Coordinate Reference Systems with sf

Workshop: I2DS Tools for Data Science

Lisa Pramann & Caitlin Sarro
Hertie School
November 4th 2021

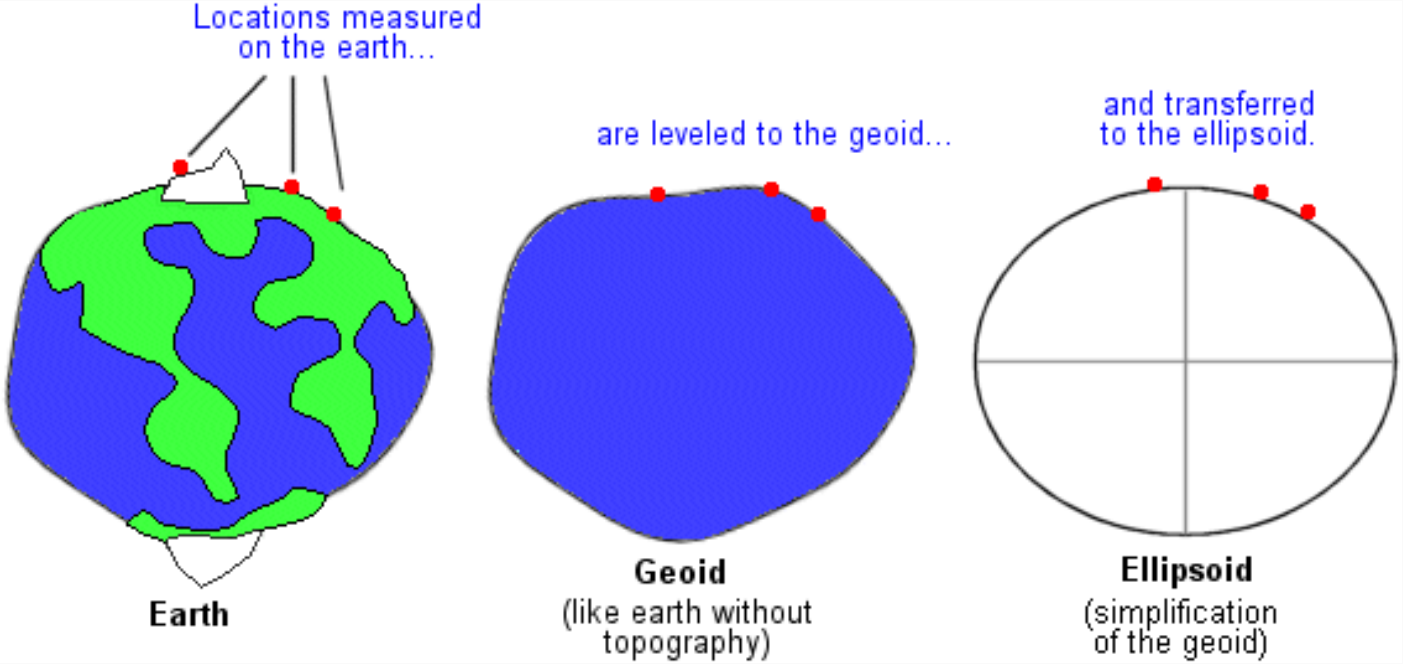
Agenda

1. Modelling the world
2. What are CRS?
3. CRS and Datums
4. CRS in R
   - 4.1 Two Key Features
   - 4.2 Spatial Data Operations
5. Further Resources

Modelling the World

How does the Earth really look like?

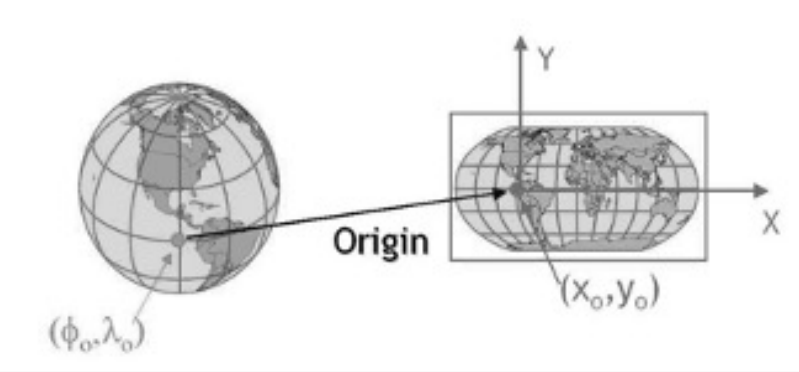
Refreshing you knowledge from your Geo-course during high school/undergrad-studies...



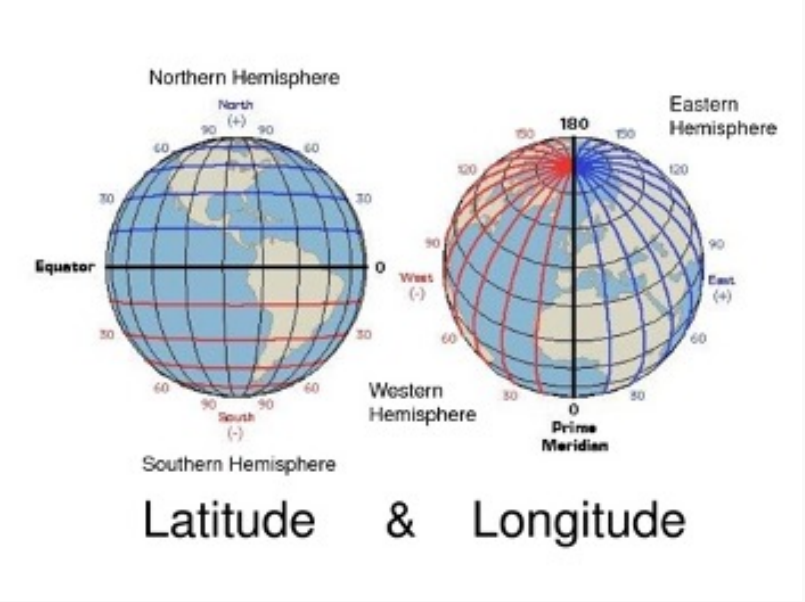
What is a “Coordinate Reference System” (CRS) ?

(= Geographic Coordination System or Spatial Reference System)

- Used to model the World
- Set locations
- Utilize
  - latitude (horizontal)
  - longitude (vertical)
- basis for planar coordinates and GIS (Geoinformation Systems)



Sreedevi, 2021



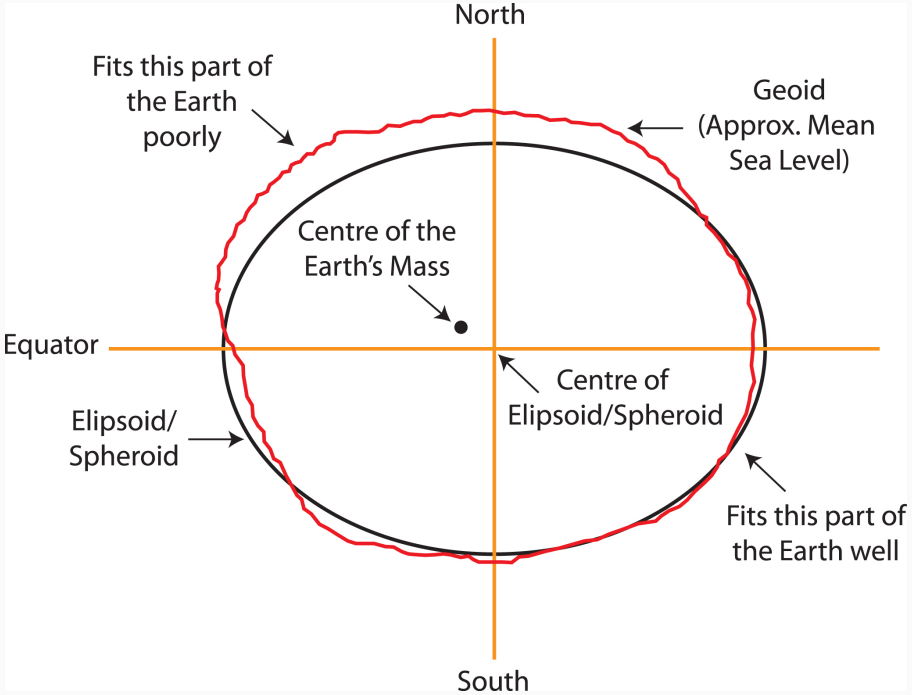
Mike Mitter, 2019

CRS and Datums

As you have seen the Ellipsoid cannot model the earth (not even the Geoid) completely

Therefore, different Coordinate Reference Systems exist:

- **World Geodetic System**
  - (WGS84, EPSG:4326)
  - approximates the whole earth
  - standard model
- **North American Datum**
  - (NAD83, EPSG:6269)
- **Australian Geodetic Datum**
  - (AGD84, EPSG:420)
- Datums can usually be converted to one another.



IOSM Austrailia, 2021

Introduction to sf package

Simple Features (sf) describe how objects in the real world (such as a building or a tree) can be represented in computers.

Features have a *geometry* describing *where* on Earth the feature is located, and they have attributes, which describe other properties.

These geometries are represented by points, lines or polygons, or collections thereof (no curves).

```
# A tibble: 155 x 5
   cadmium copper  lead  zinc       geometry
     <dbl>  <dbl> <dbl> <dbl>    <POINT [m]>
 1    11.7   85.0   299  1022 (181072 333611)
 2     8.6   81.0   277  1141 (181025 333558)
 3     6.5   68.0   199   640 (181165 333537)
 4     2.6   81.0   116   257 (181298 333484)
 5     2.8   48.0   117   269 (181307 333330)
 6     3   61.0   137   281 (181390 333260)
 7     3.2   31.0   132   346 (181165 333370)
 8     2.8   29.0   150   406 (181027 333363)
 9     2.4   37.0   133   347 (181060 333231)
10     1.6   24.0    80   183 (181232 333168)
# ... with 145 more rows
```

Geocomputation with R, 2021

Example

Let's look at how data is catagorized in a simple feature:

```
R> class(nc) #nc is a shapefile for North Carolina
```

```
## [1] "sf"         "data.frame"
```

```
R> attr(nc, "sf_column")
```

```
## [1] "geometry"
```

CRS in sf

Let's look at how CRSs are stored in R spatial objects and how they can be set. For this, we need to read-in a vector dataset:

```
R> vector_filepath ← system.file("shapes/world.gpkg", package = "spData")
R> new_vector ← read_sf(vector_filepath)
```

Our new object, new_vector, is a polygon representing a world map data from Natural Earth with a few variables from World Bank (?spData::world).

```
R> head(new_vector, 3)
```

```
## Simple feature collection with 3 features and 10 fields
## Geometry type: MULTIPOLYGON
## Dimension:     XY
## Bounding box:  xmin: -180 ymin: -18.28799 xmax: 180 ymax: 27.65643
## Geodetic CRS:  WGS 84
## # A tibble: 3 x 11
##   iso_a2 name_long  continent region_un subregion type  area_km2     pop lifeExp
##   <chr>  <chr>      <chr>     <chr>     <chr>     <chr>    <dbl>   <dbl>   <dbl>
## 1 FJ     Fiji       Oceania   Oceania   Melanesia Sove~   19290.  8.86e5    70.0
## 2 TZ     Tanzania   Africa    Africa    Eastern ~ Sove~  932746.  5.22e7    64.2
## 3 EH     Western S~ Africa    Africa    Northern~ Inde~   96271. NA         NA
## # ... with 2 more variables: gdpPercap <dbl>, geom <MULTIPOLYGON [°]>
```

Key Features (1)

1 | Retrieve coordinate reference system from object

In sf the CRS of an object can be retrieved using st_crs().

```
R> st_crs(new_vector) # get CRS
```

```
Coordinate Reference System:
  User input: WGS 84
  wkt:
GEOGCRS["WGS 84",
    DATUM["World Geodetic System 1984",
        ELLIPSOID["WGS 84",6378137,298.257223563,
            LENGTHUNIT["metre",1]]],
    PRIMEM["Greenwich",0,
        ANGLEUNIT["degree",0.0174532925199433]],
    CS[ellipsoidal,2],
        AXIS["geodetic latitude (Lat)",north,
            ORDER[1],
            ANGLEUNIT["degree",0.0174532925199433]],
        AXIS["geodetic longitude (Lon)",east,
            ORDER[2],
            ANGLEUNIT["degree",0.0174532925199433]],
    USAGE[
        SCOPE["Horizontal component of 3D system."],
        AREA["World."],
        BBOX[-90,-180,90,180]],
    ID["EPSG",4326]]
```

Key Features (2)

1 | Retrieve coordinate reference system from object

The st_crs function also has one helpful feature – we can retrieve some additional information about the used CRS.

For example, try to run:

```
R> st_crs(new_vector)$IsGeographic #to check if the CRS is geographic or not
```

```
## [1] TRUE
```

```
R> st_crs(new_vector)$units_gdal #to find out the CRS units
```

```
## [1] "degree"
```

```
R> st_crs(new_vector)$srid #extracts its SRID (projection code common with GIS)
```

```
## [1] "EPSG:4326"
```

```
R> st_crs(new_vector)$proj4string #extracts the proj4string representation
```

```
## [1] "+proj=longlat +datum=WGS84 +no_defs"
```

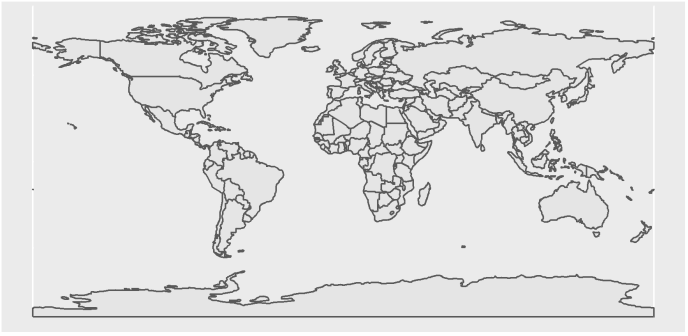
Key Features (3)

2 | Add or Change CRS

When the CRS is missing or the wrong CRS is set, the st_set_crs() function can be used. An important note is that replacing CRS does not re-project data and maintains data integrity.

```
R> # For transformations use 'st_set_crs'
R> CRS_vector ← sf::st_set_crs(world, "EPSG:4326") # set CRS
```

```
R> # Shifting the projection to center on Brussels using the CRS
R> world_view_brussels ← st_transform(world, crs = "+proj=laea +x_0=0 +y_0=0 +lon_0=50.8503396 +lat_0=4.3517103")
```



How to work with Spatial Data (1)

Some simple spatial data operations

1) **Geometric measurements**

- CRSs include information about spatial units, however it is generally difficult to work with units to do geometric calculations (see difference of latitude and longitude data and planar coordinate systems) -> the recent version of sf utilizes the units package which output is by default provided in m^2.

Lets see how that looks like for Belgium. We use the world data from the `spData` package:

```
R> Belgium ← world %>%
+   filter(name_long == "Belgium")
R> st_area(Belgium)
```

```
## 30019548314 [m^2]
```

The number appears really big, better to set the units to km^2:

```
R> units::set_units(st_area(Belgium), km^2)
```

```
## 30019.55 [km^2]
```

How to work with Spatial Data (2)

Some simple spatial data operations

2) **Geometric measurements**

- sf can also measure distances between two points based on geometric data.
  - The package `spData` contains geometric data on London's districts `lnd` and a dataset of cycle hire points in London `cycle_hire`.
  - We want to measure the distance from the cycle hire point at the Palace Gate to the centre of Redbridge.
  - The default of `st_distance()` is in m. We could use /1000 to get km.

```
R> cycle_station_palace ← cycle_hire %>%
+   filter(name == "Palace Gate")
R> redbridge ← lnd %>%
+   filter(NAME == "Redbridge")
R> redbridge_centroid ← st_centroid(redbridge) #gives central point in selected district
R> st_distance(cycle_station_palace, redbridge_centroid)
```

```
## Units: [m]
##          [,1]
## [1,] 20258.6
```

How to work with Spatial Data (3)

Creating maps with geometric data and sf

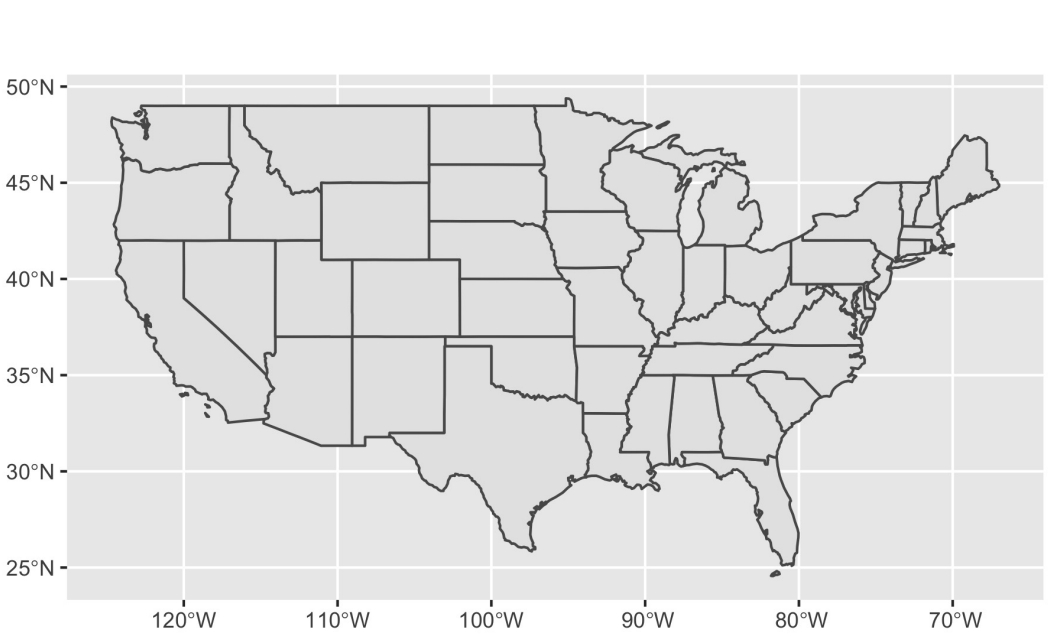
There a different ways to create maps based on geometric data.

]

1) One simple way is by using `ggplot()` in combination with `geom_sf()` from the `sf` package

Lets have a look at data from the US retrieved from the `spData` package:

```
R> ggplot() +
+ geom_sf(data = us_states) +
+   #For simple plots, you will only need geom_sf()
+ coord_sf(crs = st_crs(4326))
R> #ensures that all layers use a common CRS
R> #(would not be needed in this case)
```



How to work with Spatial Data (4)

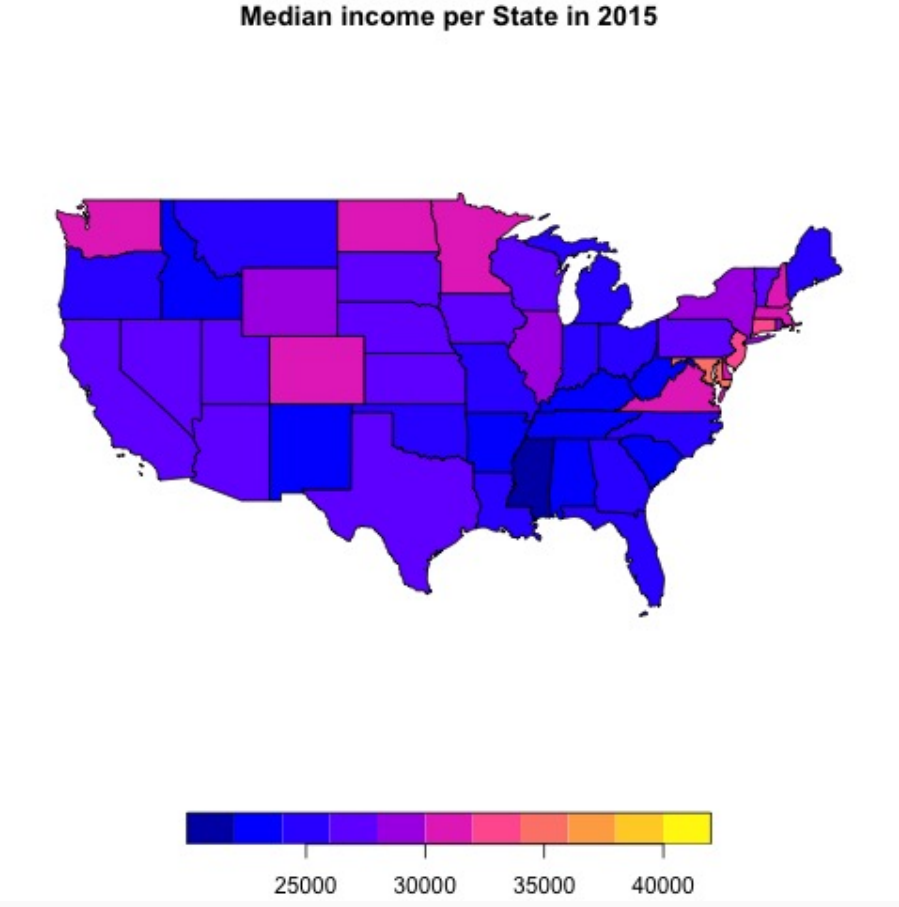
Creating maps with geometric data and sf

There a different ways to create maps based on geometric data.

2) `sf` also comes with a `plot()` function which creates a map of a sf object with one or more attributes. Lets have a look at data from the US retrieved from the `spData` package:

```
R> us_income ← left_join(
+   #joining two data frames from spData to
+   #attribute Median Income in 2015 to us_states
+   us_states, us_states_df,
+   by = c("NAME" = "state"))
```

```
R> us_income_plot ← plot(us_income["median_income_15"]
+      main = "Median income per State in 2015")
```



Conclusion

Things to take-away or have a closer look at

- `sf` offers a comprehensive way to access and model with CRS data

- Remember the most common CRS in the world (WGS84/ EPSG code: 4326) and experiment with different codes.

- *Public Policy* application: CRS is used in combination with other tools for visualization purposes by mapping certain data e.g. median income, voting behavior, COVID-vaccination rates, etc.

- Feel free to build on the skills in this session and have a look at how to use this for data visualization purposes

Further Links and Resources

Not yet enough of CRS and `sf`? Check out our resources and additional references:

Most `Credit` to Robin Lovelance Geocomputation with R, 2021

Presentation Tidy Spatial Data Analysis 2018

Blog WZB 2019 - Zooming in on maps with sf and ggplot2

Mapping in R with Emiliy Burchfield

Thank you!

Now onto the in-class session...