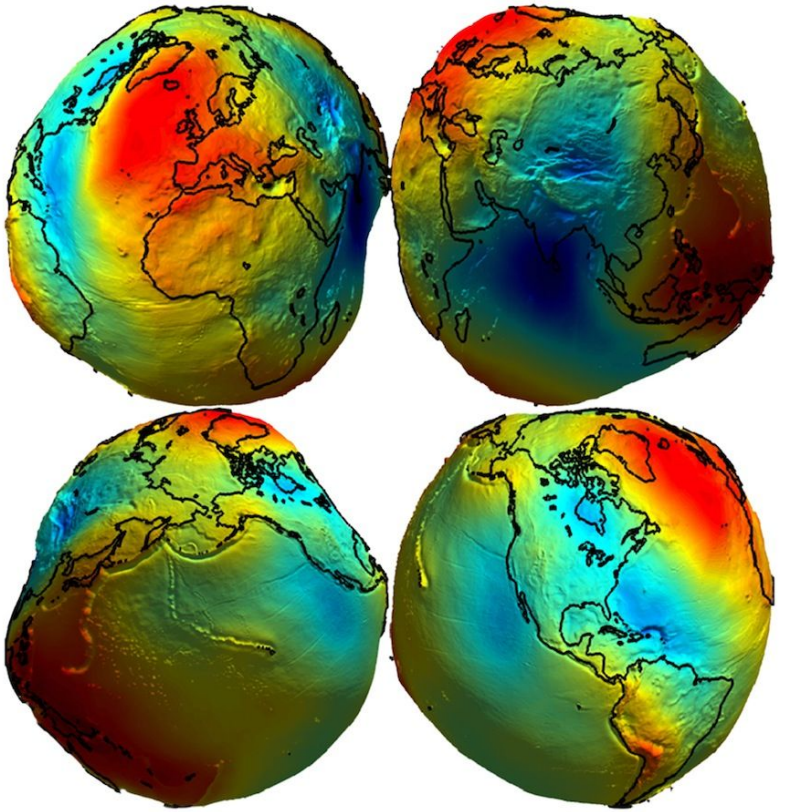
CRS mismatch is the most likely culprit

Knowing about coordinate reference systems is important

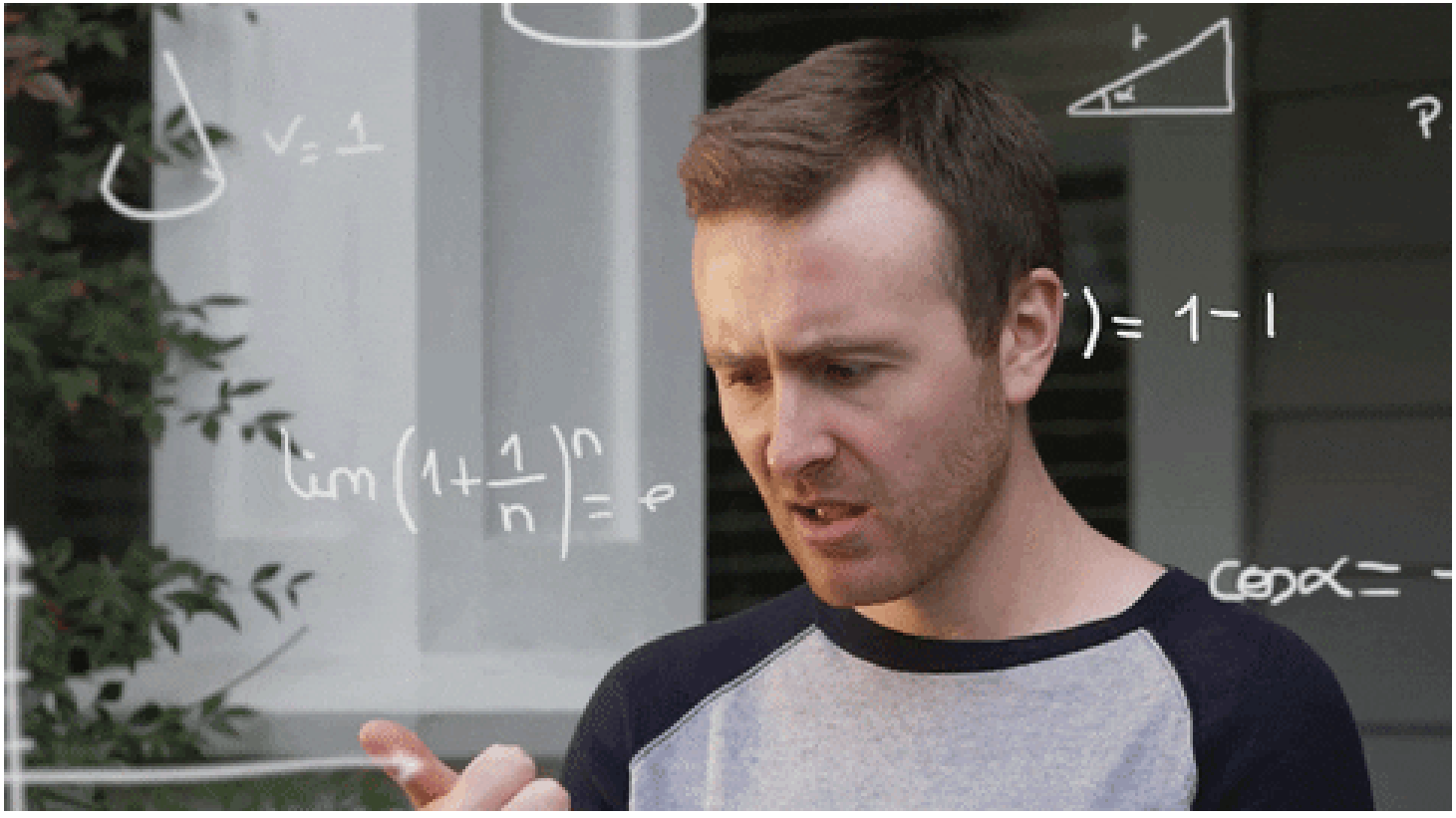
A nice reference from QGIS

Big picture

The earth is not a perfect sphere -- it's a spheroid/ellipsoid



The earth's shape is defined by a mathematical formula



And that formula has changed through time

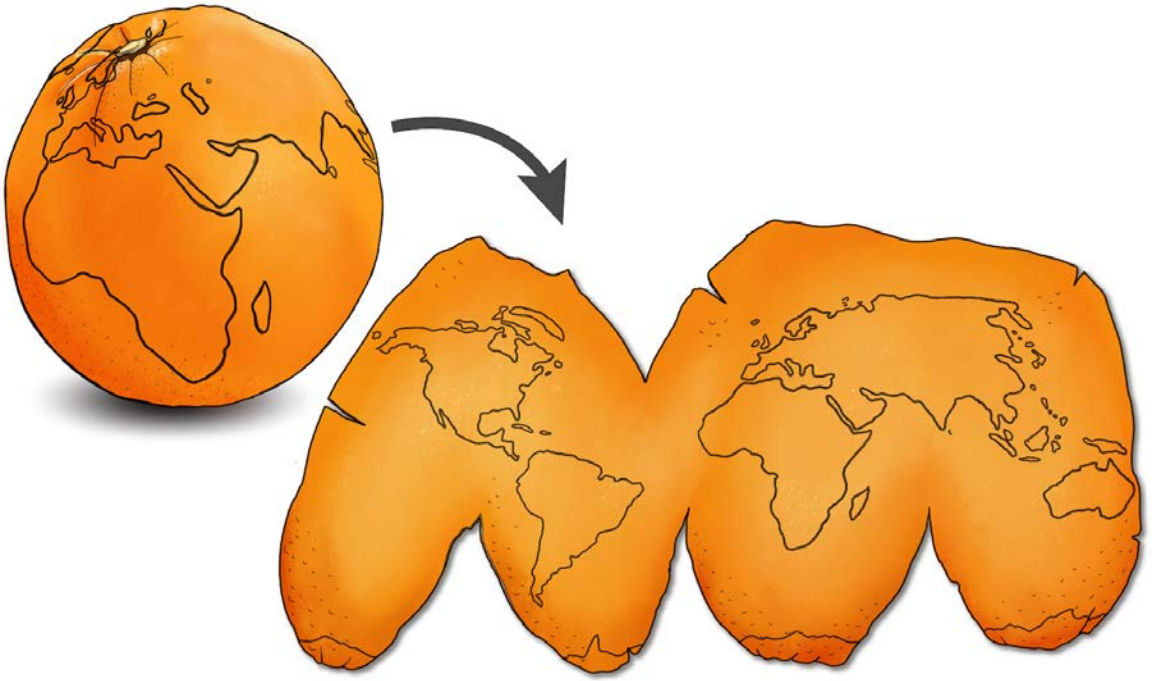
A "geographic" coordinate reference system

The spheroid definition is part of a geographic coordinate system -- a way to define locations on the earth

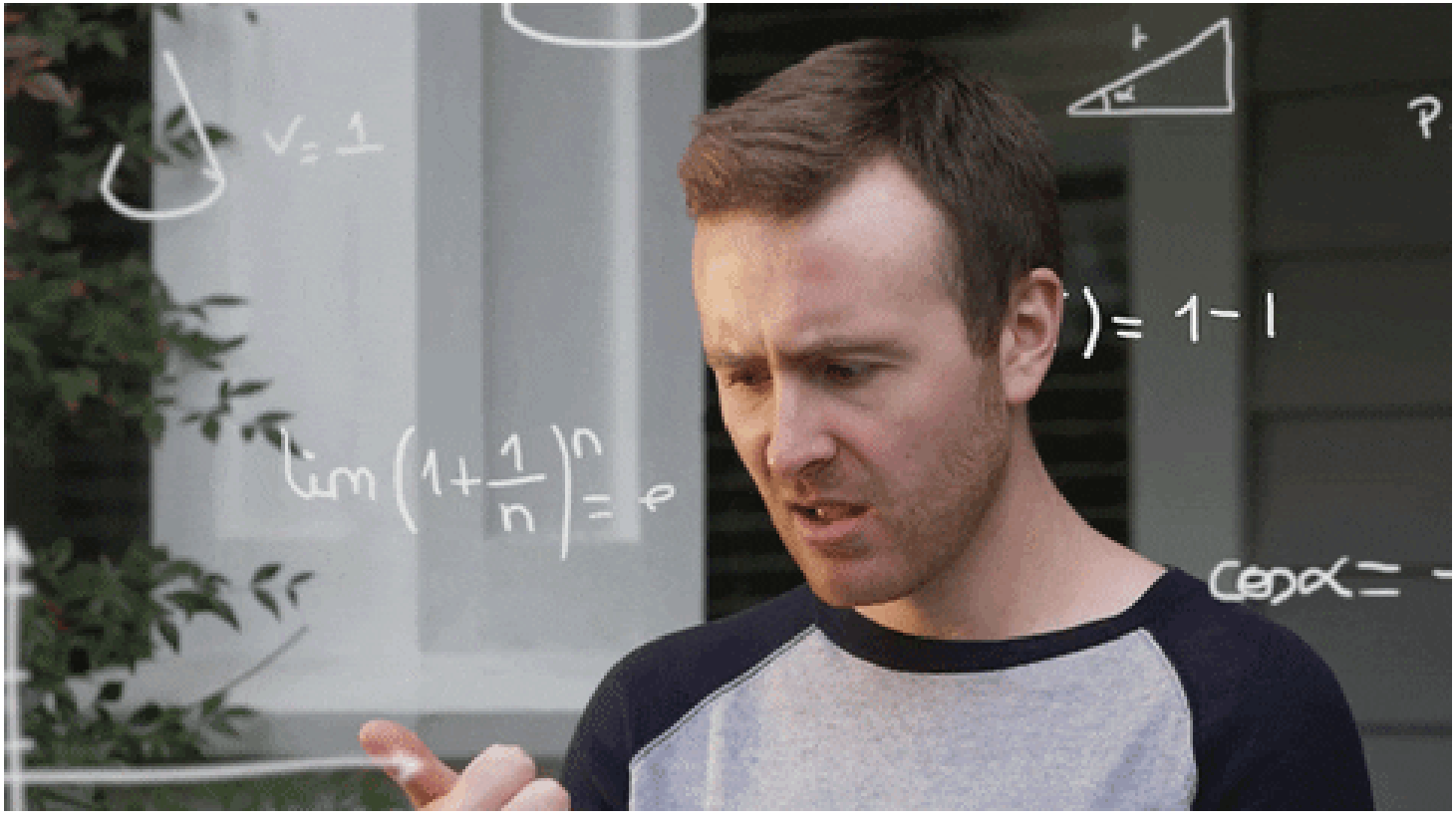
With a geographic CRS coordinates are in latitude, longitude

3-D to flat map

Converting to a 2-D map is not so straightforward



Another mathematical formula!



This conversion from 3-D to 2-D is called a projection

Or a "projected coordinate reference system"

But no projection is perfect

There are always distortions of shape, distance and relative angles

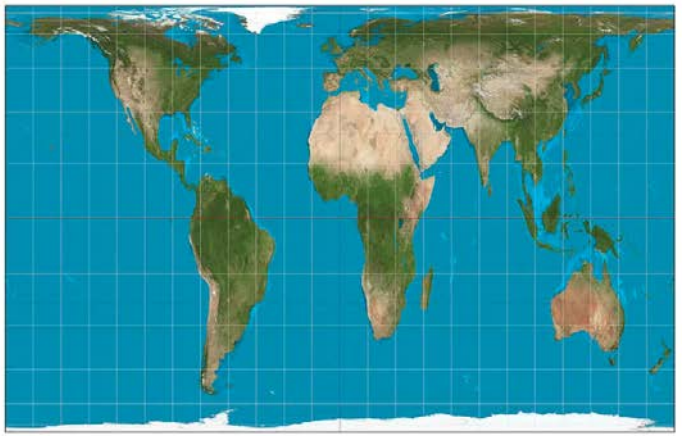
So you pick a projection based on what you're doing

- One that minimizes distortion of the type you care about
- For example, local distance measurement vs shower curtain global map?

Famous example, the mercator vs peters projection

Mercator

Peters



For mapping and geoprocessing you should use a projected CRS

- Otherwise R/GIS will "flatten" a geographic CRS resulting in tons of distortion
- Many R functions require a projected CRS (computations are much easier with a projected CRS)

CRS can be described by a pre-defined string or a code

Both unprojected and projected CRS have a set of parameters

A CRS can be specified using a "proj4string"

```
"+proj=longlat +ellps=WGS84 +no_defs"
```

An unprojected (geographic) CRS has the word `longlat`

```
"+proj=longlat +ellps=WGS84 +no_defs"
```

And most CRS have a short-hand "EPSG" code

```
4326
```

The proj4string for a projected CRS is much longer

This is the EPSG for State Plane Long Island (feet)

```
"+proj=lcc +lat_1=41.03333333333333 +lat_2=40.66666666666666
+lat_0=40.16666666666666 +lon_0=-74 +x_0=300000.0000000001
+y_0=0 +ellps=GRS80 +to_meter=0.3048006096012192 +no_defs"
```

But this one also has an EPSG code

```
2908
```

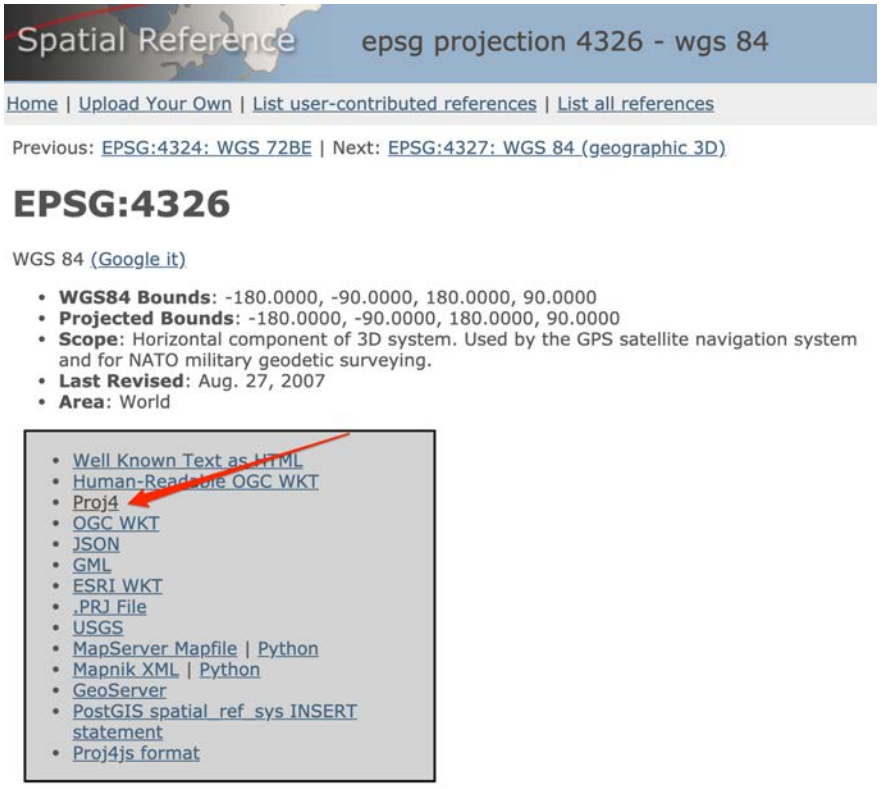
EPSG vs proj4string

- You can use either to assign/define a CRS in R for vectors
- For rasters you'll need the proj4string

A note for the future...

- There are some big changes upcoming in how CRS are defined
- There will be a move away from proj4strings
- The implications of this for {sf} and {raster} are uncertain at the moment

spatialreference.org is useful



What CRS should I use!?

Thankfully, in most cases you do not have to decide for yourself

- In most cases, the data you receive will already have a CRS you can use
- If it doesn't come with a CRS you can ask the source of the file what the CRS is

When do you have to choose a CRS yourself?

- If the data you receive is unprojected (geographic) and you want to use a projected CRS.

Nice guidance in Geocomputation with R

A good discussion by Robin Lovelace, Jakub Nowosad and Jannes Muenchow here.

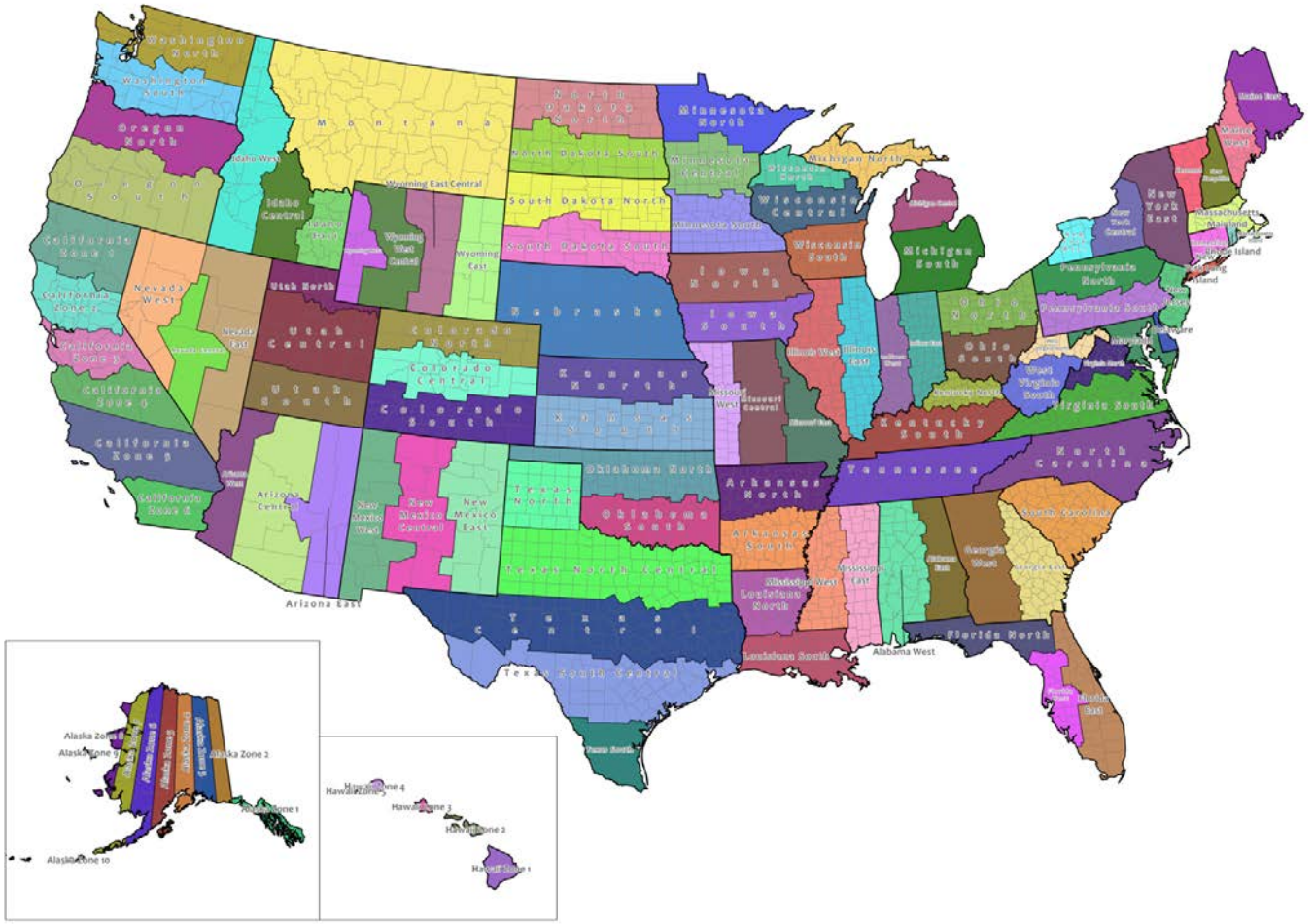
If you can, find out what government agencies are using for the area of interest

- This is usually step 1 for me

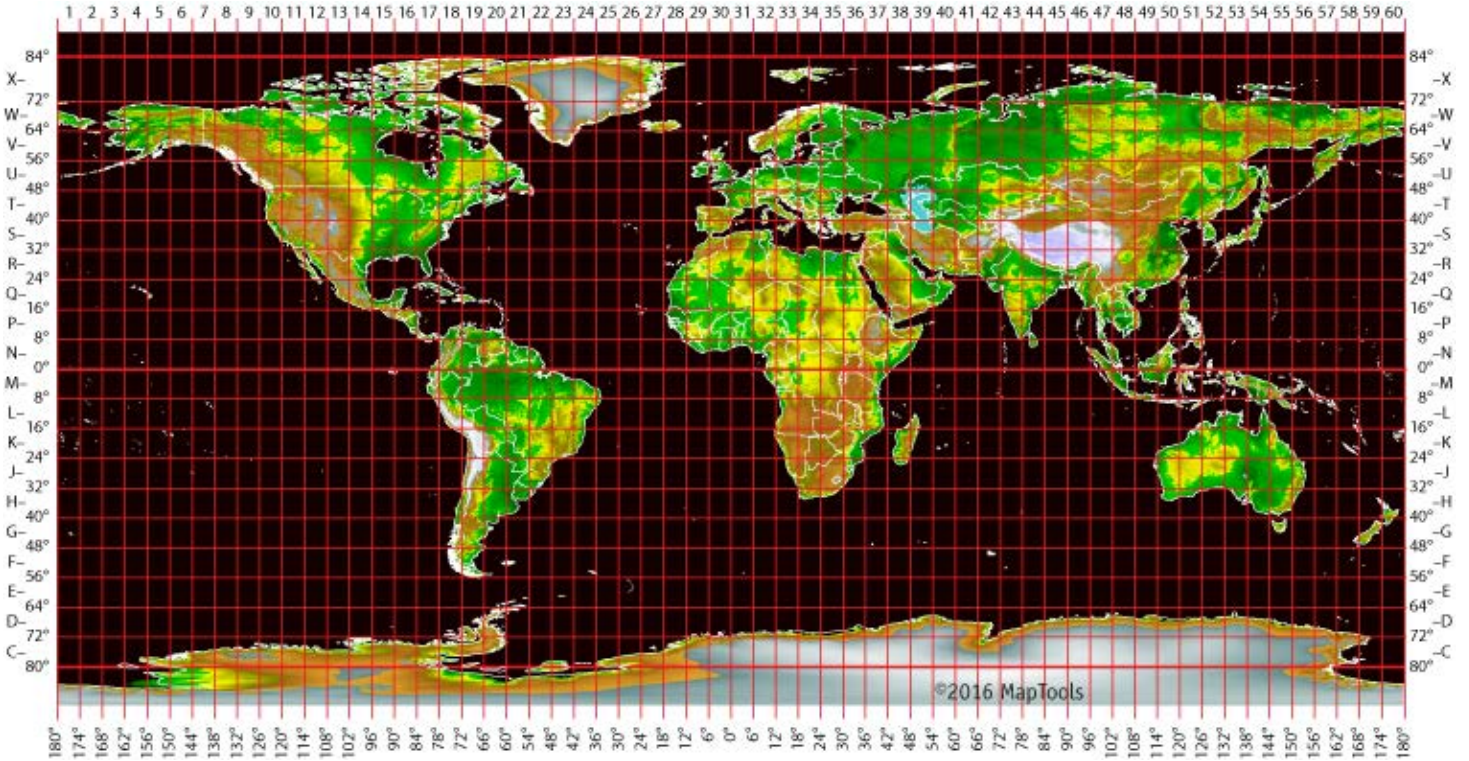
I often google "best projection for [area of interest]"

- This is often step 2 for me

I use state plane for relatively small US areas



I use UTM zones for global areas



https://www.maptools.com/tutorials/grid_zone_details

For both state plane and UTM, again, I often google:

- "State plane zone for San Francisco"
- "UTM zone for Ghana"

The CRS in R

Hopefully your spatial data come with a CRS

- Usually this is in the metadata of files you receive

{sf} and {raster} will read the CRS if it exists in the metadata

- `st_read()` will read the CRS
- `raster()` and `brick()` will read the CRS

Once your data is in R you can look at the CRS

- `st_crs()` for vectors
- `crs()` for rasters

An example with vectors

```
boroughs <- read_sf("boroughs.gpkg")
landmarks <- read_sf("landmarks.gpkg")
schools <- read_sf("schools.gpkg")
```

An example with vectors

```
st_crs(boroughs)
```

```
## Coordinate Reference System:
##   No EPSG code
##   proj4string: "+proj=lcc +lat_1=41.03333333333333 +lat_2=40.6666
```

```
st_crs(landmarks)
```

```
## Coordinate Reference System:
##   EPSG: 4326
##   proj4string: "+proj=longlat +datum=WGS84 +no_defs"
```

Trick question: Does this layer **have** a CRS?

```
st_crs(schools)
```

```
## Coordinate Reference System: NA
```

Answer: Yes but it's not defined in R

- All spatial data *has* a CRS but it's not always in the metadata
- Go back to the original source and find out what it is
- Or, gulp, make an educated guess

For points with lat/long coordinates you can generally assume WGS84

Latitude values range between -90 to 90 and longitude ranges between -180 to 180

```
##           X        Y
## 1 -74.01177 40.64915
## 2 -73.98576 40.60125
## 3 -73.97096 40.57757
## 4 -73.99175 40.59758
## 5 -74.02895 40.63418
## 6 -74.00362 40.63315
```

With much larger values the CRS is not geographic

```
##          X        Y
## 1 980985.1 175780.8
## 2 988205.1 158329.6
## 3 992317.3 149703.0
## 4 986541.2 156991.8
## 5 976215.3 170325.0
## 6 983246.6 169950.6
```

If WGS84 doesn't work you probably need to contact the creator of the data

In the example of a spatial file without a specified CRS

- Once you determine what the CRS should be
- You assign/define the CRS

Assign/Define the CRS for vectors with `st_crs()`

```
# Using a proj4string
st_crs(schools) <- "+proj=lcc +lat_1=41.03333333333333+
lat_2=40.66666666666666 +lat_0=40.16666666666666 +
lon_0=-74 +x_0=300000 +y_0=0 +ellps=GRS80 +towgs84=
0,0,0,0,0,0,0 +units=us-ft +no_defs"
```

```
# Using an EPSG code
st_crs(schools) <- 2908
```

You can even use an existing layer to assign/define

Take note, this is useful!

```
st_crs(schools) <- st_crs(boroughs)
```

For rasters use `crs()` in the same way

- Except `crs()` does not accept an EPSG code as a number

For example...

```
canopy <- raster("canopy.tif")
```

Get the CRS

```
crs(canopy)
```

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

I went to the website and looked at the documentation for the CRS

Assign/Define the CRS for rasters with `crs()`

```
# Requires proj4string
crs(canopy) <- "+proj=lcc +lat_1=41.03333333333333 +lat_2=40
```

If you want to use an EPSG code with `crs()`

```
crs(canopy) <- "+init=epsg:2908"
```

Subtle distinction between `st_crs()` and `crs()` output

With `st_crs()` you can extract EPSG and proj4string

```
vect_crs <- st_crs(schools)
```

```
vect_crs$epsg
```

```
## [1] 2908
```

```
vect_crs$proj4string
```

```
## [1] "+proj=lcc +lat_1=41.03333333333333 +lat_2=40.66666666666666
```

With `crs()` you can only get the proj4string

```
rast_crs <- crs(canopy)
```

```
rast_crs@projargs
```

```
## [1] "+proj=lcc +lat_1=41.03333333333333 +lat_2=40.66666666666666
```

Changing your layer's CRS

Often this is referred to as "projecting"

An example might be converting WGS84 to a projected CRS

Change the CRS in R

- Use `st_transform()` for vectors
- Use `projectRaster()` for rasters

Back to our motivating example

Remember these two have different CRS

```
st_crs(boroughs) # projected
```

```
## Coordinate Reference System:
##   No EPSG code
##   proj4string: "+proj=lcc +lat_1=41.03333333333333 +lat_2=40.6666
```

```
st_crs(landmarks) # unprojected
```

```
## Coordinate Reference System:
##   EPSG: 4326
##   proj4string: "+proj=longlat +datum=WGS84 +no_defs"
```

We will choose to use the projected CRS

Transform landmarks to match boroughs

```
landmarks <- st_transform(landmarks,
  crs = st_crs(boroughs))
```

Do they map now?

```
plot(st_geometry(boroughs))
plot(st_geometry(landmarks), add = TRUE,
     pch = 16, col = "red")
```

One more great thing about {tmap}

```
landmarks <- st_transform(landmarks, crs = 4326)
st_crs(landmarks)
```

```
## Coordinate Reference System:
##   EPSG: 4326
##   proj4string: "+proj=longlat +datum=WGS84 +no_defs"
```

```
st_crs(boroughs)
```

```
## Coordinate Reference System:
##   No EPSG code
##   proj4string: "+proj=lcc +lat_1=41.03333333333333 +lat_2=40.6666
```

Projections on the fly!

```
tm_shape(boroughs) + tm_polygons() +
  tm_shape(landmarks) + tm_dots(size = 0.3)
```

But don't let this allow you to get sloppy

Most `sf` functions require the same CRS so this is good practice.

open_exercise(4)