Introduction to GIS & Spatial Analysis in R

Marie-Hélène Burle
training@westgrid.ca

April 26, 2021

GIS concepts

Grasshopper Geography

Types of spatial data

Vector data

Discrete objects

Examples: countries, roads, rivers, towns

Contain: - geometry: shape & location of the objects

- attributes: additional variables (e.g. name, year, type)

Common file format: GeoJSON , shapefile

Raster data

Continuous phenomena or spatial fields

Examples: temperature, air quality, elevation, water depth

Common file formats: TIFF , GeoTIFF , NetCDF , Esri grid

Vector data

Types

- **point**: single set of coordinates
- **multi-point**: multiple sets of coordinates
- **polyline**: multiple sets for which the order matters
- **multi-polyline**: multiple of the above
- **polygon**: same as polyline but first & last sets are the same
- **multi-polygon**: multiple of the above

Raster data

Grid of equally sized rectangular cells containing values for some variables

Size of cells = resolution

For computing efficiency, rasters do not have coordinates of each cell, but the bounding box & the number of rows & columns

Coordinate Reference Systems (CRS)

A location on Earth's surface can be identified by its **coordinates** & some **reference system** called CRS

The coordinates (`x`, `y`) are called longitude & latitude

There can be a 3rd coordinate (`z`) for elevation or other measurement—usually a vertical one & a 4th (`m`) for some other data attribute—usually a horizontal measurement

In 3D, longitude & latitude are expressed in angular units (e.g. degrees) & the reference system needed is an angular CRS or geographic coordinate system (GCS)

In 2D, they are expressed in linear units (e.g. meters) & the reference system needed is a planar CRS or projected coordinate system (PCS)

Datums

Since the Earth is not a perfect sphere, we use spheroidal models to represent its surface. Those are called **geodetic datums**

Some datums are global, others local (more accurate in a particular area of the globe, but only useful there)

Examples of commonly used global datums:

- WGS84 (World Geodesic System 1984)
- NAD83 (North American Datum of 1983)

Angular CRS

An angular CRS contains a datum, an angular unit & references such as a prime meridian (e.g. the Royal Observatory, Greenwich, England)

In an angular CRS or GCS:

- Longitude (λ) represents the angle between the prime meridian & the meridian that passes through that location
- Latitude (ϕ) represents the angle between the line that passes through the center of the Earth & that location & its projection on the equatorial plane

Longitude & latitude are thus angular coordinates

Projections

To create a two-dimensional map, you need to project this 3D angular CRS into a 2D one

Various projections offer different characteristics. For instance:

- some respect areas (equal-area)
- some respect the shape of geographic features (conformal)
- some *almost* respect both for small areas

It is important to choose one with sensible properties for your goals

Examples of projections:

- Mercator
- UTM
- Robinson

Planar CRS

A planar CRS is defined by a datum, a projection & a set of parameters such as a linear unit & the origins

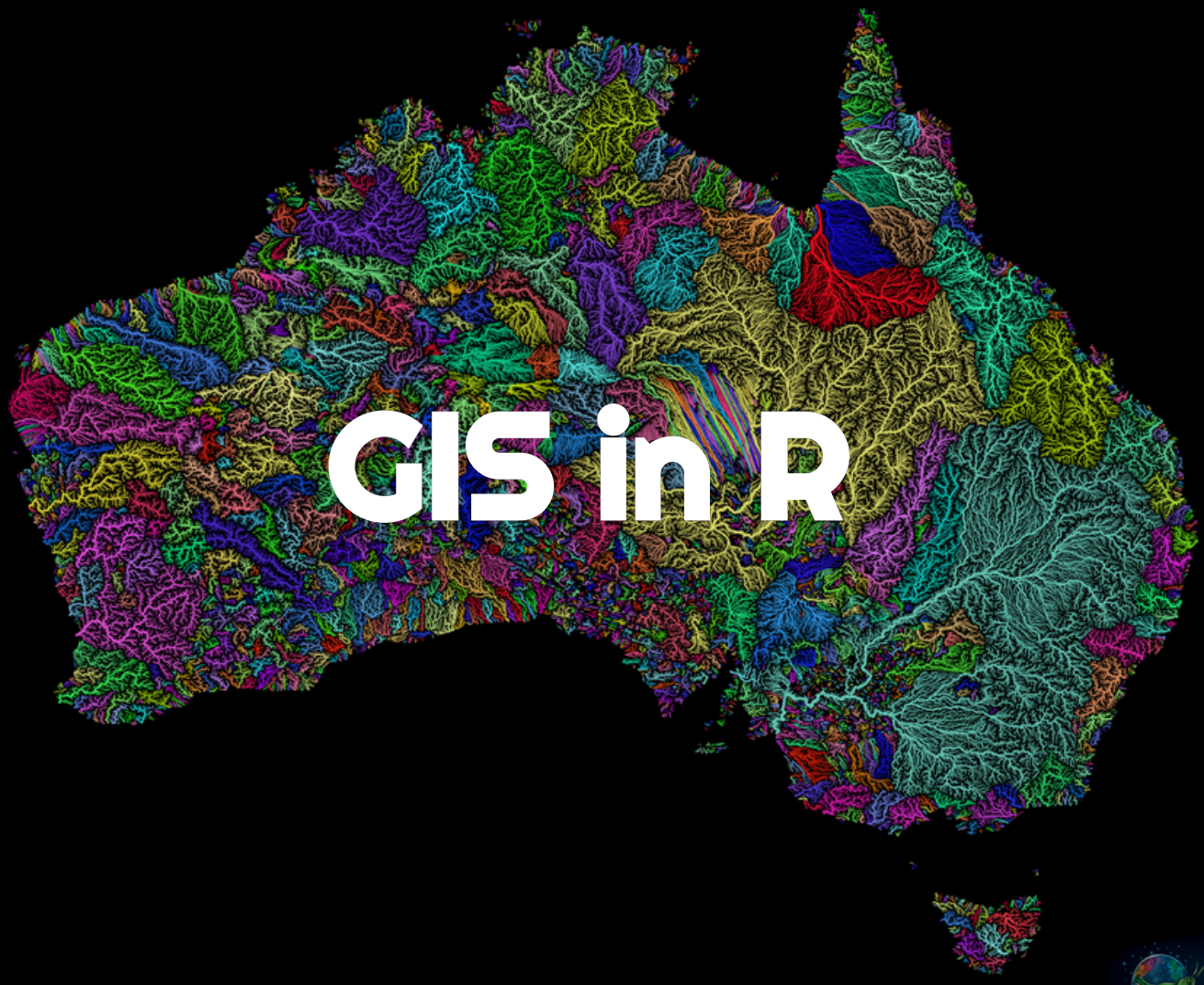
Common planar CRS have been assigned a unique ID called EPSG code which is much more convenient to use

In a planar CRS, coordinates will not be in degrees anymore but in meters (or other length unit)

Projecting into a new CRS

You can change the projection of your data

Vector data won't suffer any loss of precision, but raster data will
→ best to try to avoid reprojecting rasters: if you want to combine various datasets which have different projections, reproject vector data instead


GIS in R
Grasshopper
Geography

Open GIS data

Free GIS Data : list of free GIS datasets

Books

Geocomputation with R by Robin Lovelace, Jakub Nowosad & Jannes Muenchow

Spatial Data Science by Edzer Pebesma & Roger Bivand

Spatial Data Science with R by Robert J. Hijmans

Using Spatial Data with R by Claudia A. Engel

Tutorial

An Introduction to Spatial Data Analysis and Visualisation in R by the CDRC

Website

r-spatial by Edzer Pebesma, Marius Appel & Daniel Nüst

CRAN package list

Analysis of Spatial Data

Mailing list

R Special Interest Group on using Geographical data and Mapping

Packages

There is now a rich ecosystem of GIS packages in R[1]

1. Bivand, R.S. Progress in the R ecosystem for representing and handling spatial data. J Geogr Syst (2020). https://doi.org/10.1007/s10109-020-00336-0

Data manipulation

Older packages

- **sp**
- **raster**
- **rgdal**
- **rgeos**

Newer generation

- sf : vector data
- terra : raster data (also has vector data capabilities)

Mapping

Static maps

- ggplot2 + ggspatial
- tmap

Dynamic maps

- leaflet
- ggplot2 + gganimate
- mapview
- ggmap
- tmap

Simple Features in R

Geospatial vectors: points, lines, polygons

Simple Features

Simple Features —defined by the Open Geospatial Consortium (OGC) & formalized by ISO —is a set of standards now used by most GIS libraries

Well-known text (WKT) is a markup language for representing vector geometry objects according to those standards

A compact computer version also exists—well-known binary (WKB)—used by spatial databases

The package **sp** predates Simple Features

sf—launched in 2016—implements these standards in R in the form of sf objects: data.frames (or tibbles) containing the attributes, extended by sfc objects or simple feature geometries list-columns

sf

Useful links

- GitHub repo
- Paper
- Resources
- Cheatsheet
- 6 vignettes: 1 , 2 , 3 , 4 , 5 , 6

Spatial manipulation with sf: : CHEAT SHEET

The sf package provides a set of tools for working with geospatial vectors, i.e. points, lines, polygons, etc.

Geometric confirmation

st_contains(x, y, ...) Identifies if x is within y (i.e. point within polygon)

st_covered_by(x, y, ...) Identifies if x is completely within y (i.e. polygon completely within polygon)

st_covers(x, y, ...) Identifies if any point from x is outside of y (i.e. polygon outside polygon)

st_crosses(x, y, ...) Identifies if any geometry of x have commonalities with y

st_disjoint(x, y, ...) Identifies when geometries from x do not share space with y

st_equals(x, y, ...) Identifies if x and y share the same geometry

st_intersects(x, y, ...) Identifies if x and y geometry share any space

st_overlaps(x, y, ...) Identifies if geometries of x and y share space, are of the same dimension, but are not completely contained by each other

st_touches(x, y, ...) Identifies if geometries of x and y share a common point but their interiors do not intersect

st_within(x, y, ...) Identifies if x is in a specified distance to y

Geometric operations

st_boundary(x) Creates a polygon that encompasses the full extent of the geometry

st_buffer(x, dist, nQuadSegs) Creates a polygon covering all points of the geometry within a given distance

st_centroid(x, ..., of_largest_polygon) Creates a point at the geometric centre of the geometry

st_convex_hull(x) Creates geometry that represents the minimum convex geometry of x

st_line_merge(x) Creates linestring geometry from sewing multi linestring geometry together

st_node(x) Creates nodes on overlapping geometry where nodes do not exist

st_point_on_surface(x) Creates a point that is guarenteed to fall on the surface of the geometry

st_polygonize(x) Creates polygon geometry from linestring geometry

st_segmentize(x, dfMaxLength, ...) Creates linesting geometry from x based on a specified length

st_simplify(x, preserveTopology, dTolerance) Creates a simplified version of the geometry based on a specified tolerance

Geometry creation

st_triangulate(x, dTolerance, bOnlyEdges) Creates polygon geometry as triangles from point geometry

st_voronoi(x, envelope, dTolerance, bOnlyEdges) Creates polygon geometry covering the envolope of x, with x at the centre of the geometry

st_point(x, c(numeric vector), dim = "XYZ") Creating point geometry from numeric values

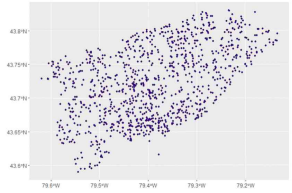
st_multipoint(x = matrix(numeric values in rows), dim = "XYZ") Creating multi point geometry from numeric values

st_linestring(x = matrix(numeric values in rows), dim = "XYZ") Creating linestring geometry from numeric values

st_multilinestring(x = list(numeric matricesin rows), dim = "XYZ") Creating multi linestring geometry from numeric values

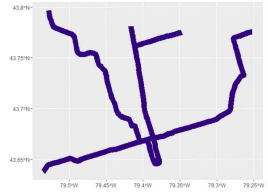
st_polygon(x = list(numeric matrices in rows), dim = "XYZ") Creating polygon geometry from numeric values

st_multipolygon(x = list(numeric matrices in rows), dim = "XYZ") Creating multi polygon geometry from numeric values



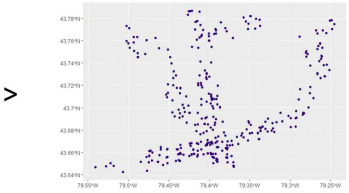
```
ggplot() +
  geom_sf(data = schools)
```

+



```
ggplot() +
  geom_sf(data = subway)
```

=>



```
ggplot() +
  geom_sf(data = st_intersection(schools, st_buffer(subway, 1000)))
```

This cheatsheet presents the sf package [Edzer Pebesma 2018] in version 0.6.3. See https://github.com/r-spatial/sf for more details.

Spatial manipulation with sf: : CHEAT SHEET

The sf package provides a set of tools for working with geospatial vectors, i.e. points, lines, polygons, etc.

Geometry operations

st_contains(x, y, ...) Identifies if x is within y (i.e. point within polygon)

st_crop(x, y, ..., xmin, ymin, xmax, ymax) Creates geometry of x that intersects a specified rectangle

st_difference(x, y) Creates geometry from x that does not intersect with y

st_intersection(x, y) Creates geometry of the shared portion of x and y

st_sym_difference(x, y) Creates geometry representing portions of x and y that do not intersect

st_snap(x, y, tolerance) Snap nodes from geometry x to geometry y

st_union(x, y, ..., by_feature) Creates multiple geometries into a a single geometry, consisiting of all geometry elements

Geometric measurement

st_area(x) Calculate the surface area of a polygon geometry based on the current coordinate reference system

st_distance(x, y, ..., dist_fun, by_element, which) Calculates the 2D distance between x and y based on the current coordinate system

st_length(x) Calculates the 2D length of a geometry based on the current coordinate system

Misc operations

st_as_sf(x, ...) Create a sf object from a non-geospatial tabular data frame

st_cast(x, to, ...) Change x geometry to a different geometry type

st_coordinates(x, ...) Creates a matrix of coordinate values from x

st_crs(x, ...) Identifies the coordinate reference system of x

st_join(x, y, join, FUN, suffix, ...) Performs a spatial left or inner join between x and y

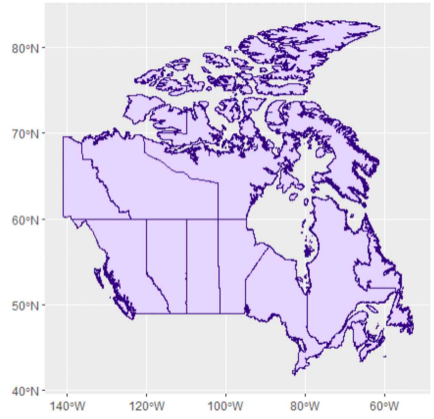
st_make_grid(x, cellsize, offset, n, crs, what) Creates rectangular grid geometry over the bounding box of x

st_nearest_feature(x, y) Creates an index of the closest feature between x and y

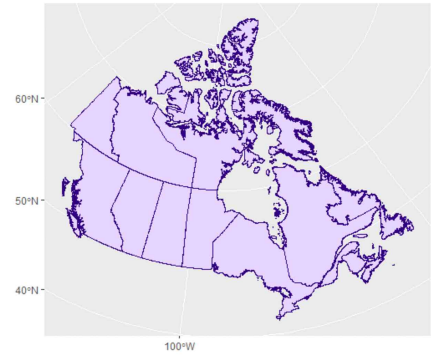
st_nearest_points(x, y, ...) Returns the closest point between x and y

st_read(dsn, layer, ...) Read file or database vector dataset as a sf object

st_transform(x, crs, ...) Convert coordinates of x to a different coordinate reference system




```
ggplot() +
  geom_sf(data = cdn) +
  coord_sf(crs = st_crs(4326))
```




```
ggplot() +
  geom_sf(data = cdn) +
  coord_sf(crs = st_crs(3347))
```

This cheatsheet presents the sf package [Edzer Pebesma 2018] in version 0.6.3. See https://github.com/r-spatial/sf for more details.

sf objects

```
> Simple feature collection with 27108 features and 4 fields
Geometry type: POLYGON
Dimension:     XY
Bounding box:  xmin: -176.1425 ymin: 52.05727 xmax: -126.8545 ymax: 69.35167
Geodetic CRS:  WGS 84
First 10 features:
            RGIId    CenLon   CenLat   Area                       geometry
1  RGI60-01.00001 -146.8230 63.68900  0.360 POLYGON ((-146.818 63.69081...
2  RGI60-01.00002 -146.6680 63.40400  0.558 POLYGON ((-146.6635 63.4076...
3  RGI60-01.00003 -146.0800 63.37600  1.685 POLYGON ((-146.0723 63.3834...
4  RGI60-01.00004 -146.1200 63.38100  3.681 POLYGON ((-146.149 63.37919...
5  RGI60-01.00005 -147.0570 63.55100  2.573 POLYGON ((-147.0431 63.5502...
6  RGI60-01.00006 -146.2440 63.57100 10.470 POLYGON ((-146.2436 63.5562...
7  RGI60-01.00007 -146.2295 63.55085  0.649 POLYGON ((-146.2495 63.5531...
8  RGI60-01.00008 -146.2960 63.54300  0.200 POLYGON ((-146.2992 63.5443...
9  RGI60-01.00009 -147.6000 63.65900  1.517 POLYGON ((-147.6147 63.6643...
10 RGI60-01.00010 -147.1700 63.51300  3.806 POLYGON ((-147.1494 63.5098...
```

sf objects

```
> Simple feature collection with 27108 features and 4 fields
Geometry type: POLYGON
Dimension:     XY
Bounding box:  xmin: -176.1425 ymin: 52.05727 xmax: -126.8545 ymax: 69.35167
Geodetic CRS:  WGS 84
First 10 features:
```

	RGIId	CenLon	CenLat	Area	geometry
1	RGI60-01.00001	-146.8230	63.68900	0.360	POLYGON ((-146.818 63.69081...
2	RGI60-01.00002	-146.6680	63.40400	0.558	POLYGON ((-146.6635 63.4076...
3	RGI60-01.00003	-146.0800	63.37600	1.685	POLYGON ((-146.0723 63.3834...
4	RGI60-01.00004	-146.1200	63.38100	3.681	POLYGON ((-146.149 63.37919...
5	RGI60-01.00005	-147.0570	63.55100	2.573	POLYGON ((-147.0431 63.5502...
6	RGI60-01.00006	-146.2440	63.57100	10.470	POLYGON ((-146.2436 63.5562...
7	RGI60-01.00007	-146.2295	63.55085	0.649	POLYGON ((-146.2495 63.5531...
8	RGI60-01.00008	-146.2960	63.54300	0.200	POLYGON ((-146.2992 63.5443...
9	RGI60-01.00009	-147.6000	63.65900	1.517	POLYGON ((-147.6147 63.6643...
10	RGI60-01.00010	-147.1700	63.51300	3.806	POLYGON ((-147.1494 63.5098...

sf object

sf objects

```
> Simple feature collection with 27108 features and 4 fields
Geometry type: POLYGON
Dimension:     XY
Bounding box:  xmin: -176.1425 ymin: 52.05727 xmax: -126.8545 ymax: 69.35167
Geodetic CRS:  WGS 84
First 10 features:
            RGIId    CenLon   CenLat   Area                       geometry
1  RGI60-01.00001 -146.8230 63.68900  0.360 POLYGON ((-146.818 63.69081...
2  RGI60-01.00002 -146.6680 63.40400  0.558 POLYGON ((-146.6635 63.4076...
3  RGI60-01.00003 -146.0800 63.37600  1.685 POLYGON ((-146.0723 63.3834...
4  RGI60-01.00004 -146.1200 63.38100  3.681 POLYGON ((-146.149 63.37919...
5  RGI60-01.00005 -147.0570 63.55100  2.573 POLYGON ((-147.0431 63.5502...
6  RGI60-01.00006 -146.2440 63.57100 10.470 POLYGON ((-146.2436 63.5562...
7  RGI60-01.00007 -146.2295 63.55085  0.649 POLYGON ((-146.2495 63.5531...
8  RGI60-01.00008 -146.2960 63.54300  0.200 POLYGON ((-146.2992 63.5443...
9  RGI60-01.00009 -147.6000 63.65900  1.517 POLYGON ((-147.6147 63.6643...
10 RGI60-01.00010 -147.1700 63.51300  3.806 POLYGON ((-147.1494 63.5098...
```

sfc object:

geometry list-column

sf objects

```
> Simple feature collection with 27108 features and 4 fields
Geometry type: POLYGON
Dimension:     XY
Bounding box:  xmin: -176.1425 ymin: 52.05727 xmax: -126.8545 ymax: 69.35167
Geodetic CRS:  WGS 84
First 10 features:
            RGIId    CenLon   CenLat   Area                       geometry
1  RGI60-01.00001 -146.8230 63.68900  0.360 POLYGON ((-146.818 63.69081...
2  RGI60-01.00002 -146.6680 63.40400  0.558 POLYGON ((-146.6635 63.4076...
3  RGI60-01.00003 -146.0800 63.37600  1.685 POLYGON ((-146.0723 63.3834...
4  RGI60-01.00004 -146.1200 63.38100  3.681 POLYGON ((-146.149 63.37919...
5  RGI60-01.00005 -147.0570 63.55100  2.573 POLYGON ((-147.0431 63.5502...
6  RGI60-01.00006 -146.2440 63.57100 10.470 POLYGON ((-146.2436 63.5562...
7  RGI60-01.00007 -146.2295 63.55085  0.649 POLYGON ((-146.2495 63.5531...
8  RGI60-01.00008 -146.2960 63.54300  0.200 POLYGON ((-146.2992 63.5443...
9  RGI60-01.00009 -147.6000 63.65900  1.517 POLYGON ((-147.6147 63.6643...
10 RGI60-01.00010 -147.1700 63.51300  3.806 POLYGON ((-147.1494 63.5098...
```

sfg object:

feature geometry

sf objects

```
> Simple feature collection with 27108 features and 4 fields
Geometry type: POLYGON
Dimension:     XY
Bounding box:  xmin: -176.1425 ymin: 52.05727 xmax: -126.8545 ymax: 69.35167
Geodetic CRS:  WGS 84
First 10 features:
            RGIId    CenLon   CenLat   Area                       geometry
1  RGI60-01.00001 -146.8230 63.68900  0.360 POLYGON ((-146.818 63.69081...
2  RGI60-01.00002 -146.6680 63.40400  0.558 POLYGON ((-146.6635 63.4076...
3  RGI60-01.00003 -146.0800 63.37600  1.685 POLYGON ((-146.0723 63.3834...
4  RGI60-01.00004 -146.1200 63.38100  3.681 POLYGON ((-146.149 63.37919...
5  RGI60-01.00005 -147.0570 63.55100  2.573 POLYGON ((-147.0431 63.5502...
6  RGI60-01.00006 -146.2440 63.57100 10.470 POLYGON ((-146.2436 63.5562...
7  RGI60-01.00007 -146.2295 63.55085  0.649 POLYGON ((-146.2495 63.5531...
8  RGI60-01.00008 -146.2960 63.54300  0.200 POLYGON ((-146.2992 63.5443...
9  RGI60-01.00009 -147.6000 63.65900  1.517 POLYGON ((-147.6147 63.6643...
10 RGI60-01.00010 -147.1700 63.51300  3.806 POLYGON ((-147.1494 63.5098...
```

feature

sf functions

Most functions start with `st_` (which refers to “spatial type”)

terra

Geospatial rasters

Faster and simpler replacement for the **raster** package by the same team

Mostly implemented in C++

Can work with datasets too large to be loaded into memory

terra

Useful links

- GitHub repo
- Resources
- Full manual

tmap

Layered grammar of graphics GIS maps

tmap

Useful links

- GitHub repo
- Resources

Help pages and vignettes

```
?tmap-element
vignette("tmap-getstarted")
# All the usual help pages, e.g.:
?tm_layout
```

tmap functions

Main functions start with `tmap_`

Functions creating map elements start with `tm_`

tmap functioning

Very similar to **ggplot2**

Typically, a map contains:

- One or multiple layer(s) (the order matters as they stack on top of each other)
- Some layout (e.g. customization of title, background, margins): `tm_layout`
- A compass: `tm_compass`
- A scale bar: `tm_scale_bar`

Each layer contains:

- Some data: `tm_shape`
- How that data will be represented: e.g. `tm_polygons`, `tm_lines`, `tm_raster`

tmap example

```
tm_shape(ak, bbox = nwa_bbox) +
  tm_polygons() +
  tm_shape(wes) +
  tm_polygons() +
  tm_layout(
    title = "Glaciers of Western North America",
    title.position = c("center", "top"),
    title.size = 1.1,
    bg.color = "#fcfcfc",
    inner.margins = c(0.06, 0.01, 0.09, 0.01),
    outer.margins = 0,
    frame.lwd = 0.2
  ) +
  tm_compass(
    type = "arrow",
    position = c("right", "top"),
    size = 1.2,
    text.size = 0.6
  ) +
  tm_scale_bar(
    breaks = c(0, 1000, 2000),
    position = c("right", "BOTTOM")
  )
```

tmap example

```
tm_shape(ak, bbox = nwa_bbox) +
  tm_polygons() +
  tm_shape(wes) +
  tm_polygons() +
  tm_layout(
    title = "Glaciers of Western North America",
    title.position = c("center", "top"),
    title.size = 1.1,
    bg.color = "#fcfcfc",
    inner.margins = c(0.06, 0.01, 0.09, 0.01),
    outer.margins = 0,
    frame.lwd = 0.2
  ) +
  tm_compass(
    type = "arrow",
    position = c("right", "top"),
    size = 1.2,
    text.size = 0.6
  ) +
  tm_scale_bar(
    breaks = c(0, 1000, 2000),
    position = c("right", "BOTTOM")
  )
```

1st layer

tmap example

```
tm_shape(ak, bbox = nwa_bbox) +
  tm_polygons() +
  tm_shape(wes) +
  tm_polygons() +
  tm_layout(
    title = "Glaciers of Western North America",
    title.position = c("center", "top"),
    title.size = 1.1,
    bg.color = "#fcfcfc",
    inner.margins = c(0.06, 0.01, 0.09, 0.01),
    outer.margins = 0,
    frame.lwd = 0.2
  ) +
  tm_compass(
    type = "arrow",
    position = c("right", "top"),
    size = 1.2,
    text.size = 0.6
  ) +
  tm_scale_bar(
    breaks = c(0, 1000, 2000),
    position = c("right", "BOTTOM")
  )
```

1st layer

How 1st layer data is represented

tmap example

```
tm_shape(ak, bbox = nwa_bbox) +
  tm_polygons() +
  tm_shape(wes) +
  tm_polygons() +
  tm_layout(
    title = "Glaciers of Western North America",
    title.position = c("center", "top"),
    title.size = 1.1,
    bg.color = "#fcfcfc",
    inner.margins = c(0.06, 0.01, 0.09, 0.01),
    outer.margins = 0,
    frame.lwd = 0.2
  ) +
  tm_compass(
    type = "arrow",
    position = c("right", "top"),
    size = 1.2,
    text.size = 0.6
  ) +
  tm_scale_bar(
    breaks = c(0, 1000, 2000),
    position = c("right", "BOTTOM")
  )
```

1st layer

How 1st layer data is represented

2nd layer

tmap example

```
tm_shape(ak, bbox = nwa_bbox) +
  tm_polygons() +
  tm_shape(wes) +
  tm_polygons() +
  tm_layout(
    title = "Glaciers of Western North America",
    title.position = c("center", "top"),
    title.size = 1.1,
    bg.color = "#fcfcfc",
    inner.margins = c(0.06, 0.01, 0.09, 0.01),
    outer.margins = 0,
    frame.lwd = 0.2
  ) +
  tm_compass(
    type = "arrow",
    position = c("right", "top"),
    size = 1.2,
    text.size = 0.6
  ) +
  tm_scale_bar(
    breaks = c(0, 1000, 2000),
    position = c("right", "BOTTOM")
  )
```

1st layer

How 1st layer data is represented

2nd layer

How 2nd layer data is represented

tmap example

```
tm_shape(ak, bbox = nwa_bbox) +
  tm_polygons() +
  tm_shape(wes) +
  tm_polygons() +
  tm_layout(
    title = "Glaciers of Western North America",
    title.position = c("center", "top"),
    title.size = 1.1,
    bg.color = "#fcfcfc",
    inner.margins = c(0.06, 0.01, 0.09, 0.01),
    outer.margins = 0,
    frame.lwd = 0.2
  ) +
  tm_compass(
    type = "arrow",
    position = c("right", "top"),
    size = 1.2,
    text.size = 0.6
  ) +
  tm_scale_bar(
    breaks = c(0, 1000, 2000),
    position = c("right", "BOTTOM")
  )
```

1st layer

How 1st layer data is represented

2nd layer

How 2nd layer data is represented

Layout

tmap example

```
tm_shape(ak, bbox = nwa_bbox) +
  tm_polygons() +
  tm_shape(wes) +
  tm_polygons() +
  tm_layout(
    title = "Glaciers of Western North America",
    title.position = c("center", "top"),
    title.size = 1.1,
    bg.color = "#fcfcfc",
    inner.margins = c(0.06, 0.01, 0.09, 0.01),
    outer.margins = 0,
    frame.lwd = 0.2
  ) +
  tm_compass(
    type = "arrow",
    position = c("right", "top"),
    size = 1.2,
    text.size = 0.6
  ) +
  tm_scale_bar(
    breaks = c(0, 1000, 2000),
    position = c("right", "BOTTOM")
  )
```

1st layer

How 1st layer data is represented

2nd layer

How 2nd layer data is represented

Layout

Compass

tmap example

```
tm_shape(ak, bbox = nwa_bbox) +
  tm_polygons() +
  tm_shape(wes) +
  tm_polygons() +
  tm_layout(
    title = "Glaciers of Western North America",
    title.position = c("center", "top"),
    title.size = 1.1,
    bg.color = "#fcfcfc",
    inner.margins = c(0.06, 0.01, 0.09, 0.01),
    outer.margins = 0,
    frame.lwd = 0.2
  ) +
  tm_compass(
    type = "arrow",
    position = c("right", "top"),
    size = 1.2,
    text.size = 0.6
  ) +
  tm_scale_bar(
    breaks = c(0, 1000, 2000),
    position = c("right", "BOTTOM")
  )
```

1st layer

How 1st layer data is represented

2nd layer

How 2nd layer data is represented

Layout

Compass

Scale

ggplot2

The standard in R plots

ggplot2

Useful links

- GitHub repo
- Resources
- Cheatsheet

Data Visualization with ggplot2 : : CHEAT SHEET

Basics

ggplot2 is based on the **grammar of graphics**, the idea that you can build every graph from the same components: a **data** set, a **coordinate system**, and geoms—visual marks that represent data points.

To display values, map variables in the data to visual properties of the geom (**aesthetics**) like **size**, **color**, and **x** and **y** locations.

Complete the template below to build a graph.

required

```
ggplot (data = <DATA>) +
<GEOM_FUNCTION> (mapping = aes( <MAPPINGS>),
  stat = <STAT>, position = <POSITION> ) +
<COORDINATE_FUNCTION> +
<FACET_FUNCTION> +
<SCALE_FUNCTION> +
<THEME_FUNCTION>
```

Not required, sensible defaults supplied

ggplot(data = mpg, **aes(**x = cty, y = hwy**))** Begins a plot that you finish by adding layers to. Add one geom function per layer.

aesthetic mappings | data | geom

qplot(x = cty, y = hwy, data = mpg, geom = "point**")** Creates a complete plot with given data, geom, and mappings. Supplies many useful defaults.

last_plot() Returns the last plot

ggsave("plot.png", width = 5, height = 5) Saves last plot as 5' x 5' file named "plot.png" in working directory. Matches file type to file extension.

Geoms

Use a geom function to represent data points, use the geom's aesthetic properties to represent variables. Each function returns a layer.

GRAPHICAL PRIMITIVES

```
a <- ggplot(economics, aes(date, unemploy))
b <- ggplot(seals, aes(x = long, y = lat))
```

a + geom_blank()
(Useful for expanding limits)

b + geom_curve(aes(yend = lat + 1, xend=long+1),curvature=1**)** - x, xend, y, yend, alpha, angle, color, curvature, linetype, size

a + geom_path(lineend="butt", linejoin="round", linemitre=1**)**
x, y, alpha, color, group, linetype, size

a + geom_polygon(aes(group = group**))**
x, y, alpha, color, fill, group, linetype, size

b + geom_rect(aes(xmin = long, ymin=lat, xmax= long + 1, ymax = lat + 1**))** - xmax, xmin, ymax, ymin, alpha, color, fill, linetype, size

a + geom_ribbon(aes(ymin=unemploy - 900, ymax=unemploy + 900**))** - x, ymax, ymin, alpha, color, fill, group, linetype, size

LINE SEGMENTS

common aesthetics: x, y, alpha, color, linetype, size

b + geom_abline(aes(intercept=0, slope=1**))**
b + geom_hline(aes(yintercept = lat**))**
b + geom_vline(aes(xintercept = long**))**

b + geom_segment(aes(yend=lat+1, xend=long+1**))**
b + geom_spoke(aes(angle = 1:1155, radius = 1**))**

ONE VARIABLE continuous

```
c <- ggplot(mpg, aes(hwy)); c2 <- ggplot(mpg)
```

c + geom_area(stat = "bin")
x, y, alpha, color, fill, linetype, size

c + geom_density(kernel = "gaussian"**)**
x, y, alpha, color, fill, group, linetype, size, weight

c + geom_dotplot()
x, y, alpha, color, fill

c + geom_freqpoly() x, y, alpha, color, group, linetype, size

c + geom_histogram(binwidth = 5**)** x, y, alpha, color, fill, linetype, size, weight

c2 + geom_qq(aes(sample = hwy**))** x, y, alpha, color, fill, linetype, size, weight

discrete

```
d <- ggplot(mpg, aes(fl))
```

d + geom_bar()
x, alpha, color, fill, linetype, size, weight

TWO VARIABLES

continuous x , continuous y

```
e <- ggplot(mpg, aes(cty, hwy))
```

e + geom_label(aes(label = cty), nudge_x = 1, nudge_y = 1, check_overlap = TRUE**)** x, y, label, alpha, angle, color, family, fontface, hjust, lineheight, size, vjust

e + geom_jitter(height = 2, width = 2**)**
x, y, alpha, color, fill, shape, size

e + geom_point(), x, y, alpha, color, fill, shape, size, stroke

e + geom_quantile(), x, y, alpha, color, group, linetype, size, weight

e + geom_rug(sides = "bl"**)**, x, y, alpha, color, linetype, size

e + geom_smooth(method = lm**)**, x, y, alpha, color, fill, group, linetype, size, weight

e + geom_text(aes(label = cty), nudge_x = 1, nudge_y = 1, check_overlap = TRUE**)**, x, y, label, alpha, angle, color, family, fontface, hjust, lineheight, size, vjust

discrete x , continuous y

```
f <- ggplot(mpg, aes(class, hwy))
```

f + geom_col(), x, y, alpha, color, fill, group, linetype, size

f + geom_boxplot(), x, y, lower, middle, upper, ymax, ymin, alpha, color, fill, group, linetype, shape, size, weight

f + geom_dotplot(binaxis = "y", stackdir = "center"**)**, x, y, alpha, color, fill, group

f + geom_violin(scale = "area"**)**, x, y, alpha, color, fill, group, linetype, size, weight

discrete x , discrete y

```
g <- ggplot(diamonds, aes(cut, color))
```

g + geom_count(), x, y, alpha, color, fill, shape, size, stroke

continuous bivariate distribution

```
h <- ggplot(diamonds, aes(carat, price))
```

h + geom_bin2d(binwidth = c(0.25, 500)**)**
x, y, alpha, color, fill, linetype, size, weight

h + geom_density2d()
x, y, alpha, colour, group, linetype, size

h + geom_hex()
x, y, alpha, colour, fill, size

continuous function

```
i <- ggplot(economics, aes(date, unemploy))
```

i + geom_area()
x, y, alpha, color, fill, linetype, size

i + geom_line()
x, y, alpha, color, group, linetype, size

i + geom_step(direction = "hv")
x, y, alpha, color, group, linetype, size

visualizing error

```
df <- data.frame(grp = c("A", "B"), fit = 4:5, se = 1:2)
j <- ggplot(df, aes(grp, fit, ymin = fit-se, ymax = fit+se))
```

j + geom_crossbar(fatten = 2**)**
x, y, ymax, ymin, alpha, color, fill, group, linetype, size

j + geom_errorbar(), x, ymax, ymin, alpha, color, group, linetype, size, width (also **geom_errorbarh()**)

j + geom_linerange()
x, ymin, ymax, alpha, color, group, linetype, size

j + geom_pointrange()
x, y, ymin, ymax, alpha, color, fill, group, linetype, shape, size

maps

```
data <- data.frame(murder = USArrests$Murder,
state = tolower(rownames(USArrests)))
map <- map_data("state")
k <- ggplot(data, aes(fill = murder))
```

k + geom_map(aes(map_id = state), map = map**)**
+ expand_limits(x = map$long, y = map$lat**)**,
map_id, alpha, color, fill, linetype, size

THREE VARIABLES

```
seals$z <- with(seals, sqrt(delta_long^2 + delta_lat^2)); l <- ggplot(seals, aes(long, lat))
```

l + geom_contour(aes(z = z))
x, y, z, alpha, colour, group, linetype, size, weight

l + geom_raster(aes(fill = z), hjust=0.5, vjust=0.5, interpolate=FALSE**)**
x, y, alpha, fill

l + geom_tile(aes(fill = z)**)**, x, y, alpha, color, fill, linetype, size, width

Stats An alternative way to build a layer

A stat builds new variables to plot (e.g., count, prop).

Visualize a stat by changing the default stat of a geom function, **geom_bar(stat="count")** or by using a stat function, **stat_count(geom="bar")**, which calls a default geom to make a layer (equivalent to a geom function). Use **..name..** syntax to map stat variables to aesthetics.

geom to use | stat function | geommappings

i + stat_density2d(aes(fill = ..level..), geom = "polygon")

variable created by stat

c + stat_bin(binwidth = 1, origin = 10)
x, y | ..count.., ..ncount.., ..density.., ..ndensity..

c + stat_count(width = 1) **x, y,** | ..count.., ..prop..

c + stat_density(adjust = 1, kernel = "gaussian")
x, y, | ..count.., ..density.., ..scaled..

e + stat_bin_2d(bins = 30, drop = T)
x, y, fill | ..count.., ..density..

e + stat_bin_hex(bins=30) **x, y, fill** | ..count.., ..density..

e + stat_density_2d(contour = TRUE, n = 100)
x, y, color, size | ..level..

e + stat_ellipse(level = 0.95, segments = 51, type = "t")

l + stat_contour(aes(z = z)) **x, y, z, order** | ..level..

l + stat_summary_hex(aes(z = z), bins = 30, fun = max)
x, y, z, fill | ..value..

l + stat_summary_2d(aes(z = z), bins = 30, fun = mean)
x, y, z, fill | ..value..

f + stat_boxplot(coef = 1.5) **x, y** | ..lower.., ..middle.., ..upper.., ..width.. , ..ymin.., ..ymax..

f + stat_ydensity(kernel = "gaussian", scale = "area") **x, y** | ..density.., ..scaled.., ..count.., ..n.., ..violinwidth.., ..width..

e + stat_ecdf(n = 40) **x, y** | ..x.., ..y..

e + stat_quantile(quantiles = c(0.1, 0.9), formula = y ~ log(x), method = "rq") **x, y** | ..quantile..

e + stat_smooth(method = "lm", formula = y ~ x, se=T, level=0.95) **x, y** | ..se.., ..x.., ..y.., ..ymin.., ..ymax..

ggplot() + stat_function(aes(x = -3:3), n = 99, fun = dnorm, args = list(sd=0.5)) **x** | ..x.., ..y..

e + stat_identity(na.rm = TRUE)

ggplot() + stat_qq(aes(sample=1:100), dist = qt, dparam=list(df=5)) **sample, x, y** | ..sample.., ..theoretical..

e + stat_sum() x, y, size | ..n.., ..prop..

e + stat_summary(fun.data = "mean_cl_boot")

h + stat_summary_bin(fun = "mean", geom = "bar")

e + stat_unique()

Scales

Scales map data values to the visual values of an aesthetic. To change a mapping, add a new scale.

(n <- d + geom_bar(aes(fill = fl)))

scale_ | aesthetic to adjust | prepackaged scale to use | scale-specific arguments

n + scale_fill_manual(
values = c("skyblue", "royalblue", "blue", "navy"),
limits = c("d", "e", "p", "r"), breaks =c("d", "e", "p", "r"),
name = "fuel", labels = c("D", "E", "P", "R"))

range of values to include in mapping | title to use in legend/axis | labels to use in legend/axis | breaks to use in legend/axis

GENERAL PURPOSE SCALES

Use with most aesthetics

scale_*_continuous() - map cont' values to visual ones
scale_*_discrete() - map discrete values to visual ones
scale_*_identity() - use data values **as** visual ones
scale_*_manual(values = c()) - map discrete values to manually chosen visual ones
scale_*_date(date_labels = "%m/%d"), date_breaks = "2 weeks") - treat data values as dates.
scale_*_datetime() - treat data x values as date times. Use same arguments as scale_x_date(). See ?strptime for label formats.

X & Y LOCATION SCALES

Use with x or y aesthetics (x shown here)

scale_x_log10() - Plot x on log10 scale
scale_x_reverse() - Reverse direction of x axis
scale_x_sqrt() - Plot x on square root scale

COLOR AND FILL SCALES (DISCRETE)

n <- d + geom_bar(aes(fill = fl))

n + scale_fill_brewer(palette = "Blues")
For palette choices:
RColorBrewer::display.brewer.all()

n + scale_fill_grey(start = 0.2, end = 0.8, na.value = "red")

COLOR AND FILL SCALES (CONTINUOUS)

o <- c + geom_dotplot(aes(fill = ..x..))

o + scale_fill_distiller(palette = "Blues")

o + scale_fill_gradient(low="red", high="yellow")

o + scale_fill_gradient2(low="red", high="blue", mid = "white", midpoint = 25)

o + scale_fill_gradientn(colours=topo.colors(6))
Also: rainbow(), heat.colors(), terrain.colors(), cm.colors(), RColorBrewer::brewer.pal()

SHAPE AND SIZE SCALES

p <- e + geom_point(aes(shape = fl, size = cyl))
p + scale_shape() + scale_size()
p + scale_shape_manual(values = c(3:7))

p + scale_radius(range = c(1,6))
p + scale_size_area(max_size = 6)

Coordinate Systems

r <- d + geom_bar()

r + coord_cartesian(xlim = c(0, 5))
xlim, ylim
The default cartesian coordinate system

r + coord_fixed(ratio = 1/2)
ratio, xlim, ylim
Cartesian coordinates with fixed aspect ratio between x and y units

r + coord_flip()
xlim, ylim
Flipped Cartesian coordinates

r + coord_polar(theta = "x", direction=1)
theta, start, direction
Polar coordinates

r + coord_trans(ytrans = "sqrt")
xtrans, ytrans, limx, limy
Transformed cartesian coordinates. Set xtrans and ytrans to the name of a window function.

π + coord_quickmap()
π + coord_map(projection = "ortho", orientation=c(41, -74, 0))projection, xlim, ylim
Map projections from the mapproj package (mercator (default), azequalarea, lagrange, etc.)

Position Adjustments

Position adjustments determine how to arrange geoms that would otherwise occupy the same space.

s <- ggplot(mpg, aes(fl, fill = drv))

s + geom_bar(position = "dodge")
Arrange elements side by side

s + geom_bar(position = "fill")
Stack elements on top of one another, normalize height

e + geom_point(position = "jitter")
Add random noise to X and Y position of each element to avoid overplotting

e + geom_label(position = "nudge")
Nudge labels away from points

s + geom_bar(position = "stack")
Stack elements on top of one another

Each position adjustment can be recast as a function with manual **width** and **height** arguments
s + geom_bar(position = position_dodge(width = 1))

Themes

r + theme_bw()
White background with grid lines

r + theme_gray()
Grey background (default theme)

r + theme_dark()
dark for contrast

r + theme_classic()
r + theme_light()

r + theme_linedraw()
r + theme_minimal()
Minimal themes

r + theme_void()
Empty theme

Faceting

Facets divide a plot into subplots based on the values of one or more discrete variables.

t <- ggplot(mpg, aes(cty, hwy)) + geom_point()

t + facet_grid(cols = vars(fl))
facet into columns based on fl

t + facet_grid(rows = vars(year))
facet into rows based on year

t + facet_grid(rows = vars(year), cols = vars(fl))
facet into both rows and columns

t + facet_wrap(vars(fl))
wrap facets into a rectangular layout

Set **scales** to let axis limits vary across facets

t + facet_grid(rows = vars(drv), cols = vars(fl), scales = "free")
x and y axis limits adjust to individual facets
"free_x" - x axis limits adjust
"free_y" - y axis limits adjust

Set **labeller** to adjust facet labels

t + facet_grid(cols = vars(fl), labeller = label_both)

fl: c	fl: d	fl: e	fl: p	fl: r

t + facet_grid(rows = vars(fl), labeller = label_bquote(alpha ^ .(fl)))

α^{c} α^{d} α^{e} α^{p} α^{r}

Labels

t + labs(**x** = "New x axis label", **y** = "New y axis label", **title** ="Add a title above the plot", **subtitle** = "Add a subtitle below title", **caption** = "Add a caption below plot", <AES> = "New <AES> legend title")

Use scale functions to update legend labels

t + annotate(geom = "text", x = 8, y = 9, label = "A")

geom to place | manual values for geom's aesthetics

Legends

n + theme(legend.position = "bottom")
Place legend at "bottom", "top", "left", or "right"

n + guides(fill = "none")
Set legend type for each aesthetic: colorbar, legend, or none (no legend)

n + scale_fill_discrete(name = "Title", labels = c("A", "B", "C", "D", "E"))
Set legend title and labels with a scale function.

Zooming

Without clipping (preferred)

t + coord_cartesian(
xlim = c(0, 100), ylim = c(10, 20))

With clipping (removes unseen data points)

t + xlim(0, 100) **+ ylim**(10, 20)

t + scale_x_continuous(limits = c(0, 100)) **+ scale_y_continuous**(limits = c(0, 100))

ggplot2

geom_sf allows to plot `sf` objects (i.e. make maps)

Let's work on a project

Grasshopper
Geography

Let's work on a project

Retreat of glaciers in North America

Grasshopper Geography

Data

For this workshop, we will use:

- the Alaska as well as the Western Canada & USA subsets of the Randolph Glacier Inventory version 6.0[1]
- the USGS time series of the named glaciers of Glacier National Park [2]
- the Alaska as well as the Western Canada & USA subsets of the consensus estimate for the ice thickness distribution of all glaciers on Earth dataset [3]

The datasets can be downloaded as zip files from these websites

1. RGI Consortium (2017). Randolph Glacier Inventory – A Dataset of Global Glacier Outlines: Version 6.0: Technical Report, Global Land Ice Measurements from Space, Colorado, USA. Digital Media. DOI: https://doi.org/10.7265/N5-RGI-60.
2. Fagre, D.B., McKeon, L.A., Dick, K.A. & Fountain, A.G., 2017, Glacier margin time series (1966, 1998, 2005, 2015) of the named glaciers of Glacier National Park, MT, USA: U.S. Geological Survey data release. DOI: https://doi.org/10.5066/F7P26WB1.
3. Farinotti, Daniel, 2019, A consensus estimate for the ice thickness distribution of all glaciers on Earth - dataset, Zurich. ETH Zurich. DOI: https://doi.org/10.3929/ethz-b-000315707.

Packages

Packages need to be installed before they can be loaded in a session

Packages on CRAN can be installed with:

```
install.packages("<package-name>")
```

`basemaps` is not on CRAN & needs to be installed from GitHub thanks to `devtools`:

```
install.packages("devtools")
devtools::install_github("16EAGLE/basemaps")
```

Packages

We load all the packages that we will need at the top of the script:

```
library(sf)                  # spatial vector data manipulation
library(tmap)                # map production & tiled web map
library(dplyr)               # non GIS specific (tabular data manipulation)
library(magrittr)            # non GIS specific (pipes)
library(purrr)               # non GIS specific (functional programming)
library(rnaturalearth)       # basemap data access functions
library(rnaturalearthdata)   # basemap data
library(mapview)             # tiled web map
library(grid)                # (part of base R) used to create inset map
library(ggplot2)             # alternative to tmap for map production
```

(Truncated code. View the full code at: https://westgrid-slides.netlify.app/r_gis_brc/#/49)


Reading & preparing data
Grasshopper
Geography

Randolph Glacier Inventory

This dataset contains the contour of all glaciers on Earth

We will focus on glaciers in Western North America

You can download & unzip `02_rgi60_WesternCanadaUS` & `01_rgi60_Alaska`
from the Randolph Glacier Inventory version 6.0

Reading in data

Data get imported & turned into `sf` objects with the function `sf::st_read`:

```
ak <- st_read("data/01_rgi60_Alaska")
```

> Make sure to use the absolute paths or the paths relative to your working directory (which can be obtained with `getwd`)

Reading in data

```
ak <- st_read("data/01_rgi60_Alaska")
```

> Output

```
Reading layer `01_rgi60_Alaska' from data source `./data/01_rgi60_Alaska'
              using driver `ESRI Shapefile'
Simple feature collection with 27108 features and 22 fields
Geometry type: POLYGON
Dimension:     XY
Bounding box:  xmin: -176.1425 ymin: 52.05727 xmax: -126.8545 ymax: 69.35167
Geodetic CRS:  WGS 84
```

Reading in data

Your turn:

Read in the data for the rest of north western America (from `02_rgi60_WesternCanadaUS`) and create an sf object called `wes`

First look at the data

```
ak
```

> Output (truncated. View the full output at: https://westgrid-slides.netlify.app/r_gis_brc/#/55)

```
Simple feature collection with 27108 features and 22 fields
Geometry type: POLYGON
Dimension:     XY
Bounding box:  xmin: -176.1425 ymin: 52.05727 xmax: -126.8545 ymax: 69.35167
Geodetic CRS:  WGS 84
First 10 features:
           RGIId        GLIMSId  BgnDate  EndDate    CenLon   CenLat O1Region
1  RGI60-01.00001 G213177E63689N 20090703 -9999999 -146.8230 63.68900        1
2  RGI60-01.00002 G213332E63404N 20090703 -9999999 -146.6680 63.40400        1
3  RGI60-01.00003 G213920E63376N 20090703 -9999999 -146.0800 63.37600        1
```

Structure of the data

```
str(ak)
```

> Output (truncated. View the full output at: https://westgrid-slides.netlify.app/r_gis_brc/#/56)

```
Classes ‘sf’ and 'data.frame':  27108 obs. of  23 variables:
$ RGIId   : chr  "RGI60-01.00001" "RGI60-01.00002" "RGI60-01.00003" ...
$ GLIMSId : chr  "G213177E63689N" "G213332E63404N" "G213920E63376N" ...
$ BgnDate : chr  "20090703" "20090703" "20090703" "20090703" ...
$ EndDate : chr  "-9999999" "-9999999" "-9999999" "-9999999" ...
$ CenLon  : num  -147 -147 -146 -146 -147 ...
$ CenLat  : num  63.7 63.4 63.4 63.4 63.6 ...
$ O1Region: chr  "1" "1" "1" "1" ...
$ O2Region: chr  "2" "2" "2" "2" ...
$ Area    : num  0.36 0.558 1.685 3.681 2.573 ...
```

Inspect your data

Your turn:

Inspect the `wes` object you created

Glacier National Park

This dataset contains a time series of the retreat of 39 glaciers of Glacier National Park, MT, USA for the years `1966`, `1998`, `2005` & `2015`

You can download and unzip the 4 sets of files from the USGS website

Read in and clean datasets

```
## create a function that reads and cleans the data
prep <- function(dir) {
  g <- st_read(dir)
  g %<>% rename_with(~ tolower(gsub("Area....", "area", .x)))
  g %<>% dplyr::select(
    year,
    objectid,
    glacname,
    area,
    shape_leng,
```

(Truncated code. View the full code at: https://westgrid-slides.netlify.app/r_gis_brc/#/59)

> We use `dplyr::select` because **terra** also has a `select` function

Combine datasets into one sf object

Check that the CRS are all the same:

```
all(sapply(
  list(st_crs(gnp[[1]]),
       st_crs(gnp[[2]]),
       st_crs(gnp[[3]]),
       st_crs(gnp[[4]])),
  function(x) x == st_crs(gnp[[1]])
))
```

> Output

```
[1] TRUE
```

Combine datasets into one sf object

We can `rbind` the elements of our list:

```
gnp <- do.call("rbind", gnp)
```

You can inspect your new sf object by calling it or with `str`

Estimate for ice thickness

This dataset contains an estimate for the ice thickness of all glaciers on Earth

The nomenclature follows the Randolph Glacier Inventory

Ice thickness being a spatial field, this is raster data

We will use data in `RGI60-02.16664_thickness.tif` from the ETH Zürich Research Collection
which corresponds to one of the glaciers (Agassiz) of Glacier National Park

Load raster data

Read in data and create a SpatRaster object:

```
ras <- rast("data/RGI60-02/RGI60-02.16664_thickness.tif")
```

Inspect our SpatRaster object

```
ras
```

> Output

```
class       : SpatRaster
dimensions  : 93, 74, 1  (nrow, ncol, nlyr)
resolution  : 25, 25  (x, y)
extent      : 707362.5, 709212.5, 5422962, 5425288  (xmin, xmax, ymin, ymax)
coord. ref. : +proj=utm +zone=11 +datum=WGS84 +units=m +no_defs
source      : RGI60-02.16664_thickness.tif
name        : RGI60-02.16664_thickness
```

`nlyr` gives us the number of bands (a single one here). You can also run `str(ras)`

Our data

We now have 3 sf objects & 1 SpatRaster object:

- `ak`: contour of glaciers in AK
- `wes`: contour of glaciers in the rest of Western North America
- `gnp`: time series of 39 glaciers in Glacier National Park, MT, USA
- `ras`: ice thickness of the Agassiz Glacier from Glacier National Park

Grasshopper
Geography

Let's map our sf object ak

At a bare minimum, we need `tm_shape` with the data & some info as to how to represent that data:

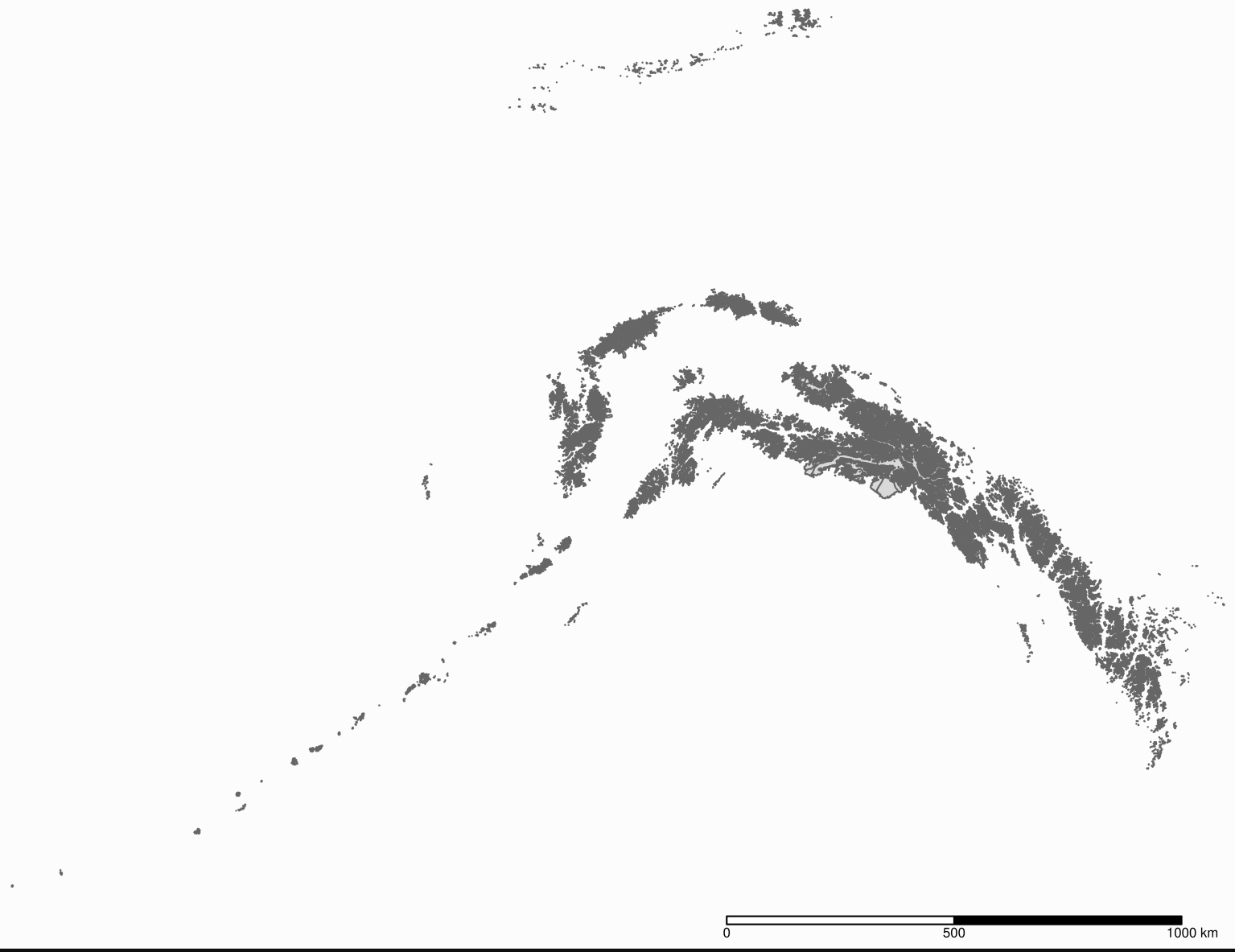
```
tm_shape(ak) +
  tm_polygons()
```

We need to label & customize it

```
tm_shape(ak) +
  tm_polygons() +
  tm_layout(
    title = "Glaciers of Alaska",
    title.position = c("center", "top"),
    title.size = 1.1,
    bg.color = "#fcfcfc",
    inner.margins = c(0.06, 0.01, 0.09, 0.01),
    outer.margins = 0,
    frame.lwd = 0.2
```

(Truncated code. View the full code at: https://westgrid-slides.netlify.app/r_gis_brc/#/69)

Glaciers of Alaska



Make a map of the wes object

Your turn:

Make a map with the `wes` object you created with the data for Western North America excluding AK


Glaciers of Western North America excluding AK
N
0
500
1000 km

Now, let's make a map with ak & wes

The Coordinate Reference Systems (CRS) must be the same

sf has a function to retrieve the CRS of an sf object: `st_crs`

```
st_crs(ak) == st_crs(wes)
```

> Output

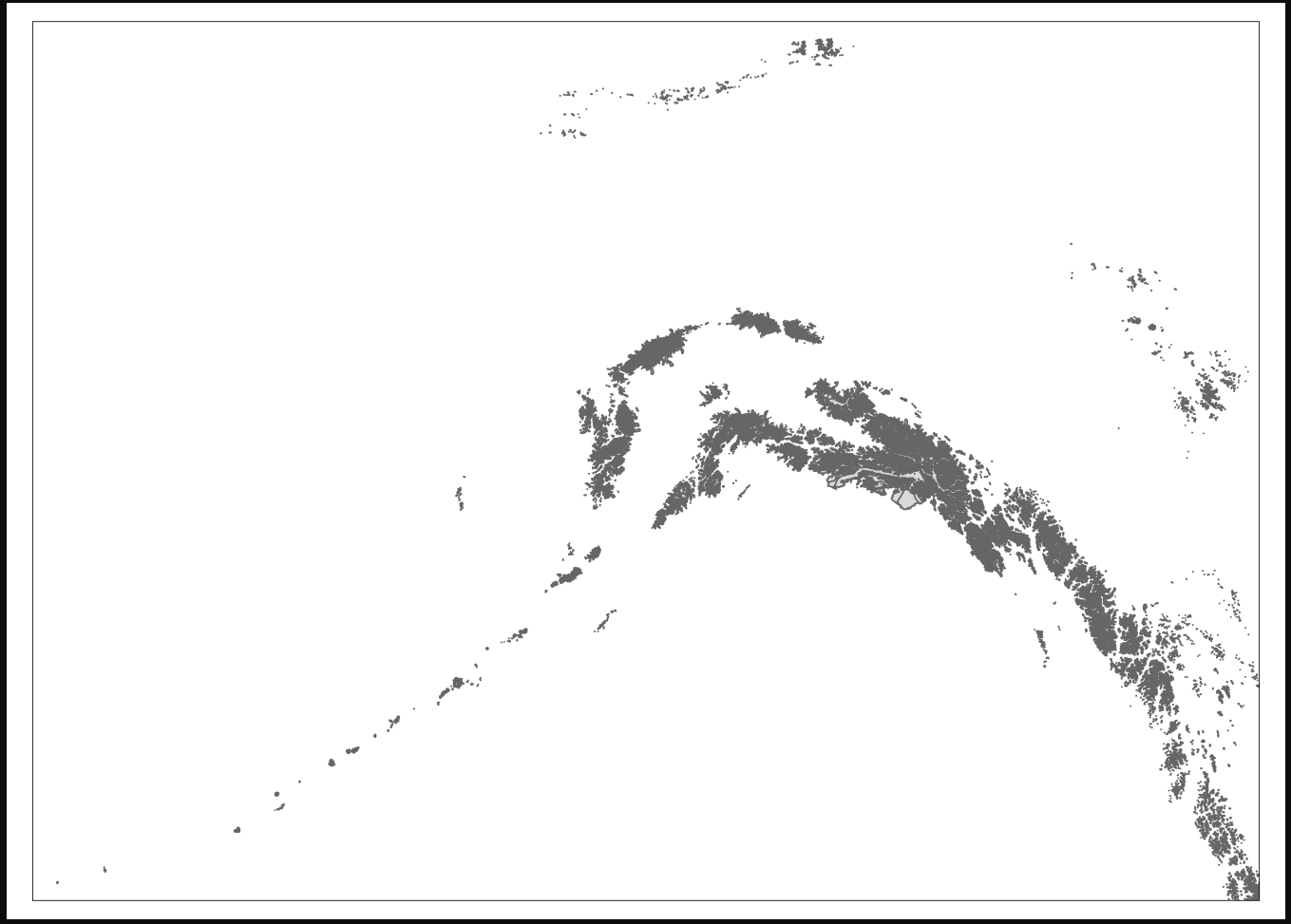
```
[1] TRUE
```

So we're good (we will see later what to do if this is not the case)

Our combined map

Let's start again with a minimum map without any layout to test things out:

```
tm_shape(ak) +
  tm_polygons() +
  tm_shape(wes) +
  tm_polygons()
```



Uh ... oh ...

What went wrong?

Maps are bound by “bounding boxes”. In tmap, they are called `bbox`

tmap sets the bbox the first time `tm_shape` is called. In our case, the bbox was thus set to the bbox of the ak object

We need to create a new bbox for our new map

Retrieving bounding boxes

sf has a function to retrieve the bbox of an sf object: `st_bbox`

The bbox of ak is:

```
st_bbox(ak)
```

> Output

```
xmin         ymin       xmax         ymax
-176.14247   52.05727   -126.85450   69.35167
```

Combining bounding boxes

bbox objects can't be combined directly

Here is how we can create a new bbox encompassing both of our bboxes:

- First, we transform our bboxes to sfc objects with `st_as_sfc`
- Then we combine those objects into a new sfc object with `st_union`
- Finally, we retrieve the bbox of that object with `st_bbox`:

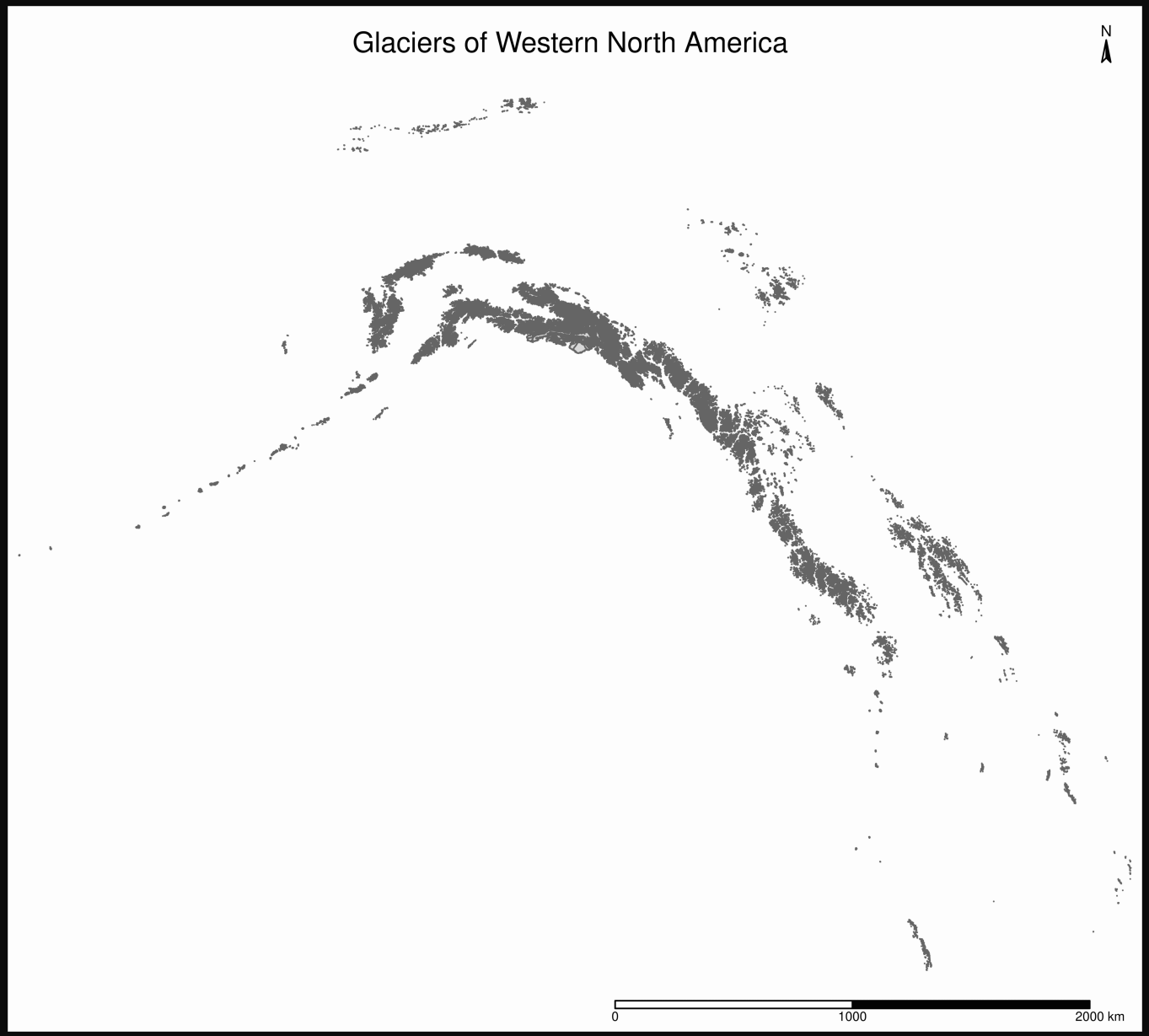
```
nwa_bbox <- st_bbox(
st_union(
st_as_sfc(st_bbox(wes)),
st_as_sfc(st_bbox(ak))
)
)
```

Back to our map

We can now use our new bounding box for the map of Western North America:

```
tm_shape(ak, bbox = nwa_bbox) +
  tm_polygons() +
  tm_shape(wes) +
  tm_polygons() +
  tm_layout(
    title = "Glaciers of Western North America",
    title.position = c("center", "top"),
    title.size = 1.1,
    bg.color = "#fcfcfc",
    inner.margins = c(0.06, 0.01, 0.09, 0.01),
```

(Truncated code. View the full code at: https://westgrid-slides.netlify.app/r_gis_brc/#/79)


Glaciers of Western North America
N
0
1000
2000 km

Let's add a basemap

We will use data from Natural Earth , a public domain map dataset

There are much more fancy options, but they usually involve creating accounts (e.g. with Google) to access some API

In addition, this dataset can be accessed direction from within R thanks to the rOpenSci packages:

- `rnaturalearth`: provides the functions
- `rnaturalearthdata`: provides the data

Create an sf object with states/provinces

```
states_all <- ne_states(
  country = c("canada", "united states of america"),
  returnclass = "sf"
)
```

> `ne_` stands for "Natural Earth"

Select relevant states/provinces

```
states <- states_all %>%
  filter(name_en == "Alaska" |
           name_en == "British Columbia" |
           name_en == "Yukon" |
           name_en == "Northwest Territories" |
           name_en ==  "Alberta" |
           name_en == "California" |
           name_en == "Washington" |
           name_en == "Oregon" |
           name_en == "Idaho" |
```

(Truncated code. View the full code at: https://westgrid-slides.netlify.app/r_gis_brc/#/83)

Add the basemap to our map

> What do we need to make sure of first?

```
st_crs(states) == st_crs(ak)
```

> Output

```
[1] TRUE
```

Add the basemap to our map

We add the basemap as a 3rd layer

Mind the order! If you put the basemap last, it will cover your data

Of course, we will use our `nwa_bbox` bounding box again

We will also break `tm_polygons` into `tm_borders` and `tm_fill` for ak and wes in order to colourise them with slightly different colours

Add the basemap to our map

```
tm_shape(states, bbox = nwa_bbox) +
  tm_polygons(col = "#f2f2f2", lwd = 0.2) +
  tm_shape(ak) +
  tm_borders(col = "#3399ff") +
  tm_fill(col = "#86baff") +
  tm_shape(wes) +
  tm_borders(col = "#3399ff") +
  tm_fill(col = "#86baff") +
  tm_layout(
    title = "Glaciers of Western North America",
```

(Truncated code. View the full code at: https://westgrid-slides.netlify.app/r_gis_brc/#/86)


Glaciers of Western North America
N
0
1000
2000 km

tmap styles

tmap has a number of styles that you can try

For instance, to set the style to “classic”, run the following before making your map:

```
tmap_style("classic")
```

> Other options are:
> "white" (default), "gray", "natural", "cobalt", "col_blind", "albatross", "beaver", "bw", "watercolor"


Glaciers of Western North America
N
0
1000
2000 km

tmap styles

To return to the default, you need to run

```
tmap_style("white")
```

or

```
tmap_options_reset()
```

which will reset every **tmap** option

Inset maps

Now, how can we combine this with our `gnp` object?

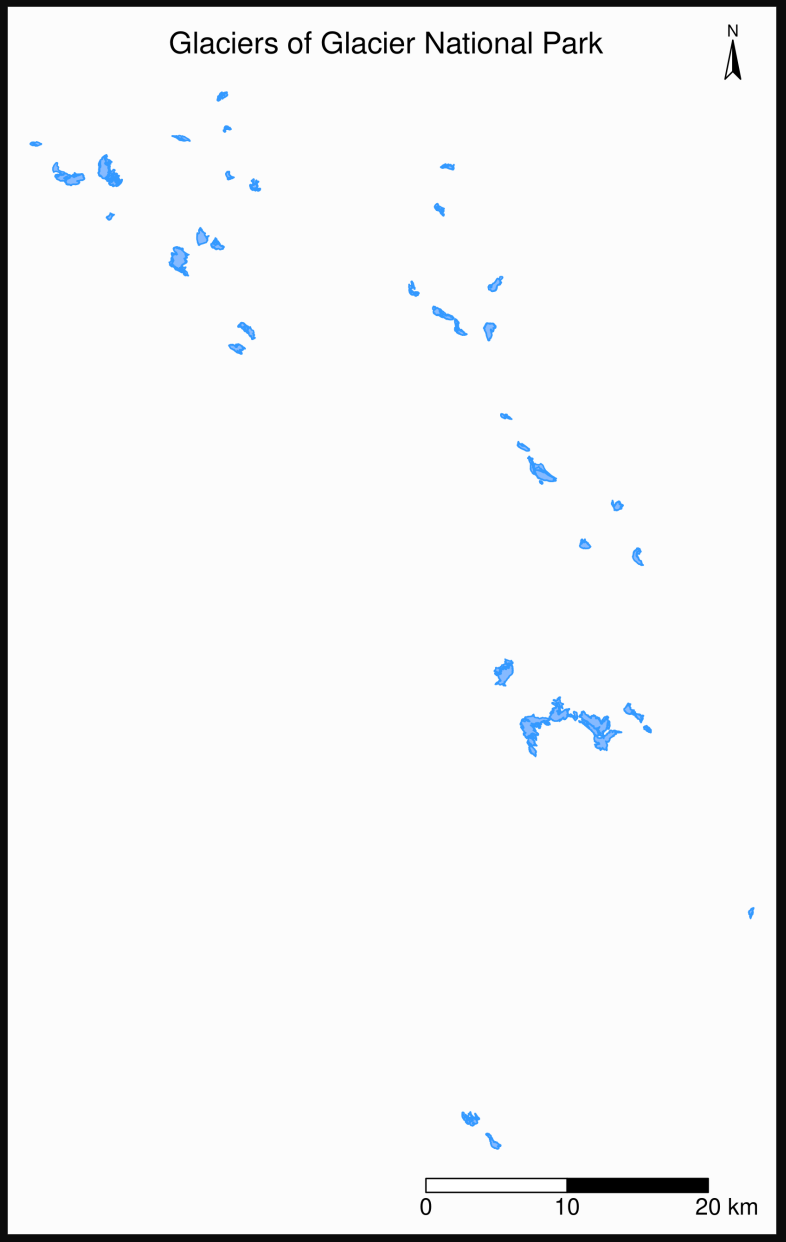
We could add it as an inset of our Western North America map

First, let's map it

Let's use the same `tm_borders` and `tm_fill` we just used:

```
tm_shape(gnp) +
  tm_borders(col = "#3399ff") +
  tm_fill(col = "#86baff") +
  tm_layout(
    title = "Glaciers of Glacier National Park",
    title.position = c("center", "top"),
    legend.title.color = "#fcfcfc",
    legend.text.size = 1,
    bg.color = "#fcfcfc",
    inner.margins = c(0.07, 0.03, 0.07, 0.03),
```

(Truncated code. View the full code at: https://westgrid-slides.netlify.app/r_gis_brc/#/92)


Glaciers of Glacier National Park
N
0
10
20 km

Create an inset map

As always, first we check that the CRS are the same:

```
st_crs(gnp) == st_crs(ak)
```

> Output

```
[1] FALSE
```

Create an inset map

As always, first we check that the CRS are the same:

```
st_crs(gnp) == st_crs(ak)
```

> Output

```
[1] FALSE
```

AH!

CRS transformation

We need to reproject `gnp` into the CRS of our other sf objects (e.g. `ak`):

```
gnp <- st_transform(gnp, st_crs(ak))
```

We can verify that the CRS are now the same:

```
st_crs(gnp) == st_crs(ak)
```

> Output

```
[1] TRUE
```

Inset maps

First step:

Add a rectangle showing the location of the GNP map in the main North America map

We need to create a new sfc object from the `gnp` bbox so that we can add it to our previous map as a new layer:

```
gnp_zone <- st_bbox(gnp) %>%
  st_as_sfc()
```

Inset maps

Second step:

Create a tmap object of the main map

Of course, we need to edit the title. Also, note the presence of our new layer:

```
main_map <- tm_shape(states, bbox = nwa_bbox) +
  tm_polygons(col = "#f2f2f2", lwd = 0.2) +
  tm_shape(ak) +
  tm_borders(col = "#3399ff") +
  tm_fill(col = "#86baff") +
  tm_shape(wes) +
  tm_borders(col = "#3399ff") +
  tm_fill(col = "#86baff") +
  tm_shape(gnp_zone) +
  tm_borders(lwd = 1.5, col = "#ff9900") +
```

(Truncated code. View the full code at: https://westgrid-slides.netlify.app/r_gis_brc/#/97)

Inset maps

Third step:

Create a tmap object of the inset map

We make sure to matching colours & edit the layouts for better readability:

```
inset_map <- tm_shape(gnp) +
  tm_borders(col = "#3399ff") +
  tm_fill(col = "#86baff") +
  tm_layout(
    legend.show = F,
    bg.color = "#fcfcfc",
    inner.margins = c(0.03, 0.03, 0.03, 0.03),
    outer.margins = 0,
    frame = "#ff9900",
    frame.lwd = 3
```

(Truncated code. View the full code at: https://westgrid-slides.netlify.app/r_gis_brc/#/98)

Inset maps

Final step:

Combine the two tmap objects

We print the main map & add the inset map with `grid::viewport`:

```
main_map
print(inset_map, vp = viewport(0.41, 0.26, width = 0.5, height = 0.5))
```


Glaciers of Glacier National Park
N
0
500
1000 km

Mapping a subset of the data

Mapping a subset of the data

To see the retreat of the ice, we need to zoom in

Let’s focus on a single glacier: Agassiz Glacier

Map of the Agassiz Glacier

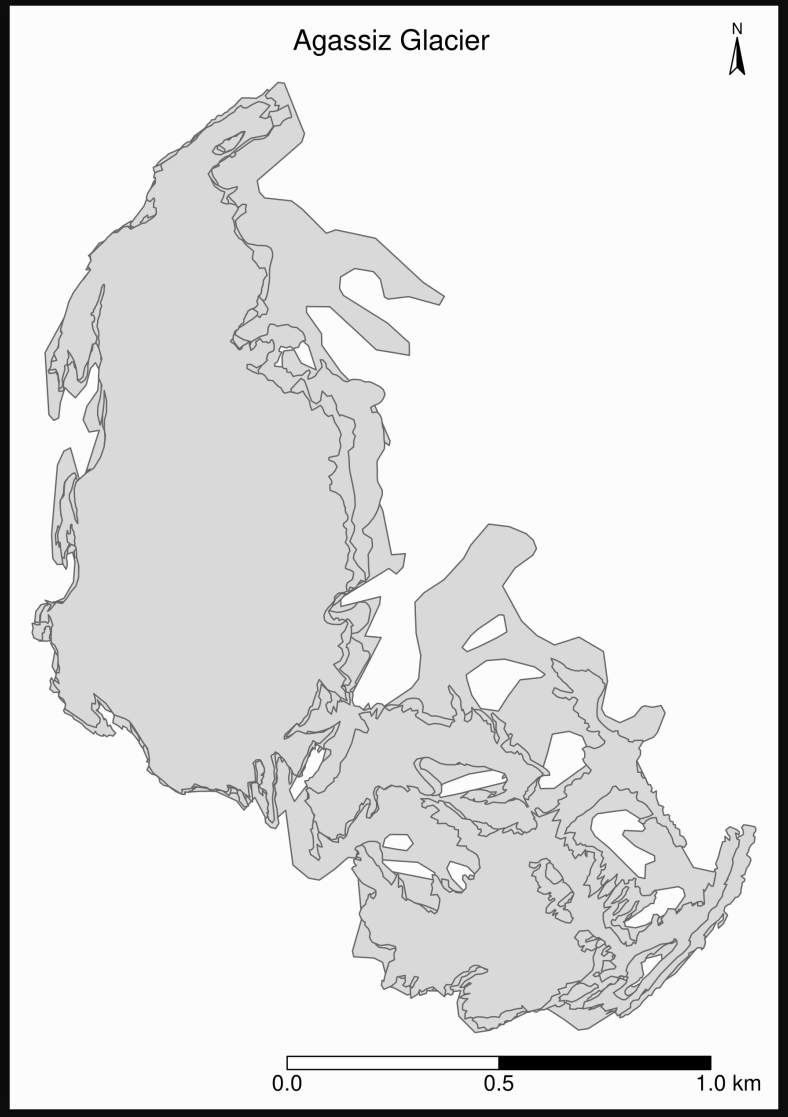
Select the data points corresponding to the Agassiz Glacier:

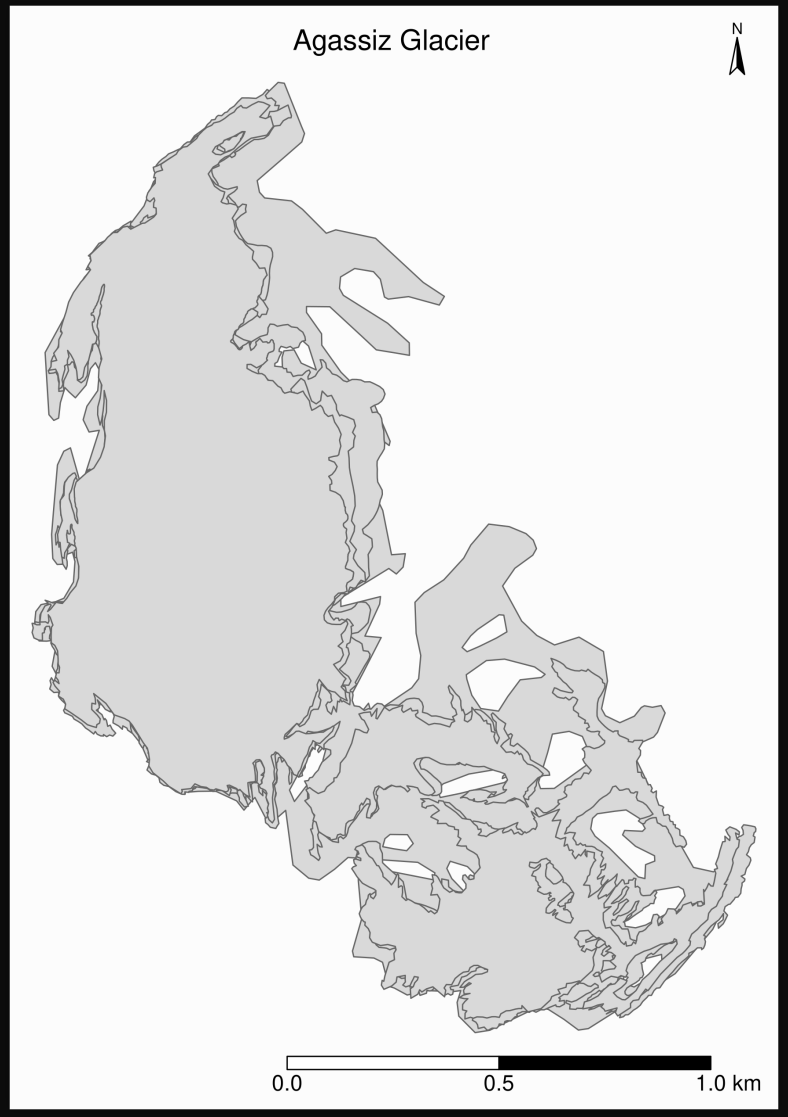
```
ag <- gnp %>% filter(glacname == "Agassiz Glacier")
```

Map of the Agassiz Glacier

```
tm_shape(ag) +
  tm_polygons() +
  tm_layout(
    title = "Agassiz Glacier",
    title.position = c("center", "top"),
    legend.position = c("left", "bottom"),
    legend.title.color = "#fcfcfc",
    legend.text.size = 1,
    bg.color = "#fcfcfc",
    inner.margins = c(0.07, 0.03, 0.07, 0.03),
```

(Truncated code. View the full code at: https://westgrid-slides.netlify.app/r_gis_brc/#/103)


Agassiz Glacier
N
0.0
0.5
1.0 km



Not great ...

Map based on attribute variables

```
tm_shape(ag) +
  tm_polygons("year", palette = "Blues") +
  tm_layout(
    title = "Agassiz Glacier",
    title.position = c("center", "top"),
    legend.position = c("left", "bottom"),
    legend.title.color = "#fcfcfc",
    legend.text.size = 1,
    bg.color = "#fcfcfc",
    inner.margins = c(0.07, 0.03, 0.07, 0.03),
```

(Truncated code. View the full code at: https://westgrid-slides.netlify.app/r_gis_brc/#/105)

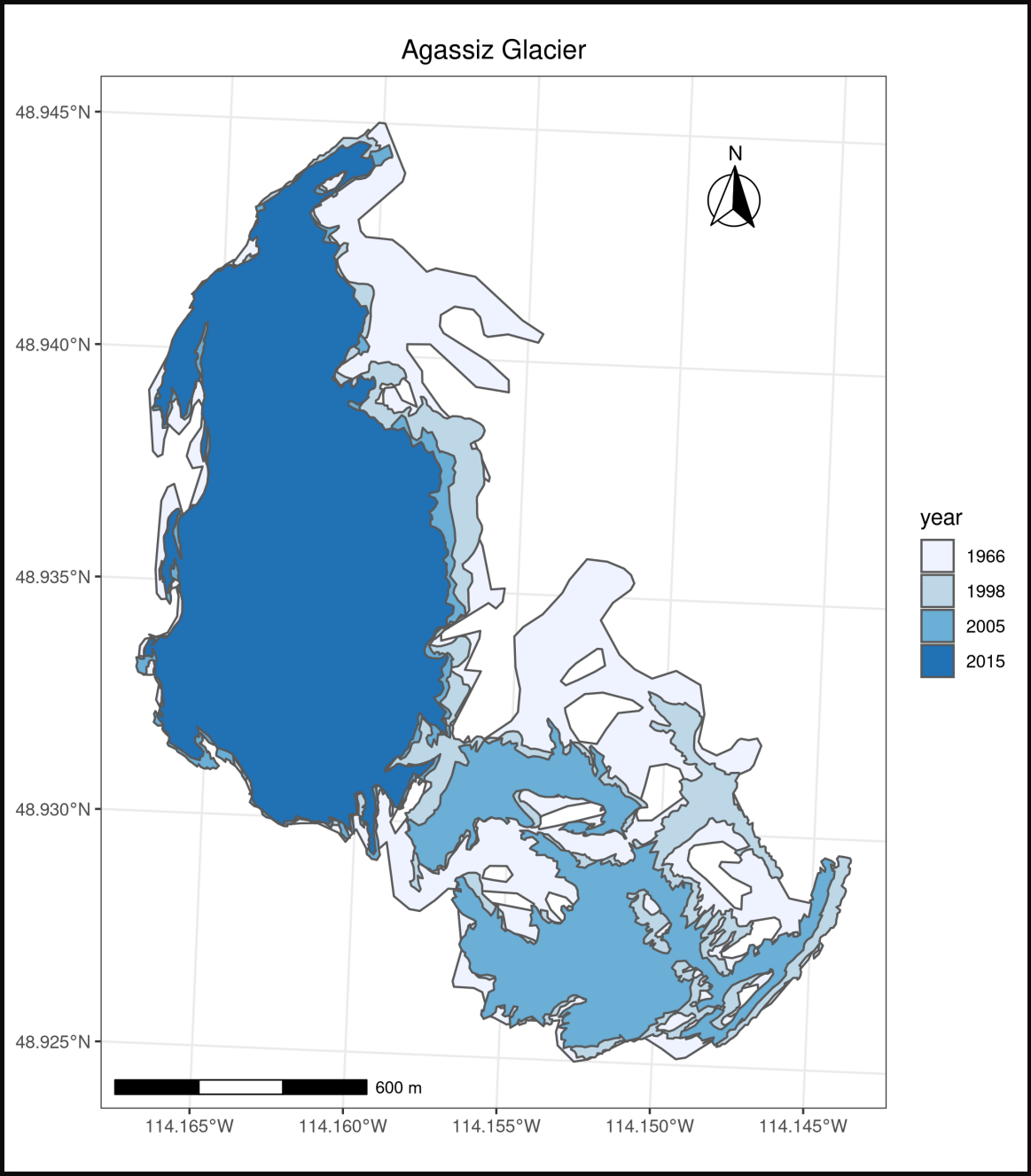

Agassiz Glacier
N
1966
1998
2005
2015
0.0
0.5
1.0 km

Using ggplot2 instead of tmap

As an alternative to **tmap**, **ggplot2** can plot maps with the `geom_sf` function:

```
ggplot(ag) +
  geom_sf(aes(fill = year)) +
  scale_fill_brewer(palette = "Blues") +
  labs(title = "Agassiz Glacier") +
  annotation_scale(location = "bl", width_hint = 0.4) +
  annotation_north_arrow(location = "tr", which_north = "true",
                         pad_x = unit(0.75, "in"), pad_y = unit(0.5, "in"),
                         style = north_arrow_fancy_orienteering) +
  theme_bw() +
  theme(plot.title = element_text(hjust = 0.5))
```

The package **ggspatial** adds a lot of functionality to **ggplot2** for spatial data


Agassiz Glacier
48.945°N
48.940°N
48.935°N
48.930°N
48.925°N
N
600 m
114.165°W
114.160°W
114.155°W
114.150°W
114.145°W
year
1966
1998
2005
2015

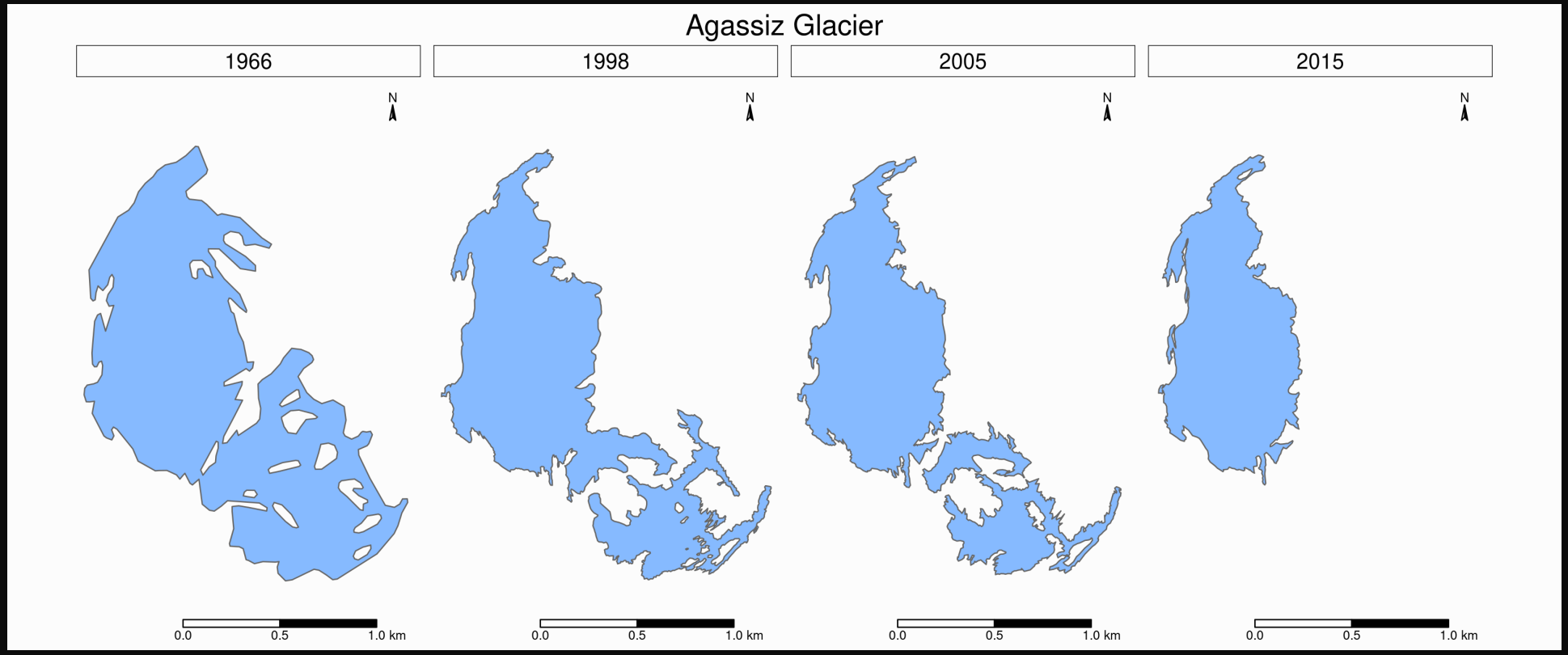

Lunch break
Grasshopper
Geography

Faceted maps

Faceted map of the retreat of Agassiz

```
tm_shape(ag) +
  tm_polygons(col = "#86baff") +
  tm_layout(
    main.title = "Agassiz Glacier",
    main.title.position = c("center", "top"),
    main.title.size = 1.2,
    legend.position = c("left", "bottom"),
    legend.title.color = "#fcfcfc",
    legend.text.size = 1,
    bg.color = "#fcfcfc",
```

(Truncated code. View the full code at: https://westgrid-slides.netlify.app/r_gis_brc/#/111)


Agassiz Glacier
1966
1998
2005
2015
N
0.0
0.5
1.0 km

Animated maps

Animated map of the Retreat of Agassiz

First, we need to create a tmap object with facets:

```
agassiz_anim <- tm_shape(ag) +
  tm_polygons(col = "#86baff") +
  tm_layout(
    title = "Agassiz Glacier",
    title.position = c("center", "top"),
    legend.position = c("left", "bottom"),
    legend.title.color = "#fcfcfc",
    legend.text.size = 1,
    bg.color = "#fcfcfc",
    inner.margins = c(0.08, 0, 0.08, 0),
```

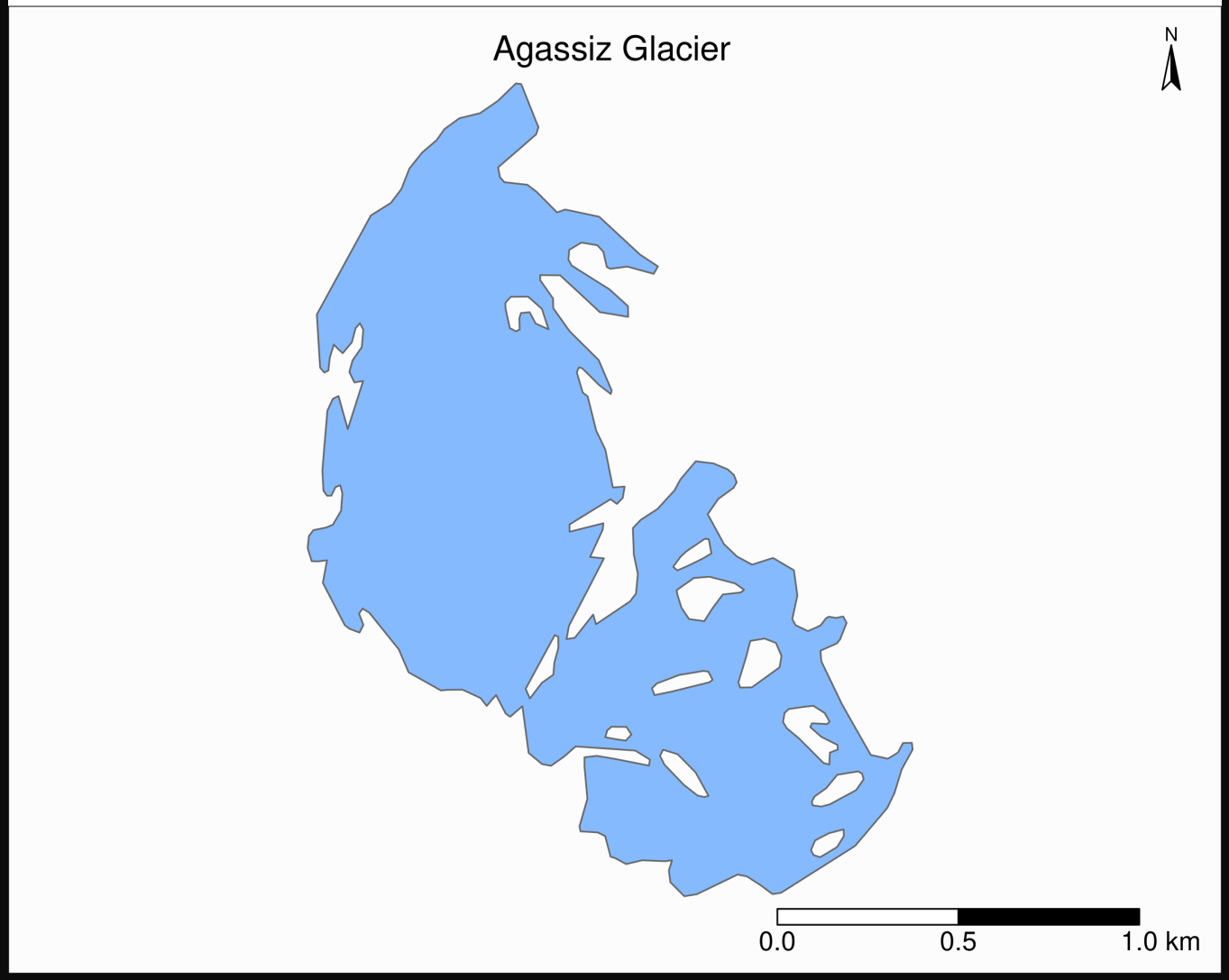
(Truncated code. View the full code at: https://westgrid-slides.netlify.app/r_gis_brc/#/114)

Animated map of the Retreat of Agassiz

Then we can pass that object to `tmap_animation`:

```
tmap_animation(
  agassiz_anim,
  filename = "ag.gif",
  dpi = 300,
  inner.margins = c(0.08, 0, 0.08, 0),
  delay = 100
)
```

1966



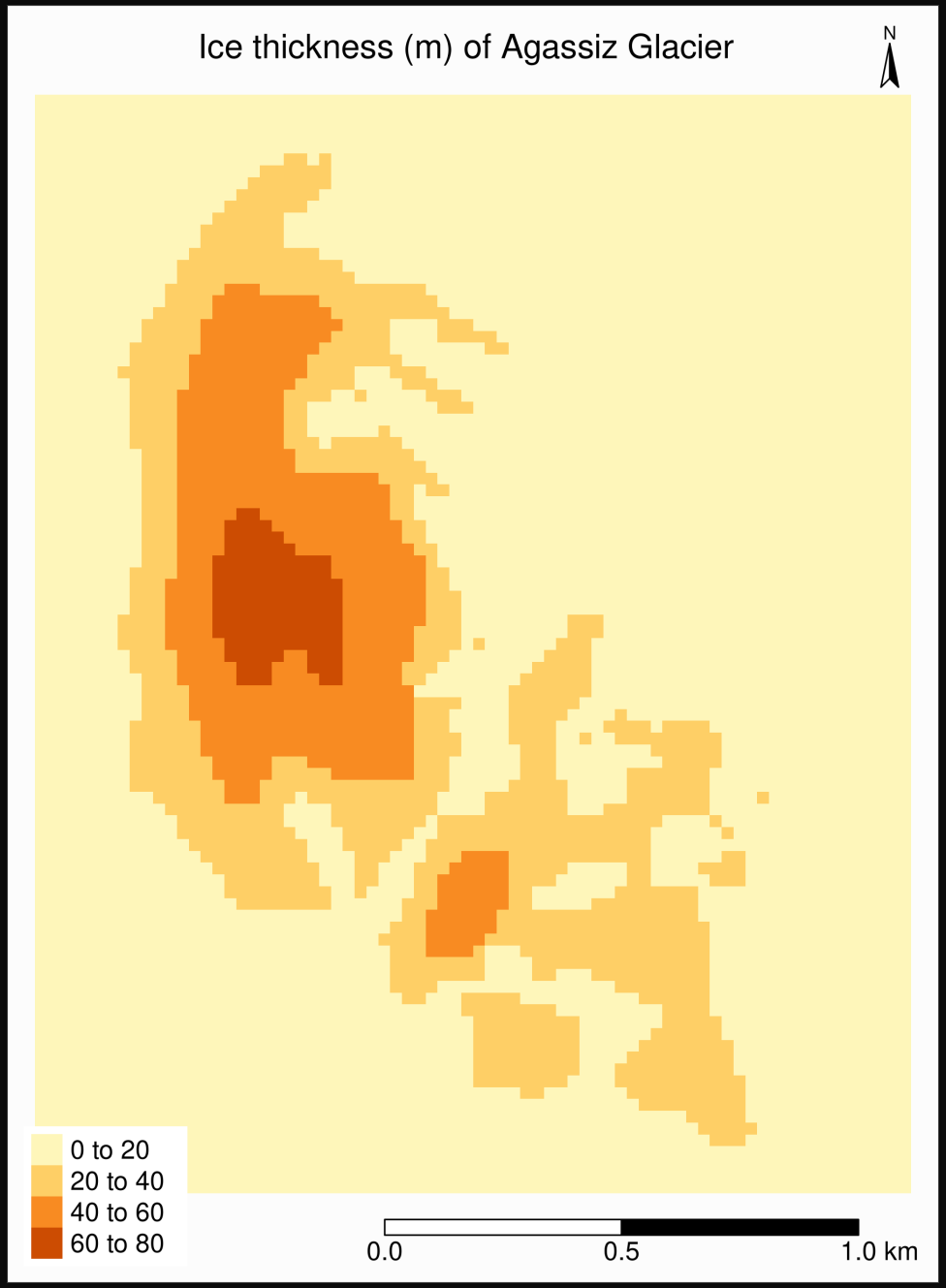
Map of ice thickness of Agassiz

Now, let's map the estimated ice thickness on Agassiz Glacier

This time, we use `tm_raster`:

```
tm_shape(ras) +
  tm_raster(title = "") +
  tm_layout(
    title = "Ice thickness (m) of Agassiz Glacier",
    title.position = c("center", "top"),
    legend.position = c("left", "bottom"),
    legend.bg.color = "#ffffff",
    legend.text.size = 1,
    bg.color = "#fcfcfc",
    inner.margins = c(0.07, 0.03, 0.07, 0.03),
```

(Truncated code. View the full code at: https://westgrid-slides.netlify.app/r_gis_brc/#/117)


Ice thickness (m) of Agassiz Glacier
N
0 to 20
20 to 40
40 to 60
60 to 80
0.0
0.5
1.0 km

Combining with Randolph data

As always, we check whether the CRS are the same:

```
st_crs(ag) == st_crs(ras)
```

> Output

```
[1] FALSE
```

We need to reproject `ag` (remember that it is best to avoid reprojecting raster data):

```
ag %<>% st_transform(st_crs(ras))
```

Combining with Randolph data

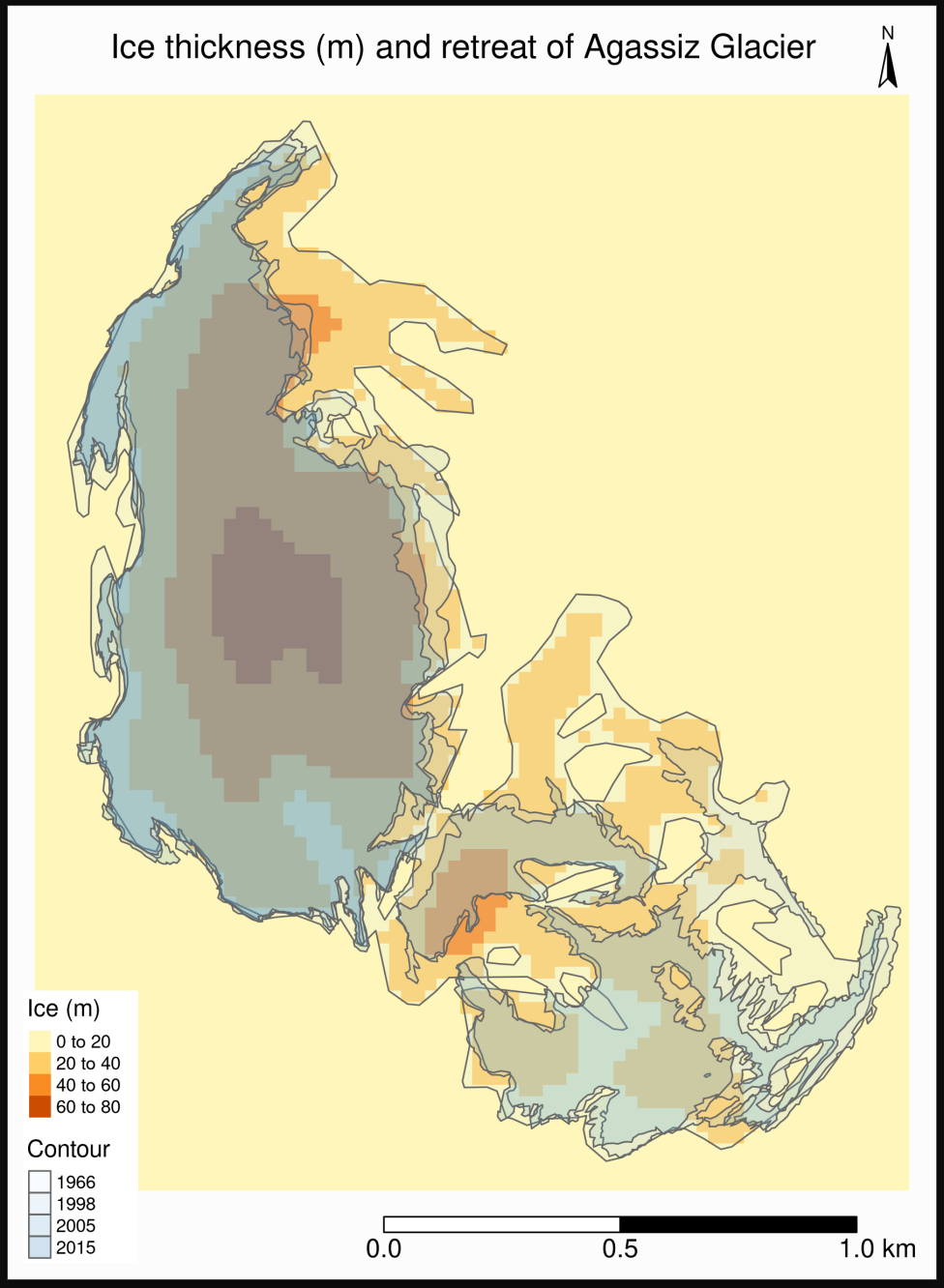
The retreat & ice thickness layers will hide each other (the order matters!)

One option is to use `tm_borders` for one of them, but we can also use transparency (alpha)

We also adjust the legend:

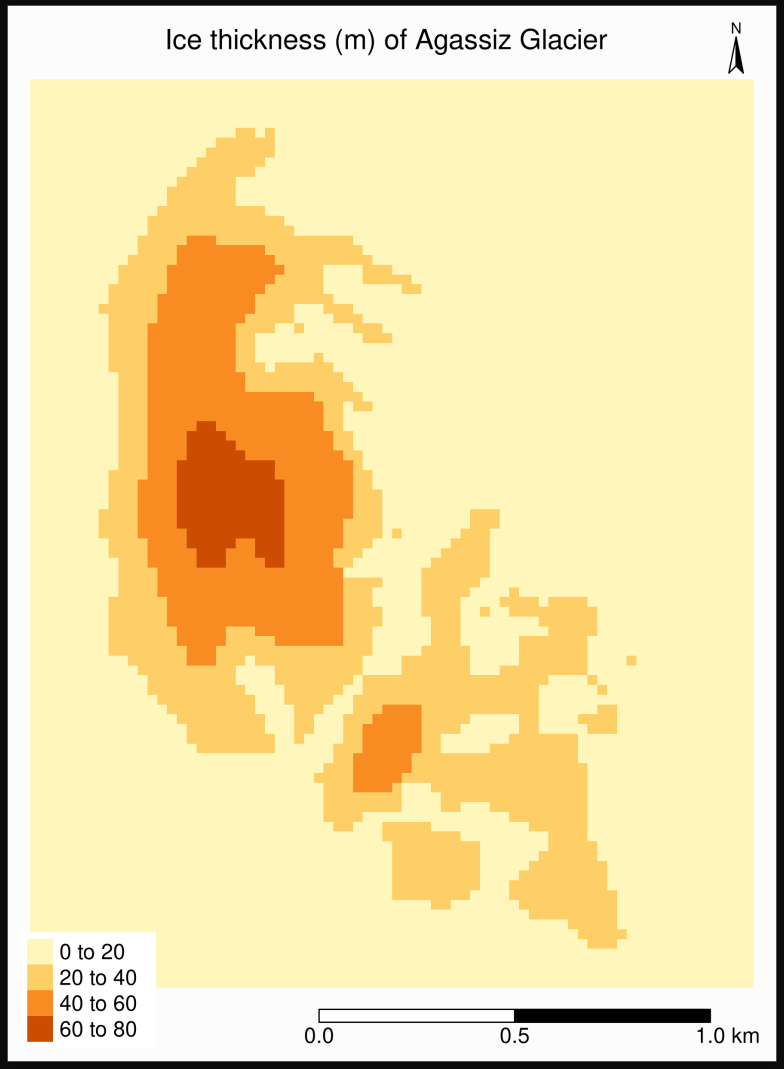
```
tm_shape(ras) +
  tm_raster(title = "Ice (m)") +
  tm_shape(ag) +
  tm_polygons("year", palette = "Blues", alpha = 0.2, title = "Contour") +
  tm_layout(
    title = "Ice thickness (m) and retreat of Agassiz Glacier",
    title.position = c("center", "top"),
    legend.position = c("left", "bottom"),
    legend.bg.color = "#ffffff",
    legend.text.size = 0.7,
```

(Truncated code. View the full code at: https://westgrid-slides.netlify.app/r_gis_brc/#/120)

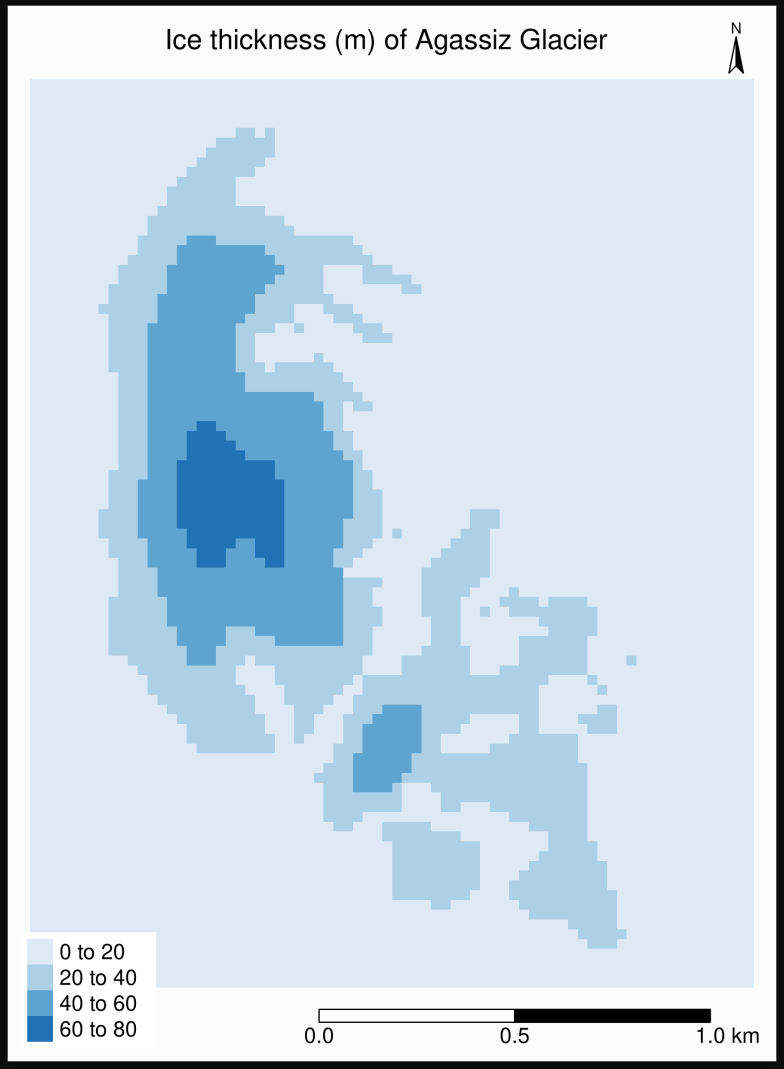

Ice thickness (m) and retreat of Agassiz Glacier
N
Ice (m)
0 to 20
20 to 40
40 to 60
60 to 80
Contour
1966
1998
2005
2015
0.0
0.5
1.0 km

Refining raster maps

Let’s go back to our ice thickness map:



We can change the palette to blue with `tm_raster(palette = "Blues")`:



We can create a more suitable interval scale:

First, let's see what the maximum value is:

```
global(ras, "max")
```

> Output

```
max
RGI60-02.16664_thickness 70.10873
```

Then we can set the breaks with `tm_raster(breaks = seq(0, 80, 5))`

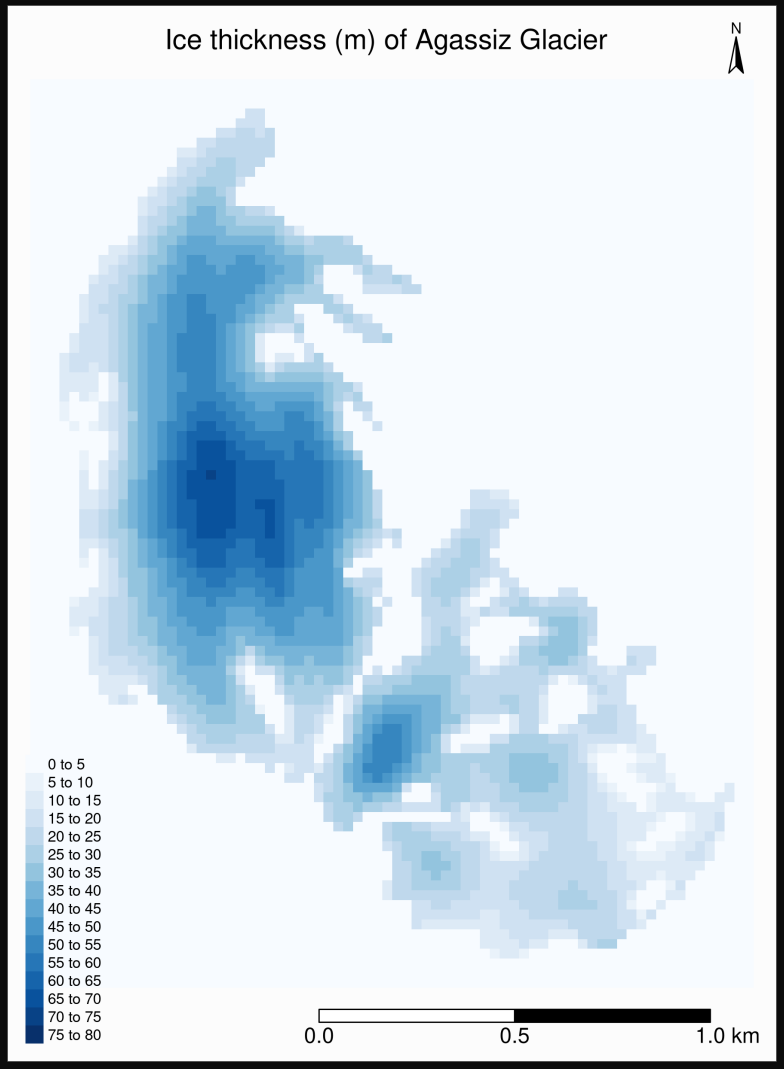
We can create a more suitable interval scale:

We also need to tweak the layout, legend, etc.:

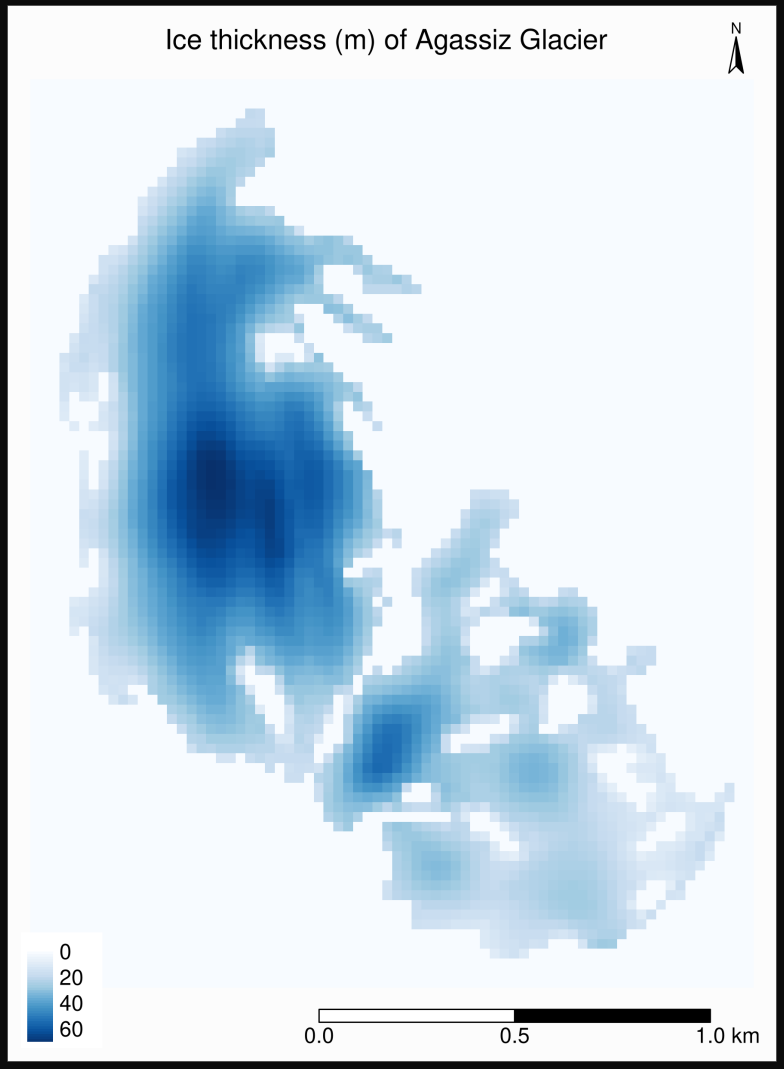
```
tm_shape(ras) +
  tm_raster(title = "", palette = "Blues", breaks = seq(0, 80, 5)) +
  tm_layout(
    title = "Ice thickness (m) of Agassiz Glacier",
    title.position = c("center", "top"),
    legend.position = c("left", "bottom"),
    legend.bg.color = "#ffffff",
    legend.text.size = 0.7,
    bg.color = "#fcfcfc",
    inner.margins = c(0.07, 0.03, 0.07, 0.03),
```

(Truncated code. View the full code at: https://westgrid-slides.netlify.app/r_gis_brc/#/126)

We can create a more suitable interval scale:



Or we can use a continuous colour scheme with `tm_raster(style = "cont")`:



Other ways to add a basemap

Basemap with ggmap

```
basemap <- get_map(
  bbox = c(
    left = st_bbox(ag)[1],
    bottom = st_bbox(ag)[2],
    right = st_bbox(ag)[3],
    top = st_bbox(ag)[4]
  ),
  source = "osm"
)
```

> **ggmap** is a powerful package, but Google now requires an API key obtained through registration

Basemap with basemaps

The package **basemaps** allows to download open source basemap data from several sources, but those cannot easily be combined with `sf` objects

This plots a satellite image of the Agassiz Glacier:

```
basemap_plot(ag, map_service = "esri", map_type = "world_imagery")
```

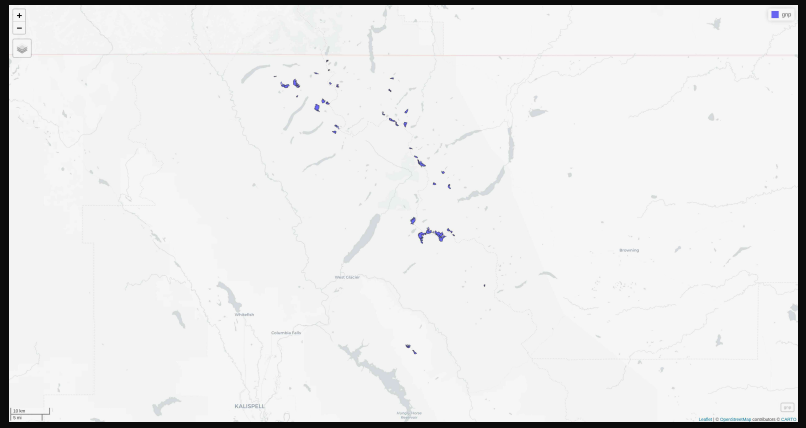
Satellite image of the Agassiz Glacier

Tiled web maps with Leaflet JS

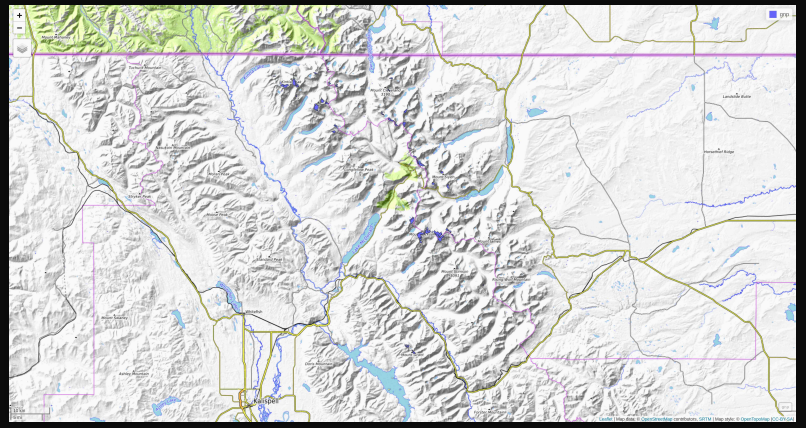
mapview

```
mapview(gnp)
```

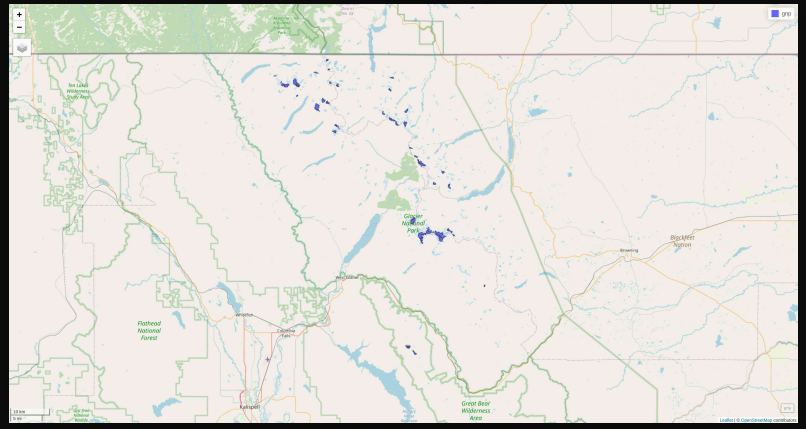
mapview



CartoDB.Positron



OpenTopoMap



OpenStreetMap

Esri.WorldImagery

tmap

So far, we have used the `plot` mode of **tmap**. There is also a `view` mode which allows interactive viewing in a browser through Leaflet

Change to `view` mode:

```
tmap_mode("view")
```

> you can also toggle between modes with `ttm`

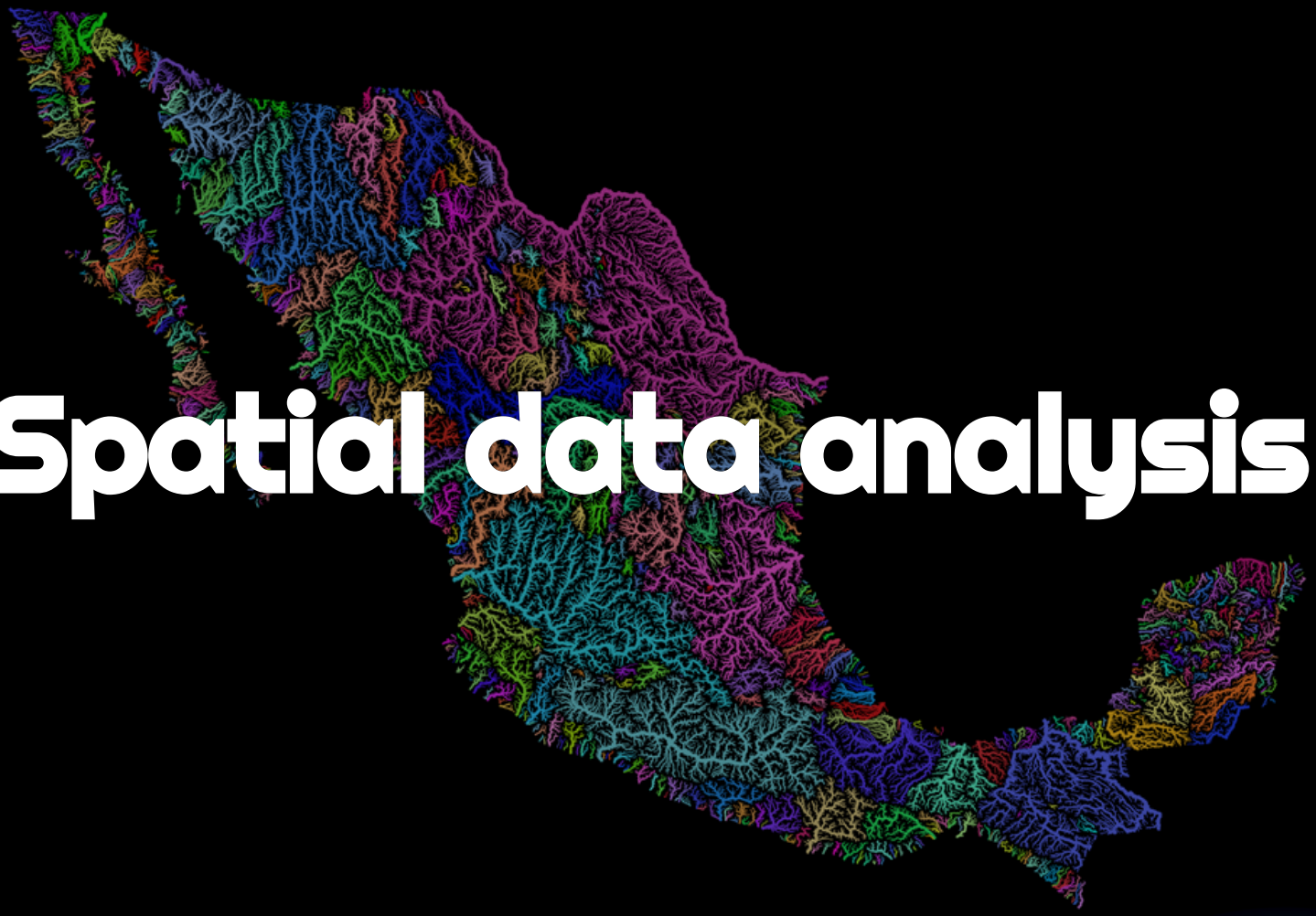
Re-plot the last map we plotted with **tmap**:

```
tmap_last()
```

leaflet

`leaflet` creates a map widget to which you add layers

```
map <- leaflet()
addTiles(map)
```



Spatial data analysis

Grasshopper
Geography

Resources

Maybe very disappointingly, I am leaving you to fend on your own here

But here are some great resources on the topic that should get you started
(& maybe I'll give another workshop on the subject some time?)

- R companion to Geographic Information Analysis
- Spatial data analysis

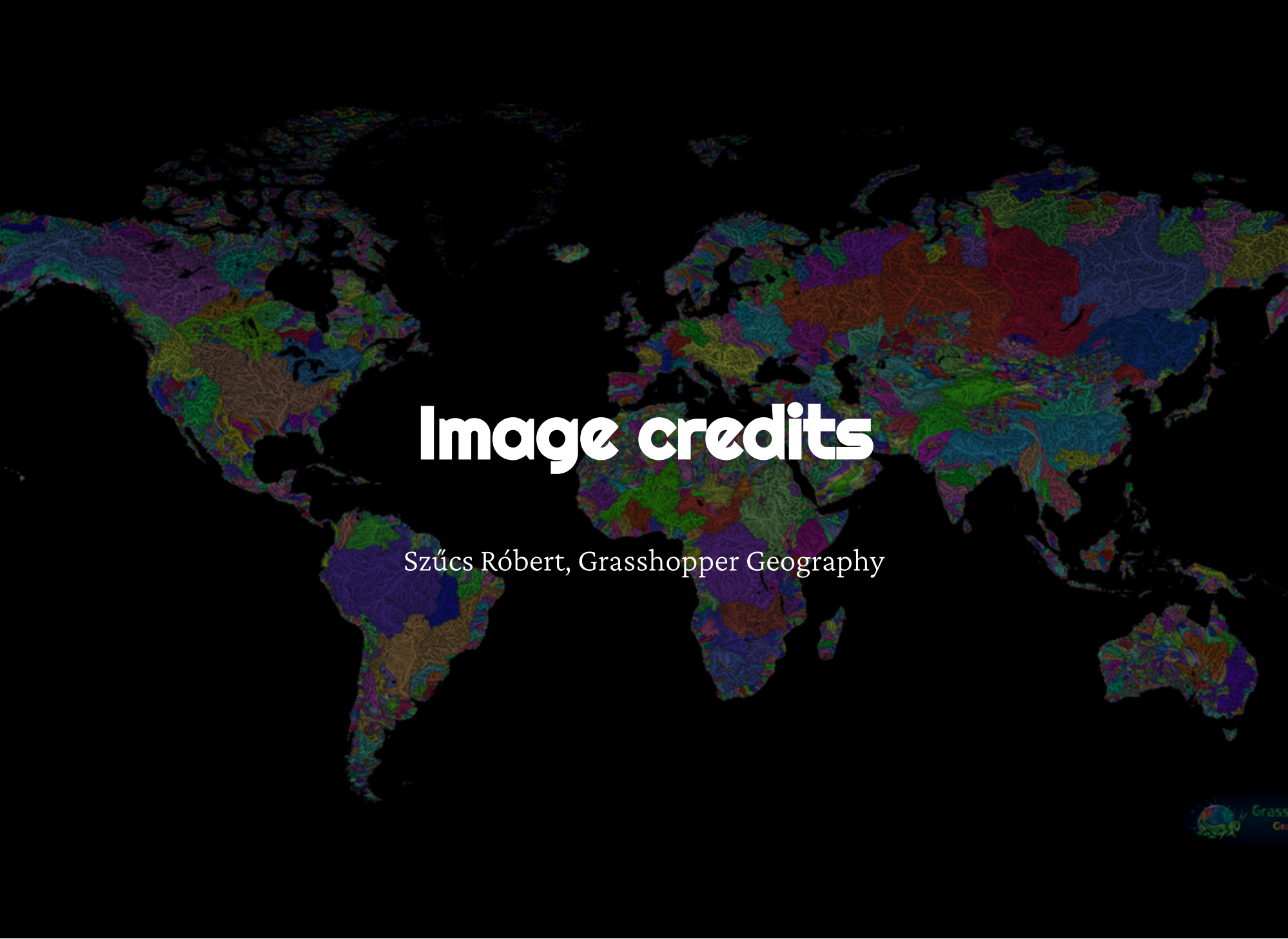


Image credits

Szűcs Róbert, Grasshopper Geography

Grasshopper Geography