MXB262 Workshop Week 10 – Visualising spatial data

Kate Helmstedt and Jessie Roberts

10/3/2019

```r
# install and load the following packages.

#library(lwgeom)
library(cowplot)
library(ggplot2)
library(ggrepel)
library(ggspatial)
library(rnaturalearth)
library(rnaturalearthdata)
library(sf)
library(maps)
library(ggrepel)
library(maptools)
```

Spatial Visualisations - Maps

Spatial data format – `sf`, `ggplot`, `ggspatial`

The recently developed `sf` package combined with `ggspatial` allows 2D maps to be created using `ggplot2`.

The `ggspatial` package allows spatial data with longitude and latitude columns to be used with `ggplot2`. Genearlly, without this package, `ggplot2` does not handle spatial data well. the `geom_sf()` and `coord_sf()` functions in `ggplot2` (version 3) lets users pass simple features (spatial data from the `sf` package) objects as plottable and manipulable layers. You can read more about the package here.

The `sf` package enables simple spatial features to be added to data frames and tibbles (which are the two very similar R data types we have been using, which differ a little bit in shape). Thus transforming data frames with longitude/latitude coordinates or other spatial features into spatial data object. You can read more about the `sf` package here.

In this worksheet we will focus on data frames with coordinates. There are other types of spatial objects, but these will not be explored in this workshop.

Polygons (vs lines, points)

The basic elements of a map are: polygons, points, lines, and text (see lecture).

Polygons - are closed shapes such as state borders.
Lines - are linear shapes that are not filled, for example highways, streams, or roads.
Points - highlight specific positions, for example cities or landmarks. *text* - is used to annotate maps and add extra details.

Lets get started!

TASK: Ensure all packages area installed and loaded. TASK: Set the `ggplot2` theme to black and white. This is not always necessary (you can play with aesthetic choices in your own maps), but is great for for learning the methods since it improves the clarity of the maps.

```r
# set ggplot2 theme to black and white.
theme_set(theme_bw())
```

The package `rnaturalearth` is a data package that provides spatial data for the boundaries of the world at different scales. Use the `ne_countries()` function to pull country data and choose the scale (medium is generally sufficient for most applications). The `returnclass()` function can return outputs as either 'sp' (default) or 'sf' class objects. The `sf` class is needed to use `ggplto2` for mapping.

The `ne_coastline()` function can be used to pull coastline data in the same way as `ne_countries()` pulls the borders between countries.

TASK: create an object called `world` using the `ne_countries()` function. Check that it is in the right class (sf).

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

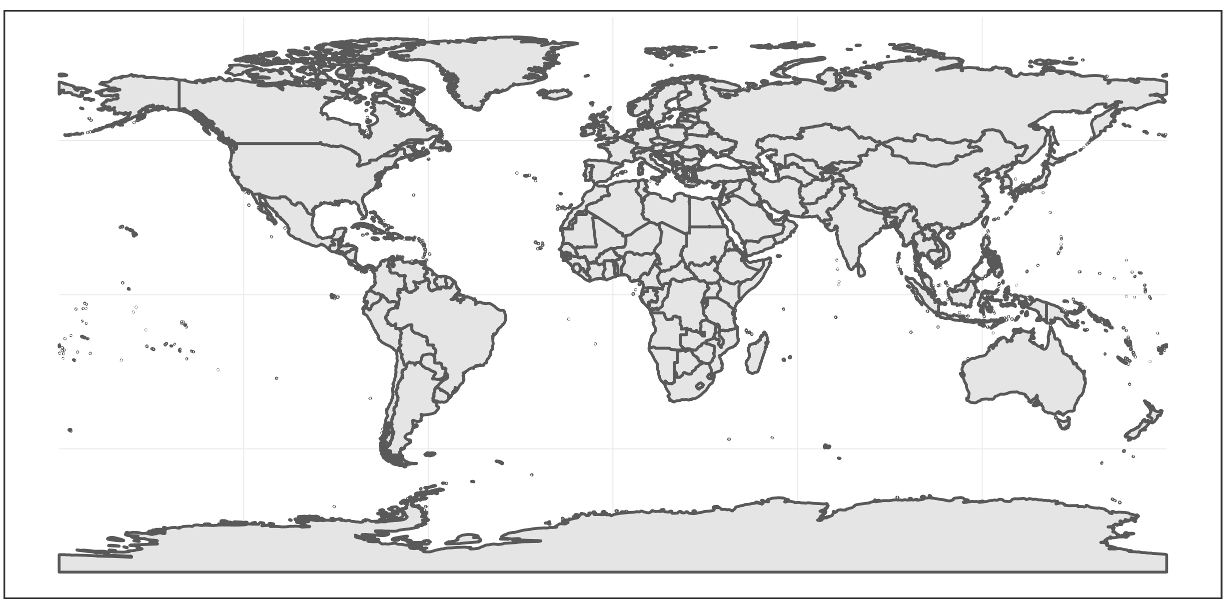
TASK: explore what data is now contained in the `world` object. Note the geometry type, the `bbox` (which gives the bounding box – the mapping word for the longitude and latitude at the boundaries of the data), and the data parameters you have for each country. All of these data are useful not only for creating detailed maps, but can also be pulled out, manipulated, and visualised in any way. This is a very useful data source for data and classifications of many types for the world.

Basic plots (using `geom_sf` in `ggplot`)

The basic `ggplot` grammar works the same for plotting spatial data as all the other plots we have seen before. The relevant geom is `geom_sf()`. As we will continue to explore below, we can layer all of the elements we know so well from `ggplot` into maps very easily through this same structure.

The basic call for creating maps is

```r
ggplot(data = world) +
    geom_sf()
```



TASK: explore other scales, and 'ne_coastline()' instead.

Titles and subtitles As in other `ggplot` graphics, `ggtitle()` can be used to add a title and subtitles. Axis names are absent by default, but can be added using `xlab()`.**

TASK: add the axis titles "Longitude" and "Latitude" in the relevant places (look back at the lecture if you're not sure what order they are in), and a title for the plot "World map".

Colour (`geom_sf()`)

Polygons on a map can be easily coloured in using the `fill =` and `colour =` arguments within the `geom_sf()` fuinction.

```
ggplot(data = world) +
    geom_sf(color = "black", fill = "lightgreen")
```

Colouring polygons by quantitative or qualitative data

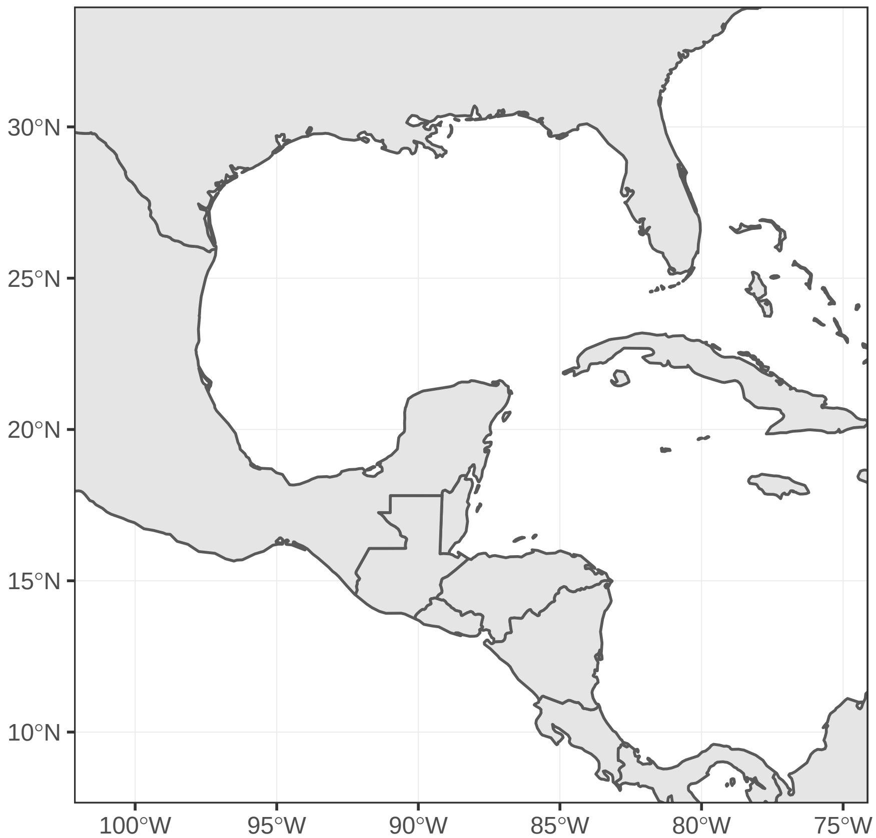
Polygons can also be coloured by a different variable. For example, `scale_fill_viridis_c()` is used to colour polygons by the square root of the population (which is stored in the variable `POP_EST` of the `world` object). This is a colorblind-friendly palette and is added by adding the `option = "plasma"` to the `scale_fill_viridis_c()` function. Try visualising with and without the `plasma` option.

```
ggplot(data = world) +
    geom_sf(aes(fill = pop_est)) +
    scale_fill_viridis_c(option = "plasma", trans = "sqrt")
```

Projection and extent

A map extent is the portion or area of the geography visible in the map. The map extent is defined using coordinate systems, which can be added using the `coord_sf()` function. This function can manipulate both the projection and extent of the map. `coord_sf()` can be used to zoom into an area of interest, using the `xlim =` and `ylim =` arguments. Note that the limits are automatically expanded by a fraction to ensure that data and axes don't overlap. This can be turned off to exactly match the limits provided using `expand = FALSE`.

```
ggplot(data = world) +
    geom_sf() +
    coord_sf(xlim = c(-102.15, -74.12), ylim = c(7.65, 33.97), expand = FALSE)
```



TASK: Create a map that shows all of Australia.

Scale bar and north arrow (`ggspatial` package)

We will use the `ggspatial` package to add a scale bar. There are other packages that can do this (e.g. `prettymapr`, `vcd`, `ggsn`, or `legendMap`), however `ggspatial` is much more user friendly and fits well within the `ggplot` grammar of graphics.

The location of the scale bar must be specified in longitude/latitude in the `lon =` and `lat =` arguments. The shaded distance inside the scale bar can be adjusted with the `distance_lon =` argument and its width is determined by the `distance_lat =` argument. These are not included in the example below, but you might like to explore these options.

Additional freatures that can be modified include: * Legend size - `legend_size =` (defaults to 3) * The North arrow behind the "N" symbol - `arrow_length =` * Arrow distance to the scale - `arrow_distance =` * Size the N north symbol itself with `arrow_north_size =` (defaults to 6)

Note that all distances are set to "km" by default, the `distance_unit =` argument can change to nautical miles ("nm") or miles ("mi").

```
ggplot(data = world) +
    geom_sf() +
     ggspatial::annotation_scale(location = "bl", width_hint = 0.5) +
     ggspatial::annotation_north_arrow(location = "bl", which_north = "true",
        pad_x = unit(0.75, "in"), pad_y = unit(0.5, "in")) +
    coord_sf(xlim = c(-102.15, -74.12), ylim = c(7.65, 33.97))
```

Note the warning message. Since the map uses unprojected data in longitude/latitude on an equidistant cylindrical projection, length in kilometers on the map directly depends mathematically on the degree of latitude. This means the scale bar does not perfectly represent distance through the whole figure (how often to we actually see that warning in real life maps? In what kinds of maps is this a bigger deal?).

Annotating

The `world` object we created above already contains country names and their centroid coordinates (the middle of the polygon). We can also use `geom_text()` to add notations.

Useful arguments of `geom_text()` are:

- `size =`
- horizontal and vertical alignment `hjust =` and `vjust =`. Which can be a number between 0 (right/bottom) and 1 (top/left) or a character ("left", "middle", "right", "bottom", "center", "top"). The text can also be offset horizontally or vertically with the `nudge_x` `nudge_y` arguments
- `color =` and `fontface =`
- `check_overlap =` checks for overlap in text. Alternatively, when there is a lot of overlapping labels, the package `ggrepel` provides a `geom_text_repel()` function that moves label around so that they do not overlap.
- `annotate()` is not an argument but an additional function that can be used to add a single character string at a specific location, as demonstrated in the Gulf of Mexico map below.

```
world_points<- st_centroid(world)
```

```
## Warning in st_centroid.sf(world): st_centroid assumes attributes are
## constant over geometries of x
```

```
## Warning in st_centroid.sfc(st_geometry(x), of_largest_polygon =
## of_largest_polygon): st_centroid does not give correct centroids for
## longitude/latitude data
```

```
world_points <- cbind(world, st_coordinates(st_centroid(world$geometry)))
```

```
## Warning in st_centroid.sfc(world$geometry): st_centroid does not give
## correct centroids for longitude/latitude data
```

```
ggplot(data = world) +
geom_sf() +
geom_text(data= world_points,aes(x=X, y=Y, label=name),
    color = "darkblue", fontface = "bold", check_overlap = FALSE) +
annotate(geom = "text", x = -90, y = 26, label = "Gulf of Mexico",
    fontface = "italic", color = "grey22", size = 6) +
coord_sf(xlim = c(-102.15, -74.12), ylim = c(7.65, 33.97), expand = FALSE)
```

Aesthetic elements

The map theme can be edited to make it more appealing. We suggested the use of `theme_bw()` for a standard theme, but there are many other themes that can be used (see `?ggtheme()`). Specific theme elements can also be adjusted using the following.

```
ggplot(data = world) +
  geom_sf(fill= "antiquewhite") +
  geom_text(data= world_points, aes(x=X, y=Y, label=name),
            color = "darkblue", fontface = "bold", check_overlap = FALSE) +
  annotate(geom = "text", x = -90, y = 26, label = "Gulf of Mexico",
           fontface = "italic", color = "grey22", size = 6) +
  ggspatial::annotation_scale(location = "bl", width_hint = 0.5) +
```

```r
ggspatial::annotation_north_arrow(location = "bl", which_north = "true",
                                  pad_x = unit(0.75, "in"), pad_y = unit(0.5, "in")) +
coord_sf(xlim = c(-102.15, -74.12), ylim = c(7.65, 33.97), expand = FALSE) +
xlab("Longitude") + ylab("Latitude") + ggtitle("Map of the Gulf of Mexico and the Caribbean Sea") +
theme(panel.grid.major = element_line(color = gray(.5), linetype = "dashed", size = 0.5),
      panel.background = element_rect(fill = "aliceblue"))
```

TASK: use your knowledge of colours and longitude/latitude to make small changes to this code to explore and identify what each small component means. Keep the code that you generate from this exploration in a fresh R script file, with a lot of comments (lines that begin with `#` and aren't evaluated) to yourself so you can remember what you've discovered. Comments are incredibly useful when your code will be read in the future by either you or someone else – rather than having to do all the exploration and thinking again, you can just look at your comments and remember what you noted while exploring the first time.

Saving the map (`ggsave()`)

The `ggsave()` function makes it easy to save these maps ina variety of formats including PNG (raster bitmap) and PDF (vector graphics), and provids control over the size and resolution. Below we save a PDF version of the map, with the best quality, as well as a PNG version for web purposes:

```r
ggsave("map.pdf")
```

```
## Saving 6.5 x 4.5 in image
```

```r
ggsave("map_web.png", width = 6, height = 6, dpi = "screen")
```

Now you've learnt the basics, let try a different map and adding a few more details.

Adding points using coordinates

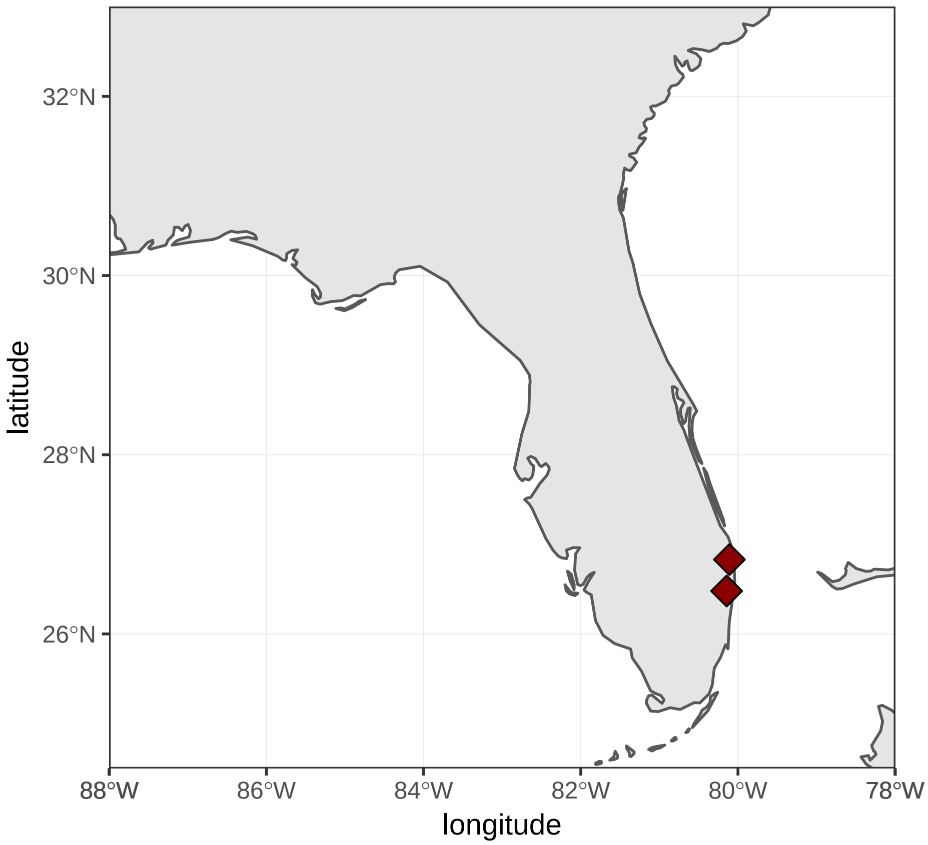
Let's start by defining two study sites according to their longitude and latitude and store them in a regular data.frame.

```r
sites <- data.frame(longitude = c(-80.144005, -80.109), latitude = c(26.479005,
    26.83))
```

Using the `geom_point()` function

The quickest way to add point coordinates is with `geom_point()`. Characteristics of the points can be adjusted as for any other use of `geom_point()` (e.g. color of the outline and the filling, shape, size, etc.). This can be achieved for all points or using groupings from the data (i.e defining their "aesthetics"). In this example, we zoom in a little further into the Gulf of Mexico and add two diamond shaped points (`shape = 23`), filled in dark red (`fill = "darkred"`) and increase the size (`size = 4`), Location defined above.

```r
ggplot(data = world) +
    geom_sf() +
    geom_point(data = sites, aes(x = longitude, y = latitude), size = 4,
        shape = 23, fill = "darkred") +
    coord_sf(xlim = c(-88, -78), ylim = c(24.5, 33), expand = FALSE)
```



This is quick and easy, and allows us some level of customisation. However, it doesn't allow us to embed data about the points into the map, it just lets us place them. If we want to use the data itself to inform things like colour and size of the points, we need a way to manipulate the data itself rather than just change plotting options.

Using the `sf` package

An alternative and more flexible way to add points that can refer back to the data is using the `sf` package.

Fist the data.frame must be converted to an `sf` object using the `st_as_sf()` function. The argument `coords =` should be fed the names of columns containing the coordinates. This can be very useful if the two objects (here `world` and `sites`) are not in the same projection. To achieve the same result the projection (WGS84, which is the CRS code #4326) has to be defined in the `sf` object.

Note: the `arg` argument in the map below stands for 'attribute-geometry-relationship'. It defines how the non-geometry columns relate to the geometry and it can have the values: "constant" (e.g. lang use), "aggregate" (e.g. population density or count) or "identity" (e.g. city name). In the example below the `world` object is the geometry and `sites` are specific locations. The `crs = 4326` argument defines the coorindate reference system (i.e. the projection and use of lat/long or other measurements). 4326 is the standard system for an elipsoid world map with lat/long data and can be used in all projects for this unit.

```
sites <- st_as_sf(sites, coords = c("longitude", "latitude"),
    crs = 4326, agr = "identity")
```

TASK: explore your new dataset `sites`. What data is included?

Plot using your new dataset, using two geoms – the first is the simple polygon data of the world, and the second adds our newly created data.

```r
ggplot(data = world) +
    geom_sf() +
    geom_sf(data = sites, size = 4, shape = 23, fill = "darkred") +
    coord_sf(xlim = c(-88, -78), ylim = c(24.5, 33), expand = FALSE)
```

Note that `coord_sf()` has to be called after all `geom_sf()` calls, in order to supercede any former input.

Adding Polygons

Polygons are used to show states, counties, electorates, suburbs or other relevant regions on a mal. The package `maps` (which is automatically installed and loaded with `ggplot2`) provides polygon data for US state and county boundaries.

States

```r
states <- st_as_sf(map("state", plot = FALSE, fill = TRUE))
head(states)
```

```
## Simple feature collection with 6 features and 1 field
## geometry type:  MULTIPOLYGON
## dimension:      XY
## bbox:           xmin: -124.3834 ymin: 30.24071 xmax: -71.78015 ymax: 42.04937
## epsg (SRID):    4326
## proj4string:    +proj=longlat +datum=WGS84 +no_defs
##                         geometry          ID
## 1 MULTIPOLYGON (((-87.46201 3...     alabama
## 2 MULTIPOLYGON (((-114.6374 3...     arizona
## 3 MULTIPOLYGON (((-94.05103 3...    arkansas
## 4 MULTIPOLYGON (((-120.006 42...  california
## 5 MULTIPOLYGON (((-102.0552 4...    colorado
## 6 MULTIPOLYGON (((-73.49902 4... connecticut
```

The ID variable contains the State names. A simple way to add State names to the map, is to compute the centroid of each state polygon and use this as the coordinates for the annotated names. Centroids are computed with the function `st_centroid()` and their coordinates extracted with `st_coordinates()`, both from the package `sf`, and attached to `states`

```r
states <- cbind(states, st_coordinates(st_centroid(states)))
```

```
## Warning in st_centroid.sf(states): st_centroid assumes attributes are
## constant over geometries of x
```

```
## Warning in st_centroid.sfc(st_geometry(x), of_largest_polygon =
## of_largest_polygon): st_centroid does not give correct centroids for
## longitude/latitude data
```

Note the warning, which basically says that centroid coordinates using longitude/latitude data are not exact, which is perfectly fine for our visualisation purposes.

State names, which are not capitalised in the data from `maps`, can be changed to title case using the function `toTitleCase()` from the package `tools`. This seems trivial – and is if we are just doing data manipulation or calculations – but is critically important when creating visualisations for communication.

```
library("tools")
states$ID <- toTitleCase(states$ID)
head(states)
```

```
## Simple feature collection with 6 features and 3 fields
## geometry type:  MULTIPOLYGON
## dimension:      XY
## bbox:           xmin: -124.3834 ymin: 30.24071 xmax: -71.78015 ymax: 42.04937
## epsg (SRID):    4326
## proj4string:    +proj=longlat +datum=WGS84 +no_defs
##            ID          X        Y                       geometry
## 1     Alabama  -86.83042 32.80316 MULTIPOLYGON (((-87.46201 3...
## 2     Arizona -111.66786 34.30060 MULTIPOLYGON (((-114.6374 3...
## 3    Arkansas  -92.44013 34.90418 MULTIPOLYGON (((-94.05103 3...
## 4  California -119.60154 37.26901 MULTIPOLYGON (((-120.006 42...
## 5    Colorado -105.55251 38.99797 MULTIPOLYGON (((-102.0552 4...
## 6 Connecticut  -72.72598 41.62566 MULTIPOLYGON (((-73.49902 4...
```

Using `geom_text()`, state names can be added. We specify that the columns in the `states` data frame labelled `X` and `Y` determine the placement, as well as the label (from `ID` in the dataframe) and can control the font size.

```
ggplot(data = world) +
    geom_sf() +
    geom_sf(data = states, fill = NA) +
    geom_text(data = states, aes(X, Y, label = ID), size = 5) +
    coord_sf(xlim = c(-88, -78), ylim = c(24.5, 33), expand = FALSE)
```

QUESTION: Why does the Florida label sit in a weird place?

State names can be re-aligned slightly on the y-axis byt creating a new variable `nudge_y`, which is -1 for all states (moved slightly South), 0.5 for Florida (moved slightly North), and -1.5 for South Carolina (moved further South):

```
states$nudge_y <- -1
states$nudge_y[states$ID == "Florida"] <- 0.5
states$nudge_y[states$ID == "South Carolina"] <- -1.5
```

We can also draw a rectangle behind the state name, using the function `geom_label()` instead of `geom_text()`.

```
ggplot(data = world) +
    geom_sf() +
    geom_sf(data = states, fill = NA) +
    geom_label(data = states, aes(X, Y, label = ID), size = 5, fontface = "bold",
        nudge_y = states$nudge_y) +
    coord_sf(xlim = c(-88, -78), ylim = c(24.5, 33), expand = FALSE)
```

Counties (polygon data)

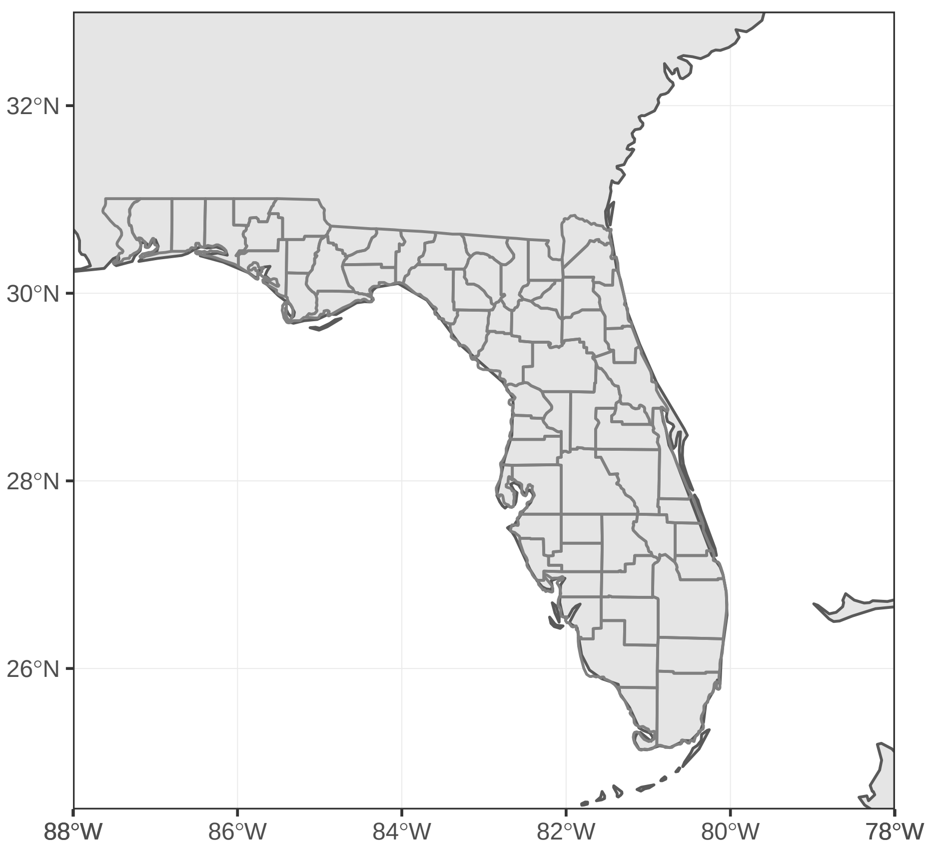
The US is divided smaller than state level into counties. The Australian analogue would be electorates or postcodes rather than cities, because every point in the US falls into a county. We will use these counties to show that areas of polygons can easily be computed, remembering that polygons can come from user-input data to, which won't necessarily have a known area (recall the fires in the lecture). County area can be computed using the `st_area()` function within the `sf` package

```
counties <- st_as_sf(map("county", plot = FALSE, fill = TRUE))
counties <- subset(counties, grepl("florida", counties$ID))
counties$area <- as.numeric(st_area(counties))
head(counties)
```

```
## Simple feature collection with 6 features and 2 fields
## geometry type:  MULTIPOLYGON
## dimension:      XY
## bbox:           xmin: -85.98951 ymin: 25.94926 xmax: -80.08804 ymax: 30.57303
## epsg (SRID):    4326
## proj4string:    +proj=longlat +datum=WGS84 +no_defs
##                         geometry               ID       area
## 292 MULTIPOLYGON (((-82.66062 2...  florida,alachua 2498863359
## 293 MULTIPOLYGON (((-82.04182 3...    florida,baker 1542466064
## 294 MULTIPOLYGON (((-85.40509 3...      florida,bay 1946587533
## 295 MULTIPOLYGON (((-82.4257 29... florida,bradford  818898090
## 296 MULTIPOLYGON (((-80.94747 2...  florida,brevard 2189682999
## 297 MULTIPOLYGON (((-80.89018 2...  florida,broward 3167386973
```

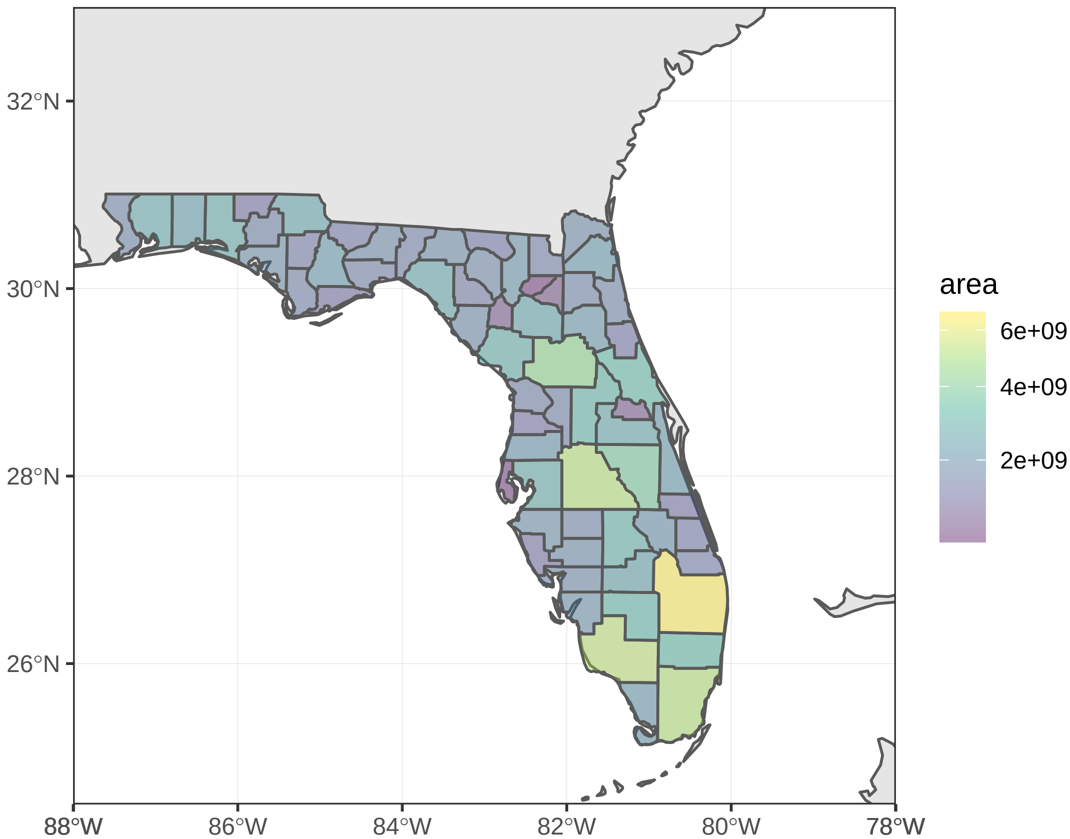
A grey outline of county lines can now be added.

```
ggplot(data = world) +
    geom_sf() +
    geom_sf(data = counties, fill = NA, color = gray(.5)) +
    coord_sf(xlim = c(-88, -78), ylim = c(24.5, 33), expand = FALSE)
```



These polygons can be coloured on any data available at that level. The map below shows area of each region using the "viridis" colorblind-friendly palette, transparency has also been added.

```r
ggplot(data = world) +
    geom_sf() +
    geom_sf(data = counties, aes(fill = area)) +
    scale_fill_viridis_c(trans = "sqrt", alpha = .4) +
    coord_sf(xlim = c(-88, -78), ylim = c(24.5, 33), expand = FALSE)
```



Cities (point data)

Next we will add city names for the five largest cities in Florida. First prepare a data frame with the five largest cities including their geographic coordinates. To save you registering your own google API, we've collected the lat and lng below.

```r
flcities <- data.frame(state = rep("Florida", 5), city = c("Miami",
    "Tampa", "Orlando", "Jacksonville", "Sarasota"), lat = c(25.7616798,
    27.950575, 28.5383355, 30.3321838, 27.3364347), lng = c(-80.1917902,
    -82.4571776, -81.3792365, -81.655651, -82.5306527))
```

Convert the data frame with coordinates to an `sf` object.

```r
flcities <- st_as_sf(flcities, coords = c("lng", "lat"), remove = FALSE,
    crs = 4326, agr = "constant")
```

TASK: explore the new `flcities` dataset.

TASK: Add city locations and names to the map.

The package `ggrepel` can adjust the label placement (`geom_text_repel()` and `geom_label_repel()`). It will automatically correct any overlap, "nudge" the labels away from land into the see, and connect them to the city locations.

```r
ggplot(data = world) +
    geom_sf() +
    geom_sf(data = counties, fill = NA, color = gray(.5)) +
    geom_text_repel(data = flcities, aes(x = lng, y = lat, label = city),
        fontface = "bold", nudge_x = c(1, -1.5, 2, 2, -1), nudge_y = c(0.25,
            -0.25, 0.5, 0.5, -0.5)) +
    coord_sf(xlim = c(-88, -78), ylim = c(24.5, 33), expand = FALSE)
```

Final map

Add or adjust the theme, titles, subtitles, axis labels, and add a scale bar:

```r
ggplot(data = world) +
    geom_sf(fill = "antiquewhite1") +
    geom_sf(data = counties, aes(fill = area)) +
    geom_sf(data = states, fill = NA) +
    geom_sf(data = sites, size = 4, shape = 23, fill = "darkred") +
    geom_text_repel(data = flcities, aes(x = lng, y = lat, label = city),
        fontface = "bold", nudge_x = c(1, -1.5, 2, 2, -1), nudge_y = c(0.25,
            -0.25, 0.5, 0.5, -0.5)) +
    geom_label(data = states, aes(X, Y, label = ID), size = 5, fontface = "bold",
        nudge_y = states$nudge_y) +
    scale_fill_viridis_c(trans = "sqrt", alpha = .4) +
    annotation_scale(location = "bl", width_hint = 0.4) +
    annotation_north_arrow(location = "bl", which_north = "true",
        pad_x = unit(0.75, "in"), pad_y = unit(0.5, "in"),
        style = north_arrow_fancy_orienteering) +
    coord_sf(xlim = c(-88, -78), ylim = c(24.5, 33), expand = FALSE) +
    xlab("Longitude") + ylab("Latitude") +
    ggtitle("Observation Sites", subtitle = "(2 sites in Palm Beach County, Florida)") +
    theme(panel.grid.major = element_line(color = gray(0.5), linetype = "dashed",
        size = 0.5), panel.background = element_rect(fill = "aliceblue"))
```