# HDAT9800

# Visualisation and Communication of Health Data

## Chapter 5

*Drs James Farrow & Tim Churches*

# Leaflet

Open source package for interactive maps

R package wraps *leaflet* features

Embeddable in Rmarkdown, *knitr* documents and *shiny* apps

```
install.packages("leaflet")
```

# Using leaflet

Using *leaflet*

- create a map
- add layers
- display the map

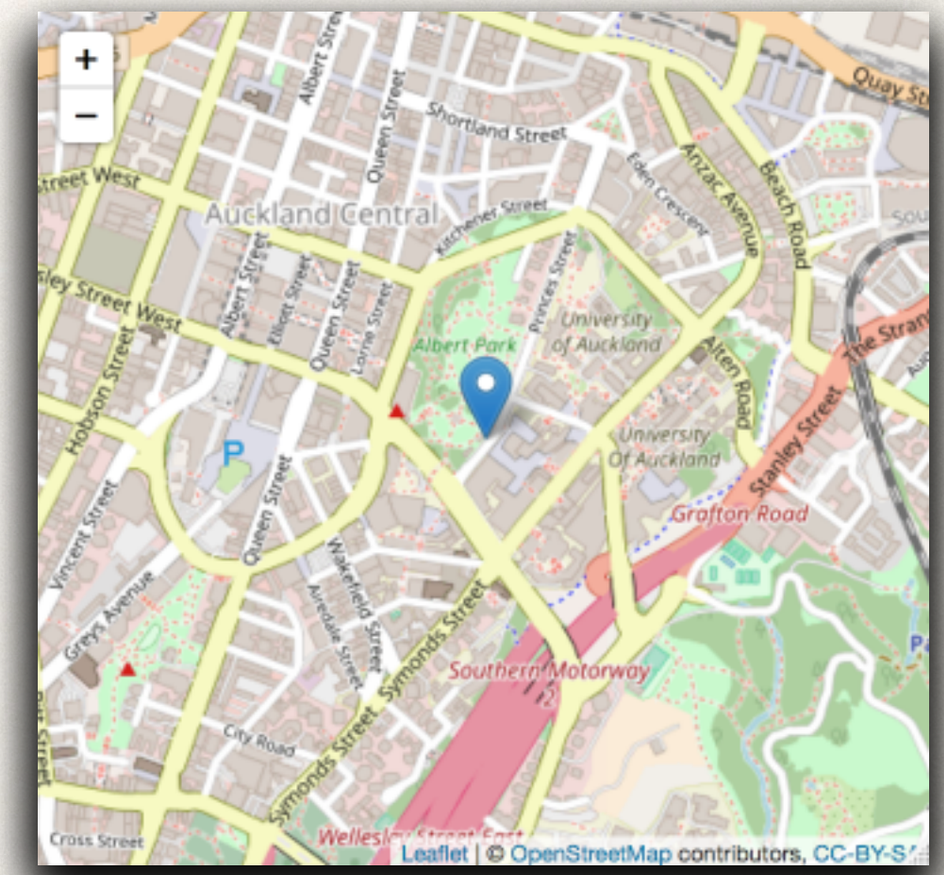
Compare with *ggplot2*

# Using leaflet

```
m <- leaflet()

m <- addTiles(m)

m <- addMarkers(m, lng=174.768,
      lat=-36.852,
      popup="The birthplace of R")
```



# Pipelines

*leaflet* functions take the map as the first argument

This makes them ideal for use with `%>%`

```
m <- leaflet() %>%
     addTiles() %>%
     addMarkers(lng=174.768, lat=-36.852,
                popup="The birthplace of R")
```

# The Map Widget

*leaflet()* creates a map widget object

- argument *options* takes a *leafletOptions(…)* object

Methods to manipulate the map

- *setView()*
- *fitBounds(), clearBounds()*

# Passing data to *leaflet*

*leaflet()* and the layer functions take a *data* argument

- lat/long matrix
- data frame with lat/lng columns

From *sp* package

- SpatialPoints, Line(s), SpatialLines, Polygon(s), SpatialPolygons

From the *maps* package

- the data frame returned from *map()*

# A word on coordinate systems

Latitude and longitude are coordinates on an *oblate spheroid* (the *geoid*)

They are not cartesian x/y coordinates

You cannot use Pythagoras's Theorem to calculate distance

1° of longitude/latitude varies over the surface of the Earth

| Location | 1° longitude | 1° latitude |
|---|---|---|
| Equator | 111.319 km | 110.574 km |
| Tropic of Cancer/Capricorn | 102.141km | 110.751 km |
| 60° N/S | 55.800km | 111.412.km |
| Poles | 0 km | 111.694 km |

# Australia

1° of longitude

- 106.53 km near Cairns
- 98.22 km near Tweed Heads
- 92.54 km near Sydney
- 89.41 km near Bega
- 88.07 km near Melbourne
- 81.70 km near Hobart

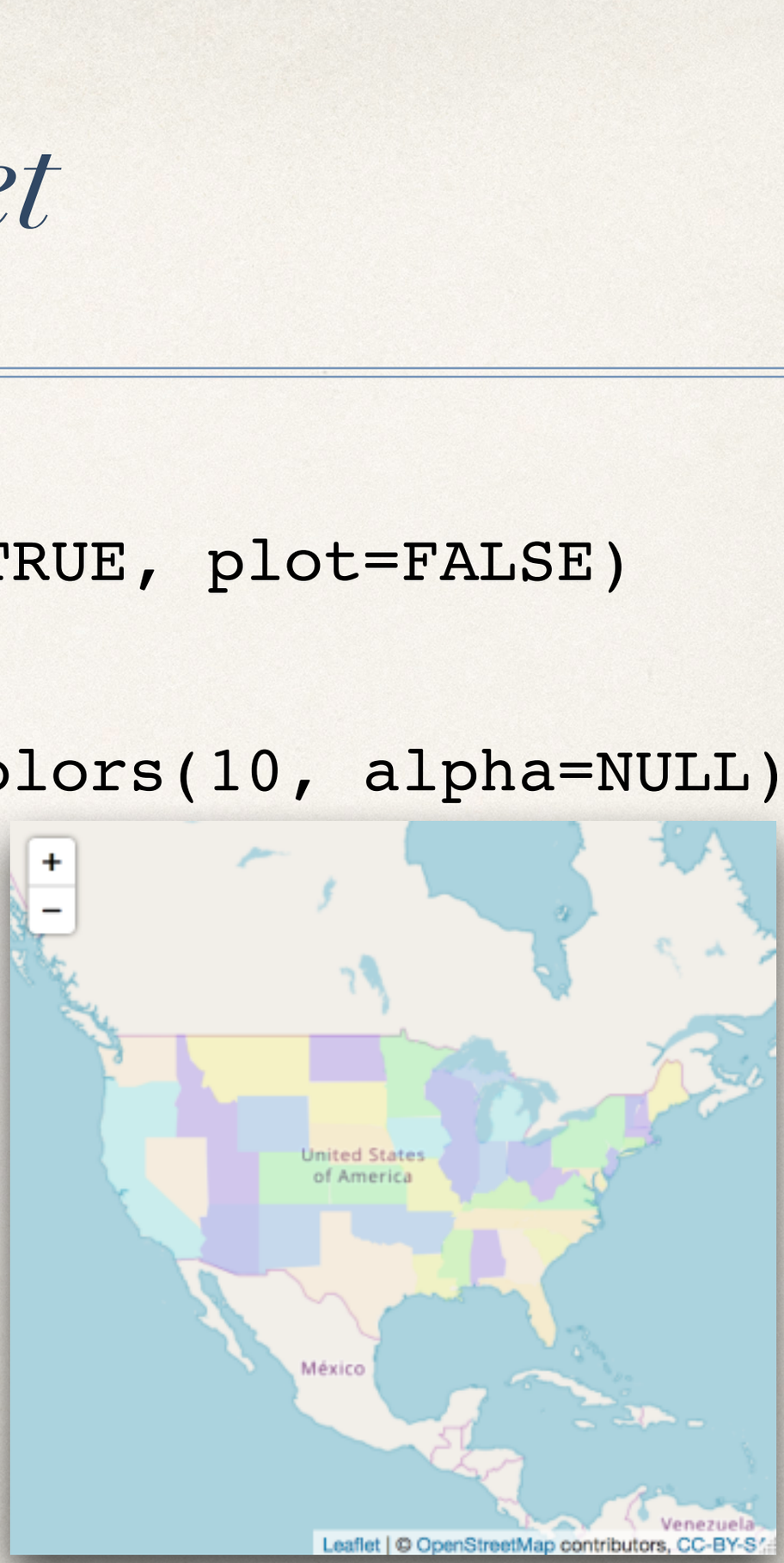
# Best fit

Different countries' mapping services use slightly different reference ellipsoids and coordinate mappings to approximate the geoid to make their maps 'fit' as best as possible. The Prime Meridian of WGS84 (World Geodetic System) does not pass through Greenwich Observatory.

# Passing data to *leaflet*

```
library(maps)
mapStates = map("state", fill=TRUE, plot=FALSE)
leaflet(data = mapStates) %>%
  addTiles() %>%
  addPolygons(fillColor=topo.colors(10, alpha=NULL),
              stroke = FALSE)
```

You can use *RColorBrewer* for palettes



# Formula options

Arguments to layer functions take R object (like vectors)

They can also take a one-sided formula

The formula is evaluation with the `data` argument as the context

So `~x` means ‘variable/column *x* in the data object’

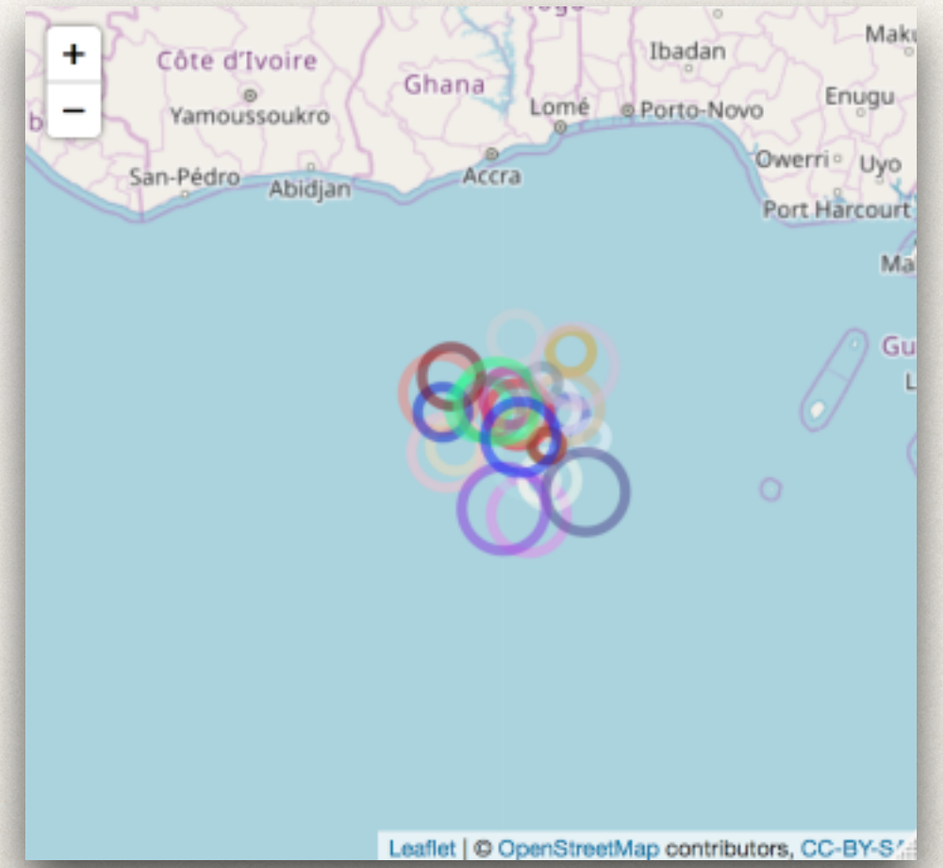
# Formula options

```
df = data.frame(
  lat = rnorm(100), lng = rnorm(100),
  size = runif(100, 5, 20),
  color = sample(colors(), 100)
)
head(df)
```

```
         lat          lng      size      color
1 -0.9533640  0.04677162  7.829027 olivedrab4
2 -1.2614975 -1.65752026 10.641469 violetred1
3  0.7709019  0.47280072 11.415945 burlywood4
4  0.1280601  1.40475562  8.246746  slateblue
5  0.7759174 -0.48755981 19.797309    maroon3
6 -1.9703236  0.58662615 18.182785     violet
```

# Formula options

```
m <- leaflet(df) %>% addTiles()
m %>% addCircleMarkers(
        radius = ~size,
        color = ~color,
        fill = FALSE)
```



# Base maps

*addTiles()* with no arguments uses OpenStreetMap tiles

- *https://www.openstreetmap.org/*

*addProviderTiles()* can be used to add third-party tiles

- *https://github.com/leaflet-extras/leaflet-providers*

Multiple tile layers can be added

- adjust layer alpha (transparency) so they can be seen

# Markers

Markers identify points-of-interest on a map

Icon markers are added using *addMarkers()* or *addAwesomeMarkers()*

Custom marker graphics can be provided

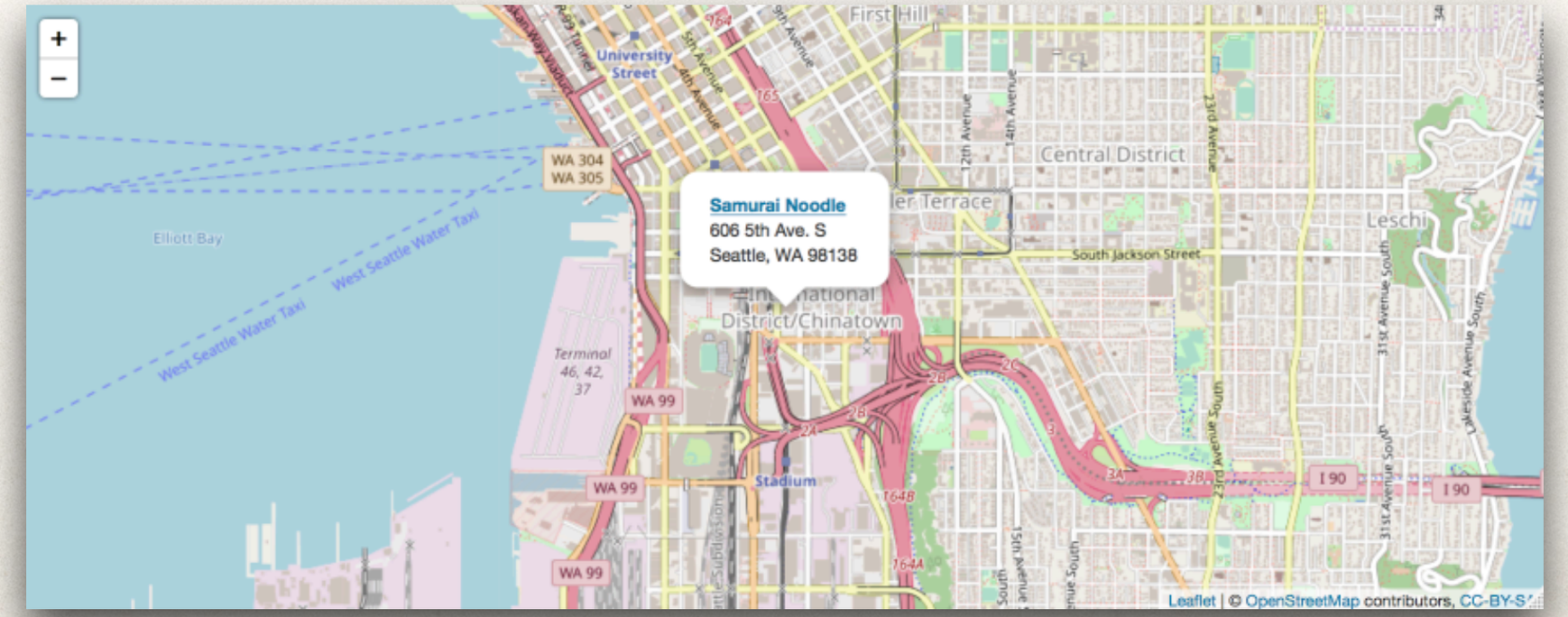
Markers can be clustered when zoomed-out to reduce clutter

# Labels and popups

Labels appear when the user hovers over a marker or shape

Popups appear when the user clicks on a marker or shape

Size, style, shape and colour can be customised



# Lines and regions

Lines and polygons can be added as layers

There are necessary to show features such as

- roads, boundaries
- mesh blocks, census districts, suburbs, postcodes, states, area health services

Various packages, *e.g. sp* and *map*, provide tools for this kind of spatial data

Be aware that boundary polygons can be very complex and contain unnecessary detail when zoomed out. Consider simplifying boundaries.

# Shiny integration

We will start looking at *shiny* next week

This week you can skip the *Shiny Integration* section of the practical

- unless you're feeling particularly adventurous

We will be using *shiny* and *leaflet* in the group assignment

# Colours

We have discussed colour before

A great extra axis for data presentation

- can be overused
- has caveats (colour-blindness, cultural aspects)
- needs to be well thought out

Colours are chosen from colour palettes

Built-in palettes and *RColorBrewer* can be used

# Colours

Smooth gradients or disjoint/clashing colours?

Scalar/continuous data or categorical/factor data?

Questions such as these inform our colour choices

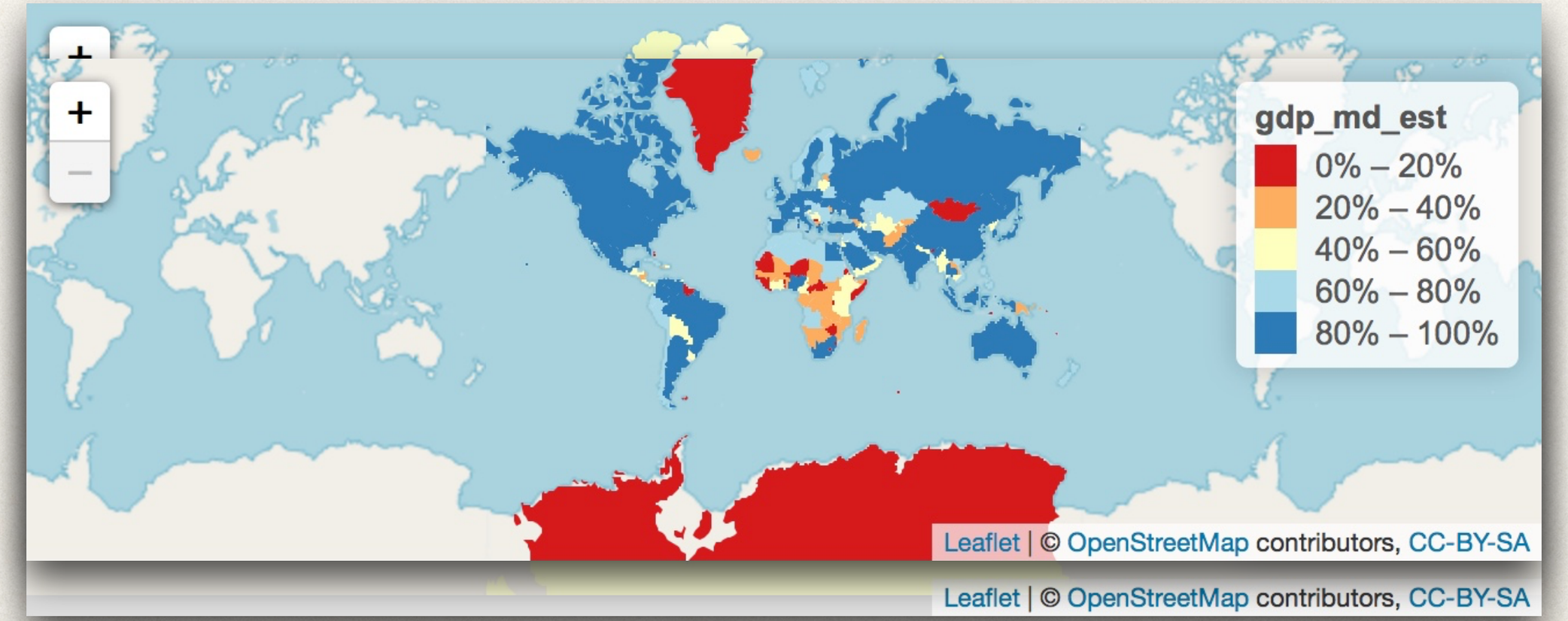
Some resources

- *https://morphocode.com/the-use-of-color-in-maps/*
- *https://tilemill-project.github.io/tilemill/docs/guides/tips-for-color/*

# Legends

As with plots, maps need legends

*addLegend()* will add a legend layer

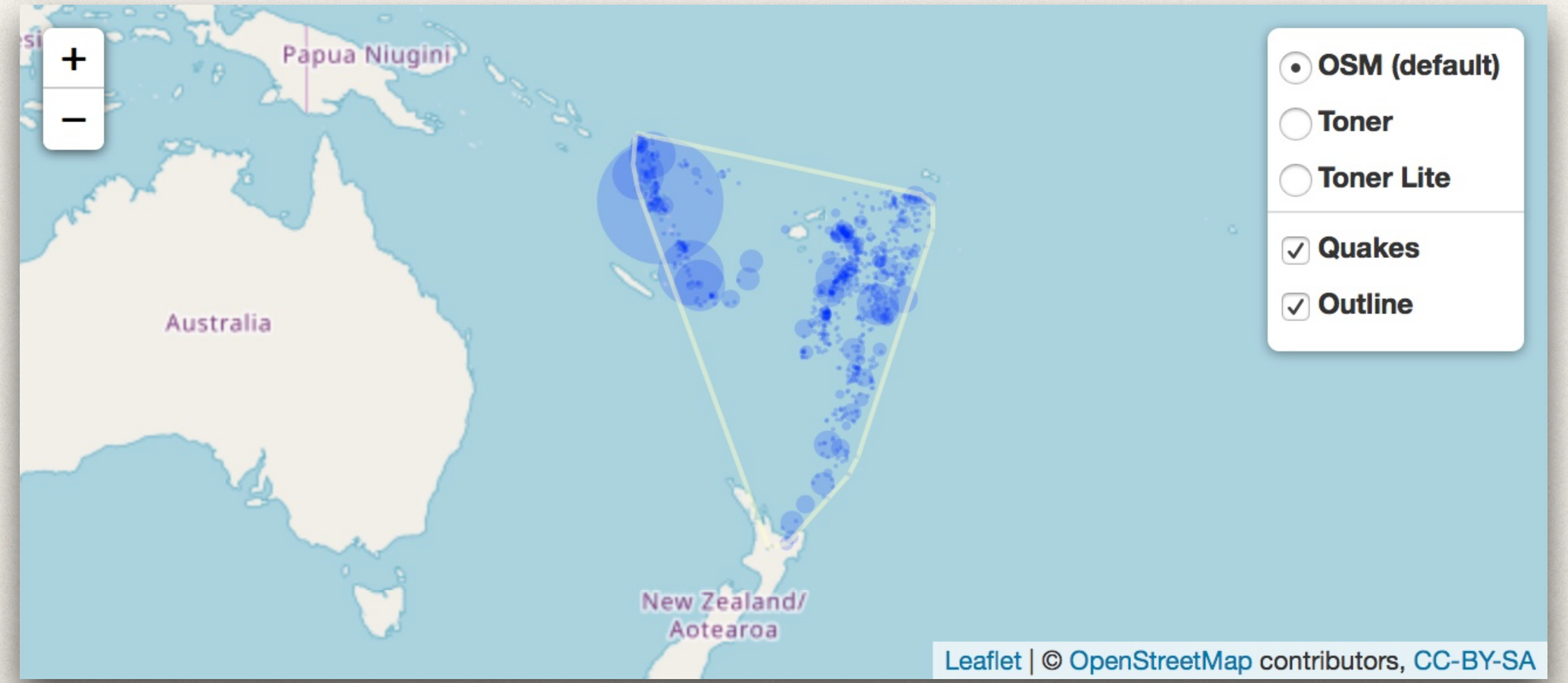


Leaflet | © OpenStreetMap contributors, CC-BY-SA

# Interactivity

Layers can be turned on and off

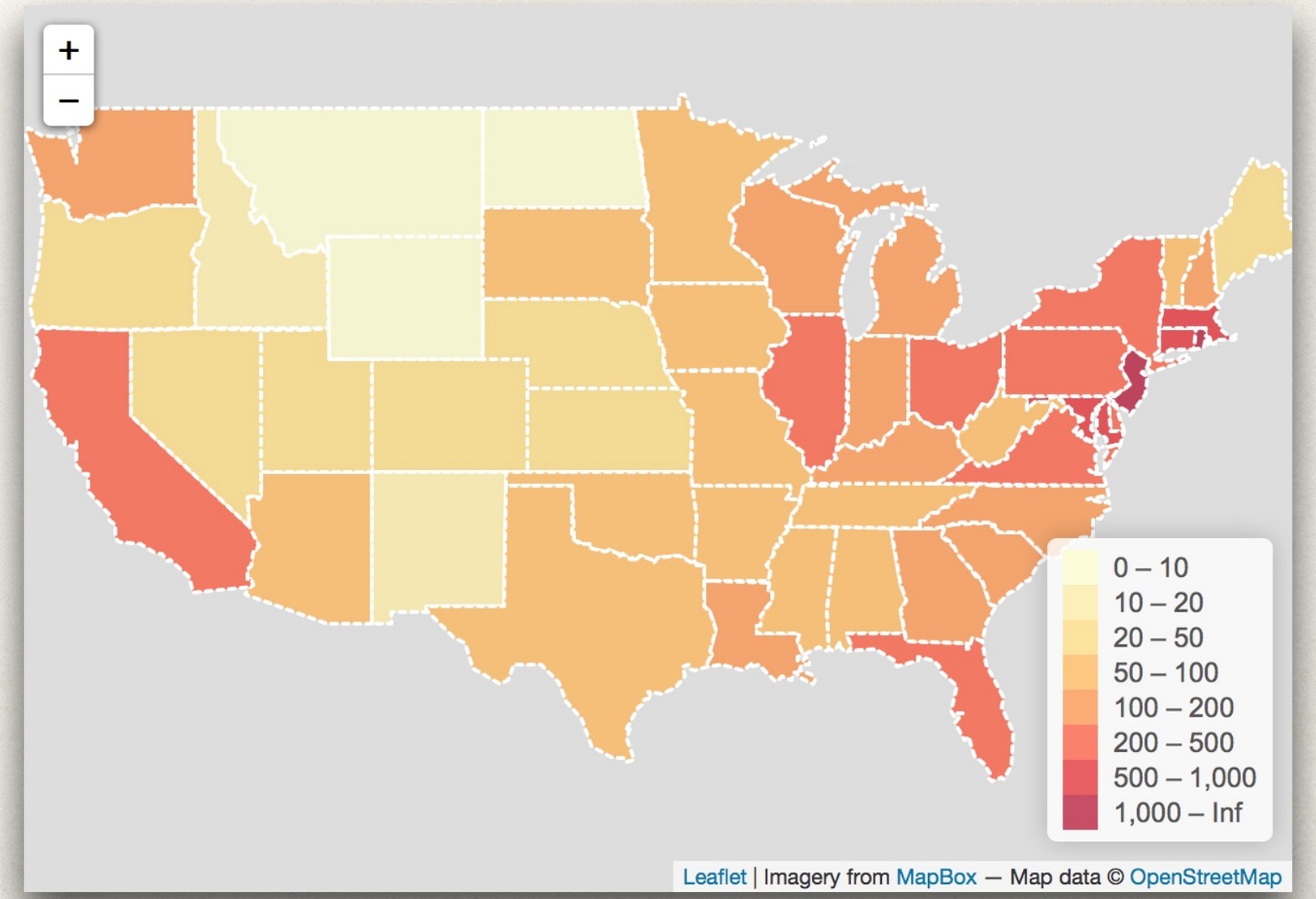
The user can control the visibility of features of interest



# Choropleths

A *choropleth* shows averages of a variable using regions or symbols

Compare with *isopleth*



# Projections

Projections detail how to take coordinates and project them onto the (usually cartesian) display plane

Raster tile maps are designed to work with a particular projection

Custom projections can be used but may need custom tile maps

# This week's practical

Build a simple interactive map displaying some static data using *leaflet*

# Next week

We start to look at *shiny* and how to create interactive applications

This is building towards one of the optional topics for your group assignments, which is an interactive geo-spatial analysis app using *shiny* and *leaflet* and other things you've learnt this semester.

It will be a group project using *git* to coordinate development

In 2022, alternative group assignment topics will also be offered.