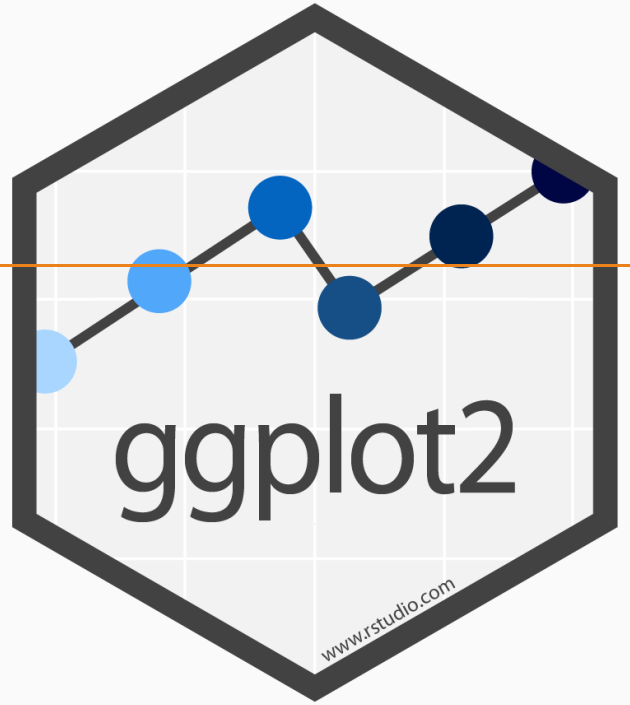
Data Visualisation in R

Introduction to ggplot2 and leaflet

Dr. Laurie Baker
Data Science Campus
2020/04/09 (updated: 2020-05-07)



Introduction to leaflet

What we'll cover today.

- Quick recap on building a plot using `ggplot2`
- Intro to the `sf` packages
- Building a map using `ggplot2`
- Coordinate Systems and geospatial objects in R
- Building interactives maps using `leaflet`.

Packages for today's adventure

```
library(tidyverse) ## For plotting and data wrangling.

library(leaflet) ## For leaflet interactive maps

library(sf) ## For spatial data

library(RColorBrewer) ## For colour palettes

library(htmltools) ## For html

library(leafsync) ## For placing plots side by side
```

Recap on building a plot using ggplot2

ggplot2 plots are built in layers

- **Data** must be in a "tidy" format.
- **Aesthetic mappings** link variables in the data to graphical properties in the **geom**etric objects.
- **Geometric objects** dictate how the **aesthetics** are interpreted as a graphical representation (points, lines, polygons, etc.)
- **Statistics** transform the input variables to displayed values. E.g. calculate the summary statistics for a boxplot (quantiles).
- **Coordinates** organize location of geometric objects, i.e. define the physical mapping of the aesthetics.

ggplot2 plots are built in layers

- **Scales** define the range of values for aesthetics (e.g. categories -> colours).
- **Facets** define the number of panels and how to split data among them (e.g. by country).
- **Themes** control every part of the graphic that is not linked to the data (i.e. font, visual appearance).

Making a map

- `sf` is a package for geospatial data manipulation and analysis.
- It works with features:
  - **points** (POINT, MULTIPOINT)
  - **lines** (LINESTRING, MULTILINESTRING)
  - **polygons** (POLYGON, MULTIPOLYGON)

Plotting a map with ggplot2

Birth Rates in North Carolina

- We will use the North Carolina (`nc`) data from the `sf` package.
- Let's load in the data for North Carolina using the function `st_read`.

```
nc_df ← st_read(system.file("shape/nc.shp", package="sf"))
```

```
## Reading layer `nc' from data source `C:\Users\l-baker\Documents\R\win-library\3.6\s
## Simple feature collection with 100 features and 14 fields
## geometry type:  MULTIPOLYGON
## dimension:      XY
## bbox:           xmin: -84.32385 ymin: 33.88199 xmax: -75.45698 ymax: 36.58965
## epsg (SRID):    4267
## proj4string:    +proj=longlat +datum=NAD27 +no_defs
```

- `st` = spatial type and `.shp` is a common shape file format (e.g. GIS).

Rename the columns and view the data North

- Number of births for counties in North Carolina in 1974
- Rename our columns to country, births, and geometry.

```
nc ← nc_df %>%
        select("NAME", "BIR74", "BIR79", "geometry") %>%
        rename("county" = "NAME", "births1974" = "BIR74", "births1979" = "BIR79")
```

Our first map

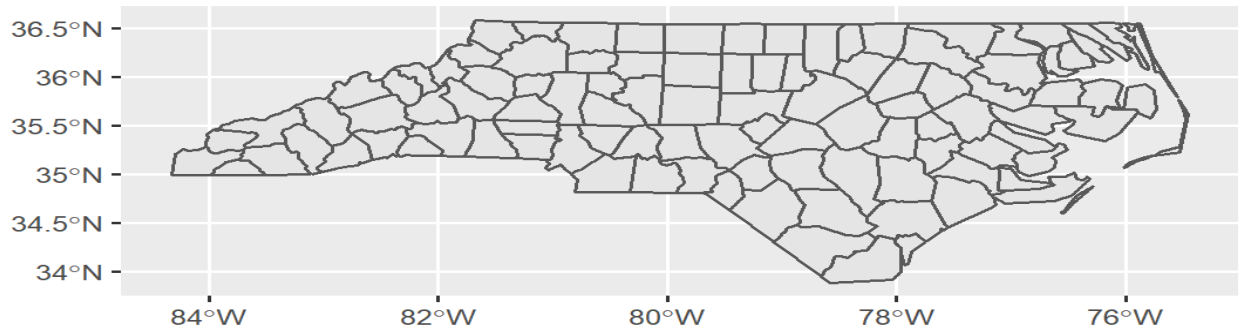
```
ggplot(nc)
```

- Data
- Geom
- Aesthetics
- Labels
- Scales

Our first map

```
ggplot(nc) +
  geom_sf()
```

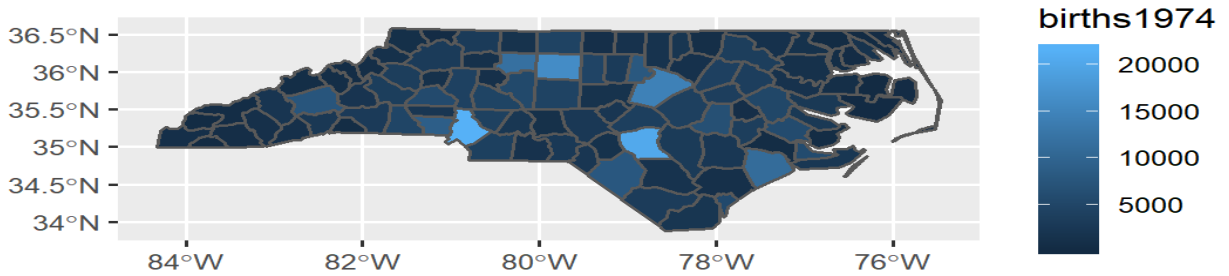
- Data
- Geom
- Aesthetics
- Labels
- Scales



Our first map

```
ggplot(nc) +
  geom_sf(aes(fill = births197
```

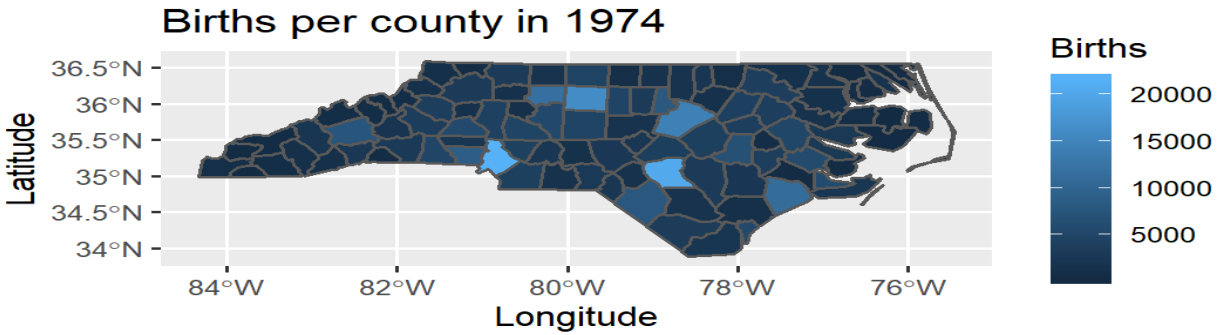
- Data
- Geom
- Aesthetics
- Labels
- Scales



Our first map

```
ggplot(nc) +
  geom_sf(aes(fill = births197
  labs(title = "Births per cou
       x = "Longitude",
       y = "Latitude",
       fill = "Births")
```

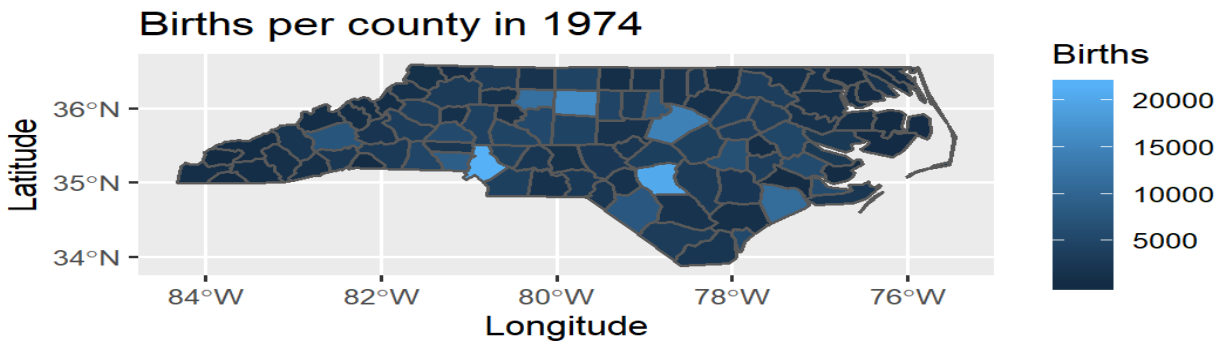
- Data
- Geom
- Aesthetics
- Labels
- Scales



Our first map

```
ggplot(nc) +
  geom_sf(aes(fill = births197
  labs(title = "Births per cou
       x = "Longitude",
       y = "Latitude",
       fill = "Births") +
  scale_y_continuous(breaks =
```

- Data
- Geom
- Aesthetics
- Labels
- Scales



Coordinate reference system

- Every location on earth is specified by a longitude and latitude.
- The Coordinate Reference system (CRS) determines how the data will be projected onto a map.
  *

Coordinate reference system

- We can check the CRS using `st_crs`:

```
st_crs(nc)
```

```
## Coordinate Reference System:
##   EPSG: 4267
##   proj4string: "+proj=longlat +datum=NAD27 +no_defs"
```

- The CRS is specified in the attributes `epsg` and `proj4string`.
- What is the epsg?
- What is the proj4string?

Transforming coordinate reference system

- You can transform a coordinate reference system using the `st_transform()`.
- But what is a sensible coordinate reference system to assign?
- Well, a good place to start is with the one that leaflet uses for plotting the world: EPSG 4326.

```
nc ← st_transform(nc, "+init=epsg:4326")

st_crs(nc)
```

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

Our first leaflet map

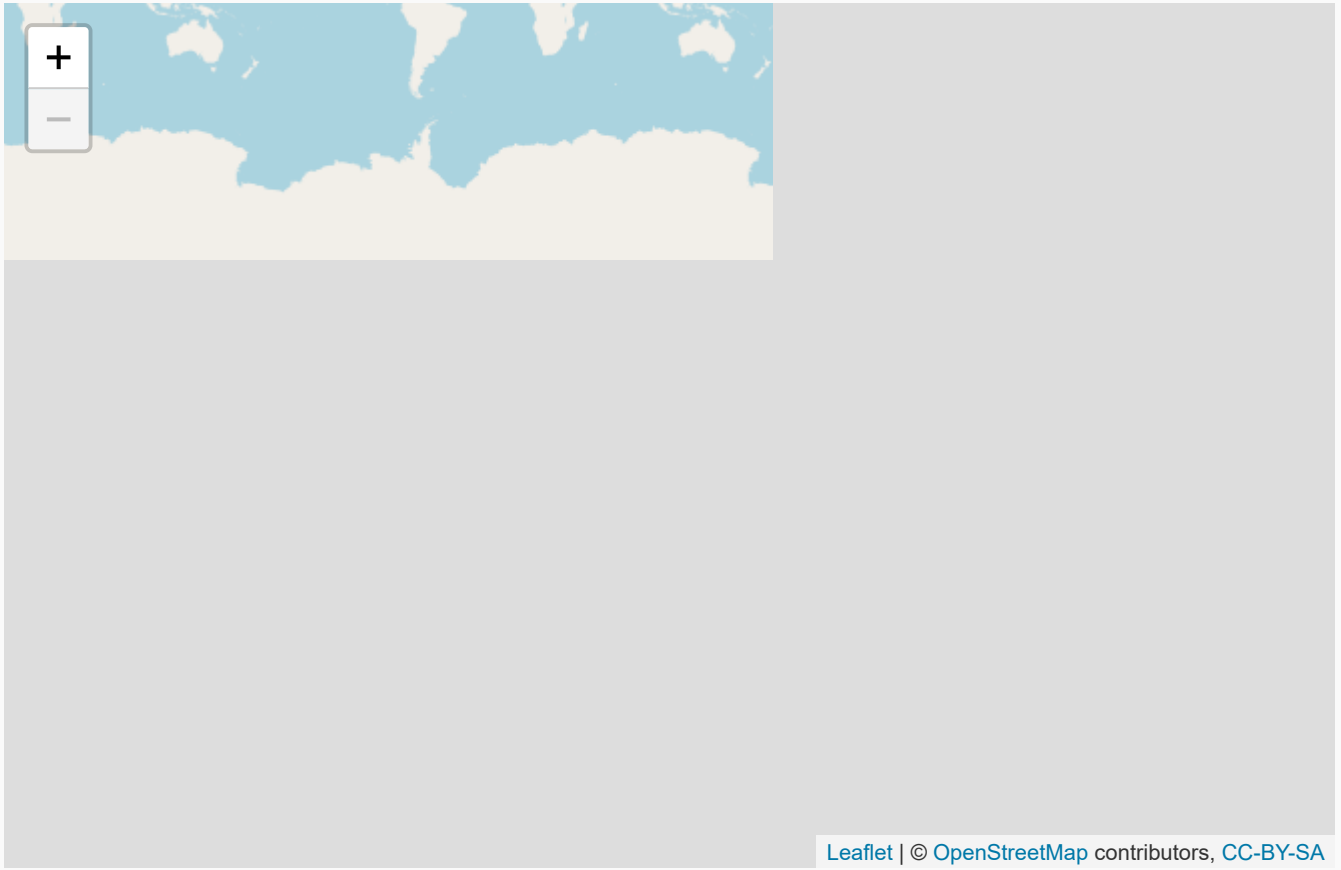
- Every plot starts with `leaflet()`

```
leaflet(data = nc)
```

Our first leaflet map

- Layers are added using `%>%`

```
leaflet(data = nc) %>%
  addTiles()
```

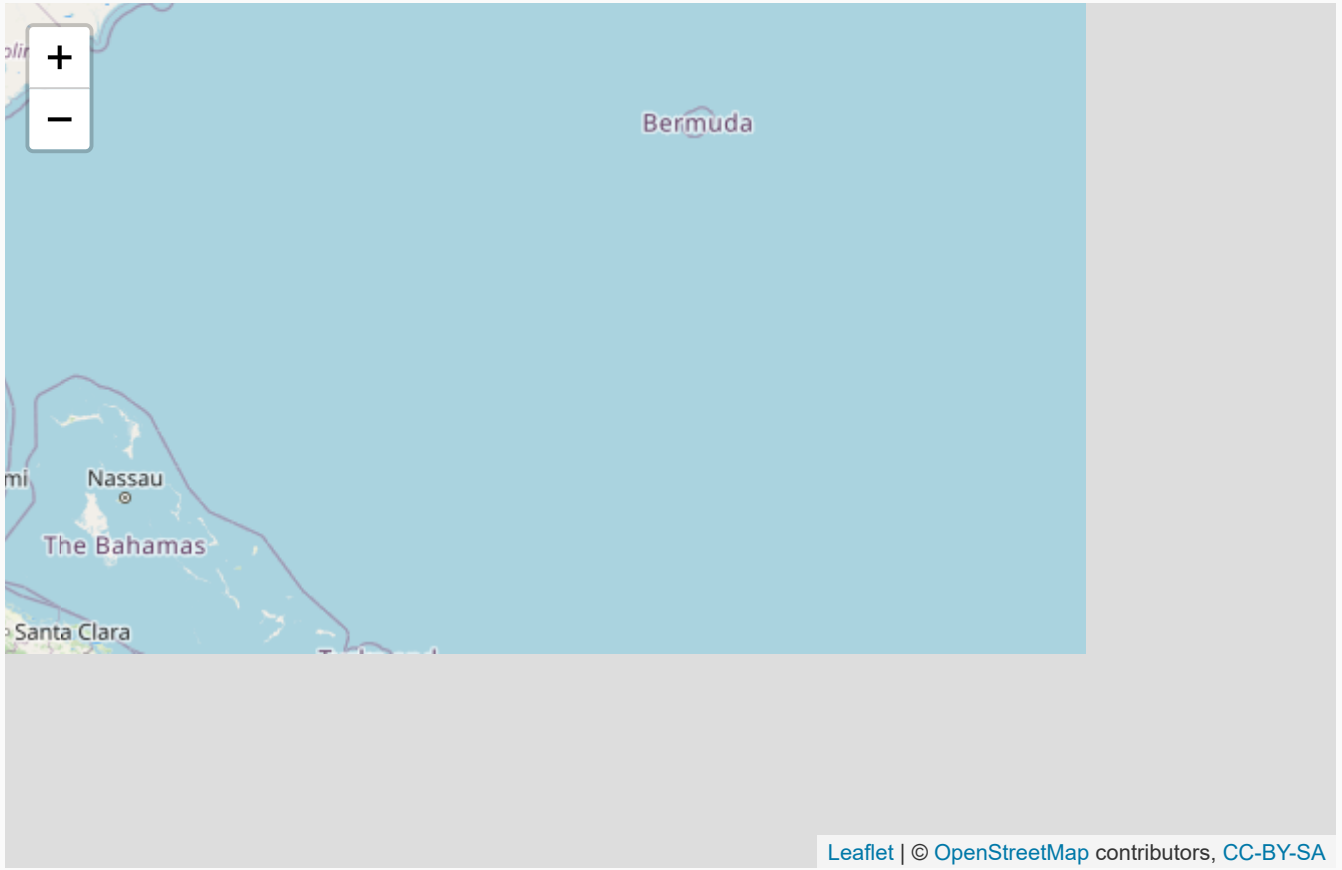


N.B. Layers are added with `%>%` in `leaflet` and `+` in `ggplot`

Our first leaflet map

- We can set the view using `setView()`

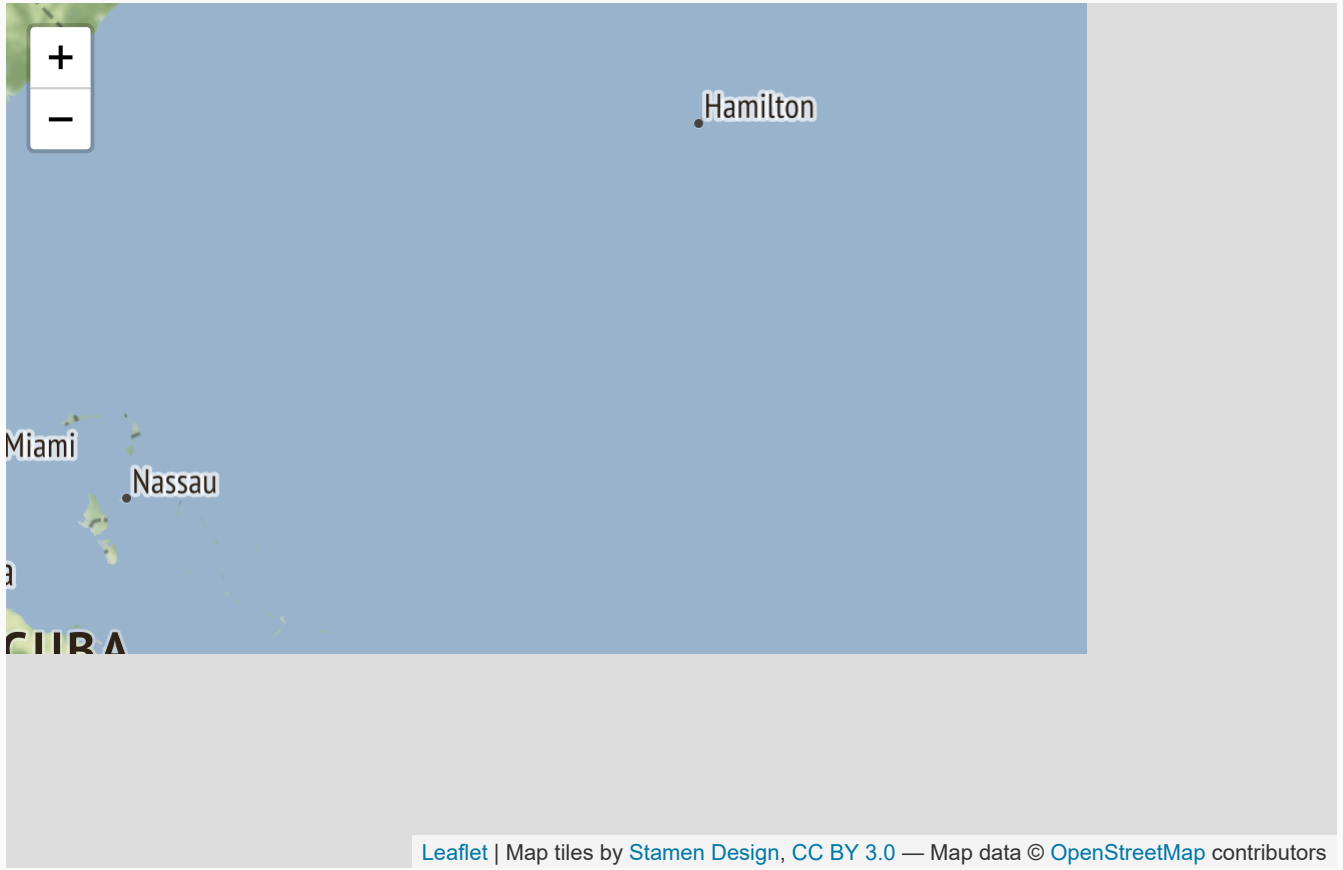
```
leaflet(data = nc) %>%
  addTiles() %>%
  setView(lng = -80,
          lat = 34.5,
          zoom = 5)
```



Our first leaflet map

- Add different background map using addProviderTiles

```
leaflet(data = nc) %>%
  addProviderTiles(providers$St
  setView(lng = -80,
          lat = 34.5,
          zoom = 5)
```



Our first leaflet map

- Add polygons using `addPolygons()`

```
leaflet(data = nc) %>%
  addProviderTiles(providers$St
  setView(lng = -80,
          lat = 34.5,
          zoom = 5) %>%
  addPolygons()
```

Creating a colour palette

- `RColorBrewer` includes **sequential** colour palettes (e.g. number of people).

```
display.brewer.all(type = "seq"
```

Creating a colour palette

- `RColorBrewer` includes **diverging** colour palettes (e.g. to show distinct categories).

```
display.brewer.all(type = "div"
```

Creating a colour palette

- First we will define the colour palette and bins for the plot.

```
head(nc$births1974); head(nc$births1979)
```

```
## [1] 1091  487 3188  508 1421 1452
```

```
## [1] 1364  542 3616  830 1606 1838
```

```
bins ← c(seq(0, 35000, 5000))
```

- Then we can define the colours for the palette:

```
pal74 ← colorBin("OrRd", domain = nc$births1974, bins = bins)

pal79 ← colorBin("OrRd", domain = nc$births1979, bins = bins)
```

Creating a colour palette

- Customising `addPolygons()`

```
leaflet(data = nc) %>%
  addProviderTiles(providers$St
  setView(lng = -80,
          lat = 34.5,
          zoom = 6) %>%
  addPolygons(data = nc,
    fillColor = ~pal74(nc$birth
    fillOpacity = 0.7,
    color = "white",
    opacity = 1,
    weight = 2
  )
```

Creating a colour palette

- Customising `addPolygons()`

```
leaflet(data = nc) %>%
  addProviderTiles(providers$St
  setView(lng = -80,
          lat = 34.5,
          zoom = 6) %>%
  addPolygons(data = nc,
    fillColor = ~pal74(nc$birth
    fillOpacity = 1,
    color = "blue",
    opacity = 0.7,
    weight = 1
  )
```

What can you customise in addPolygons()

```
?addPolygons()
```

- `color:` stroke color
- `weight:` stroke width in pixels
- `opacity:` stroke opacity
- `fillColor:` fill color

- `fillOpacity:` fill opacity

- `highlightOptions:` Options for highlighting the shape on mouse over.

What can you customise in addPolygons()

- Let's assign our plot to an object.

```
m1 ← leaflet(data = nc) %>%
  addProviderTiles(providers$St
  setView(lng = -80,
          lat = 34.5,
          zoom = 6)

m1 %>%
  addPolygons(data = nc,
    fillColor = ~pal74(nc$birth
    fillOpacity = 0.7,
    opacity = 1,
    color = "white",
    weight = 2)
```

What can you customise in addPolygons()

- Let's add some `highlightOptions`

```
m1 %>%
  addPolygons(data = nc,
      fillColor = ~pal74(nc$bi
      fillOpacity = 0.7,
      color = "white",
      opacity = 1,
      weight = 2,
    highlight = highlightOptio
        weight = 3,
        color = "blue",
        fillOpacity = 1,
        bringToFront = TRUE))
```

Let's add some labels!

`sprintf`: returns a character vector containing a formatted combination of text and variable values.

```
labels ← sprintf(
  "<strong>%s</strong><br/>%g births",
  nc$county, nc$births1974
) %>% lapply(htmltools::HTML)

head(labels)
```

```
## [[1]]
## <strong>Ashe</strong><br/>1091 births
##
## [[2]]
## <strong>Alleghany</strong><br/>487 births
##
## [[3]]
## <strong>Surry</strong><br/>3188 births
```

Let's add some labels!

- Let's add some labels

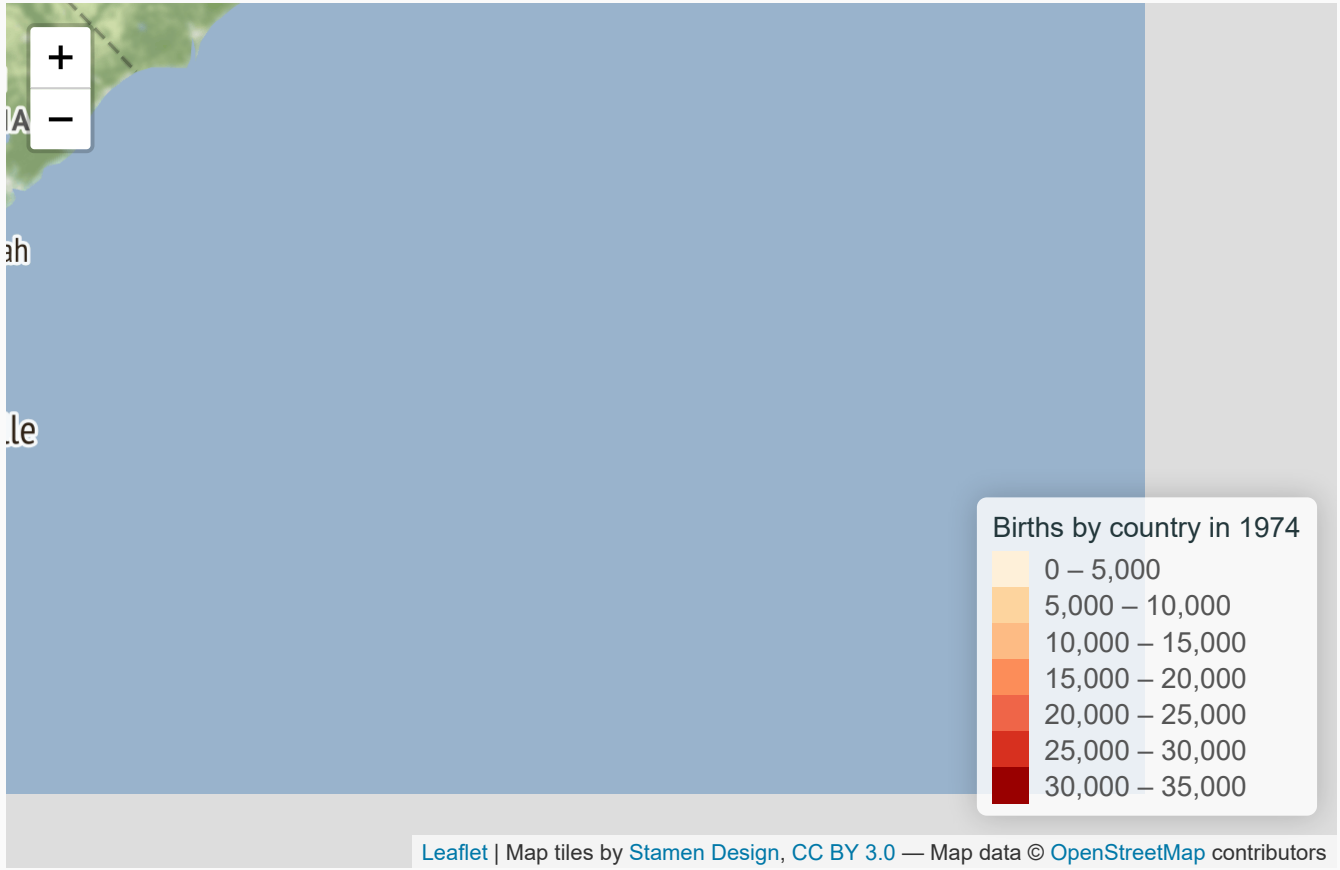
```
m1 ← m1 %>%
  addPolygons(data = nc,
      fillColor = ~pal74(nc$bi
      fillOpacity = 0.7,
      color = "white",
      opacity = 1,
      weight = 2,
    highlight = highlightOptio
        weight = 3,
        color = "blue",
        fillOpacity = 1,
        bringToFront = TRUE),
    label = labels)
```

Let's add some labels!

- Let's add a legend

```
m1 ← m1 %>%
  addLegend(
    position = "bottomright",
    pal = pal74,
    values = ~nc$births1974,
    title = "Births by country
    opacity = 1)

m1
```



Let's create a second map

- Let's create a second map of births in 1979.
- First we'll need to create a new set of labels

```
labels79 ← sprintf(
  "<strong>%s</strong><br />%g births",
  nc$county, nc$births1979
) %>% lapply(htmltools::HTML)
```

Let's create a second map

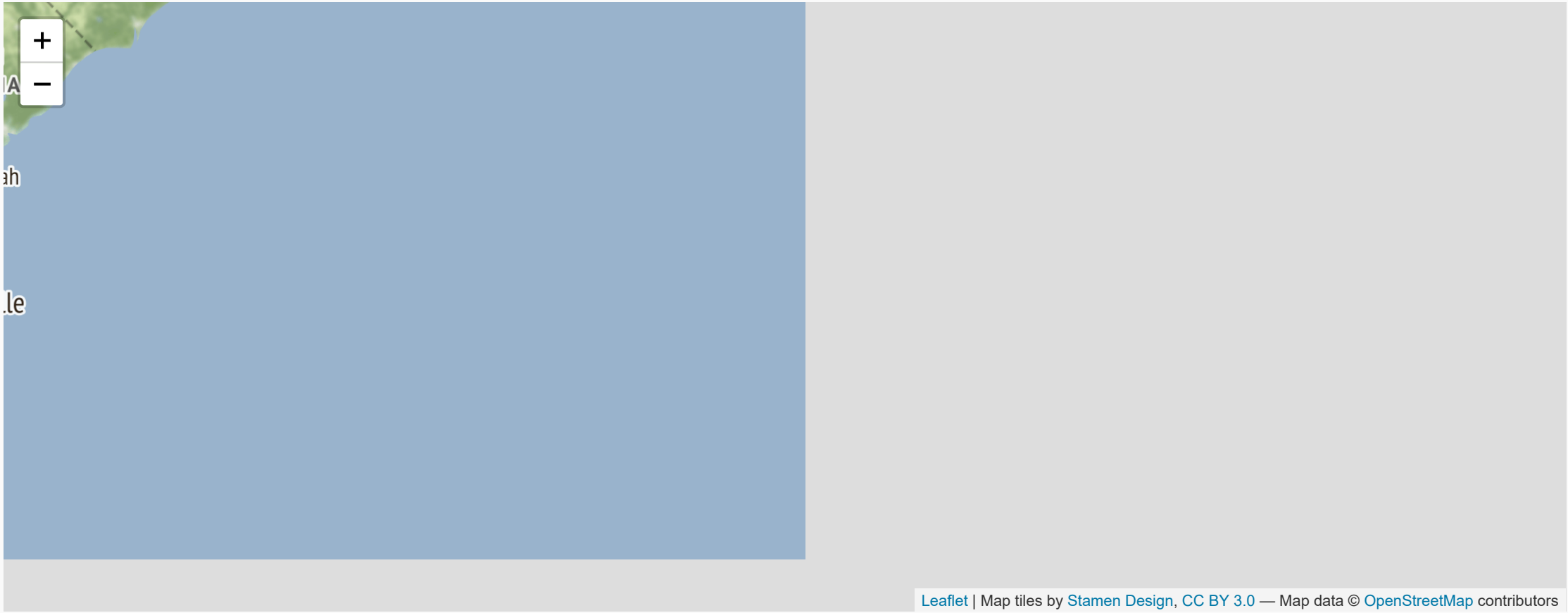
```
m2 ← leaflet(data = nc) %>%
  addProviderTiles(providers$Stamen.Terrain) %>%
  setView(lng = -80,
          lat = 34.5,
          zoom = 6) %>%
  addPolygons(data = nc,
      fillColor = ~pal79(nc$births1979),
      fillOpacity = 0.7,
      color = "white",
      opacity = 1,
      weight = 2,
    highlight = highlightOptions(
        weight = 3,
        color = "blue",
        fillOpacity = 1,
        bringToFront = TRUE),
    label = labels79)
```

Let's create a second map

```
m2 ← m2 %>%
  addLegend(
    position = "bottomright",
    pal = pal79,
    values = ~nc$births1979,
    title = "Births by country in 1979",
    opacity = 1)
```

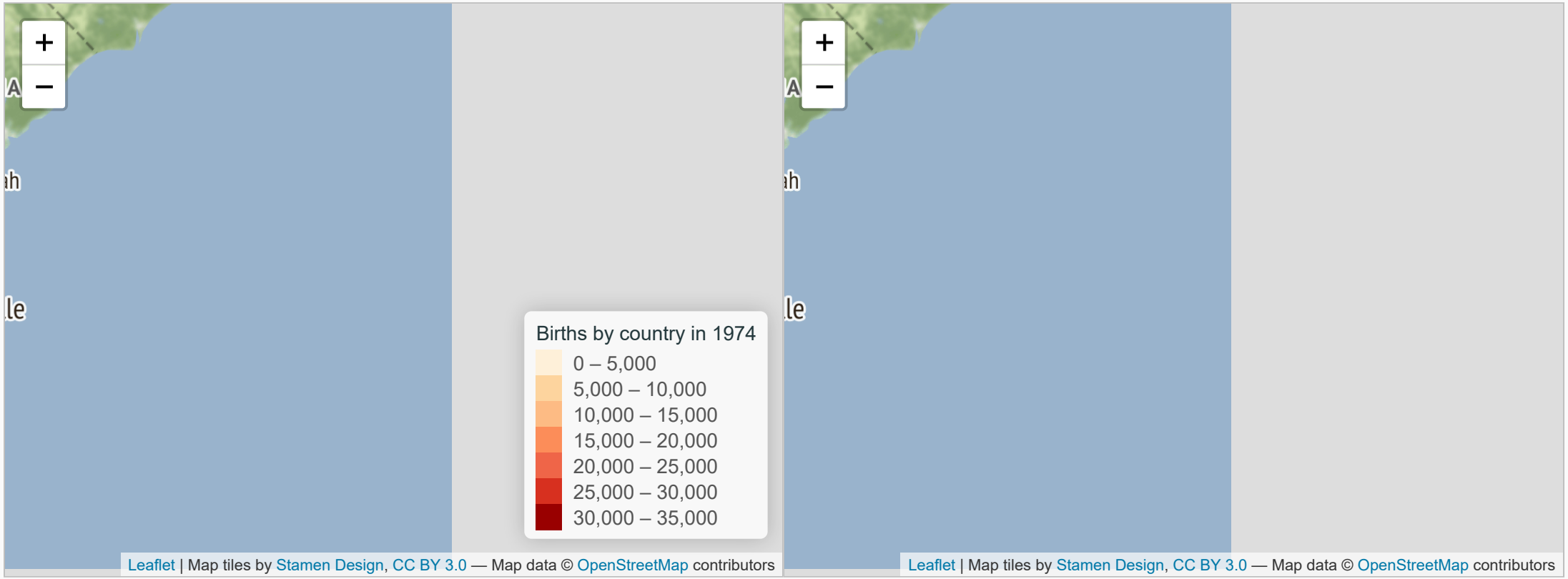
Let's create a second map

```
m2
```



Placing two maps side by side

```
leafsync::sync(m1, m2, ncol = 2, sync = "all")
```



Placing two maps side by side

Placing two plots side by side

Slides created via the R package .

The chakra comes from remark.js, , and R Markdown.

- Slides template adapted from Garrick Aden-Buie GitHub: http://github.com/gadenbuie/gentle-ggplot2