Programming with R — A Beginners' Guide for Geoscientists

5 - Maps

Tobias Stephan

10/02/2022

Contents

Packages we need:

```
pacman::p_load(
  dplyr,
  squish,
  scales,

  # plotting
  ggplot2,
  scico,
  patchwork,
  ggrepel,
  ggthemes,
  rnaturalearth,
  ggnewscale,

  # geospatial packages
  sf,
  terra,
  tidyterra,
  sp,
  raster,

  # spatial interpolation
  gstat
)
```

```
# my functions and stuff for the course
source("R/swath.R")
```

Maps

Seismics at BC coast

In this tutorial, we want to produce a map of the western coast of Canada and its seismicity. First we load the earthquake table (*.csv) that I downloaded from the USGS earthquake database.

```
earthquakes <- read.csv("Data/Map/earthquakes_2000-2020.csv", sep = ",")  |>
  arrange(depth)
head(earthquakes)
```

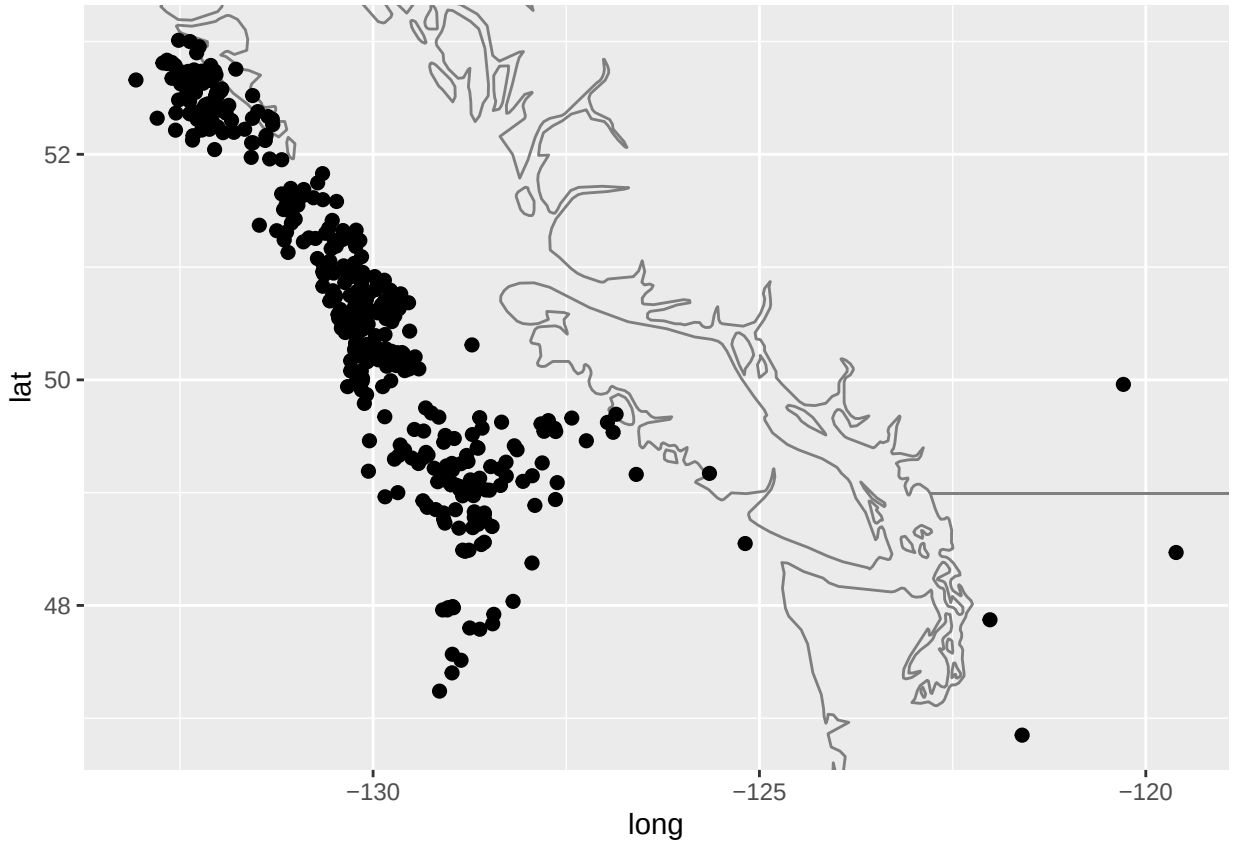
```
##                       time latitude longitude depth mag magType nst   gap  dmin
## 1 2006-10-08T02:48:26.650Z 46.84983 -121.6002 1.649 4.5      md  96  23.0    NA
## 2 2013-09-03T20:19:06.340Z 51.24400 -130.3971 2.730 6.1     mww  NA  44.0 1.704
## 3 2016-08-09T14:39:11.370Z 50.40180 -129.8461 4.720 4.7     mwr  NA 168.0 1.118
## 4 2013-12-15T18:11:47.540Z 49.26340 -127.8110 5.000 4.6     mwr  NA 138.0 0.868
## 5 2012-11-04T12:27:32.000Z 52.21400 -132.5570 5.000 5.1     mwr 449  36.8    NA
## 6 2006-03-15T22:55:44.000Z 48.79700 -128.6560 6.000 4.5     mwr  19 179.9    NA
##    rms net         id                 updated
## 1 0.35  uw uw10701498 2017-07-18T00:25:02.143Z
## 2 0.93  us usb000jg6b 2015-01-30T12:23:51.580Z
## 3 0.71  us us10006ccx 2016-10-26T03:10:24.040Z
## 4 1.44  us usc0001l77 2021-02-24T23:00:40.083Z
## 5   NA  us usp000jv0z 2014-11-07T01:49:18.128Z
## 6   NA  us usp000ec8a 2014-11-07T01:28:32.004Z
##                                 place       type horizontalError depthError
## 1   27 km NNE of Packwood, Washington earthquake           0.336       0.63
## 2  243 km WNW of Port McNeill, Canada earthquake              NA       3.70
## 3    196 km W of Port McNeill, Canada earthquake           6.600       5.30
## 4         134 km SW of Vernon, Canada earthquake              NA       1.80
## 5 277 km SSW of Prince Rupert, Canada earthquake              NA         NA
## 6        205 km WSW of Tofino, Canada earthquake              NA         NA
##   magError magNst   status locationSource magSource
## 1     0.04     18 reviewed             uw        uw
## 2       NA     NA reviewed             us        us
## 3       NA     NA reviewed             us       pgc
## 4       NA     NA reviewed             us       pgc
## 5       NA     NA reviewed            pgc       pgc
## 6       NA     NA reviewed            pgc       pgc
```

Basic maps

Now we start to set up a very basic map where we only show the earthquake locations.

```
map0 <- ggplot() + # extent of the map
  borders() + # political borders
  geom_point(
    data = earthquakes,
    aes(longitude, latitude), size = 2
  ) +
  coord_cartesian(
    xlim = range(earthquakes$longitude),
```

```
    ylim = range(earthquakes$latitude)
  )
map0
```



`ggplot()` treats the data as any other. It does not know yet that the data is actually geo-spatial.

To convert the data into spatial data, we need the package `sf` and define the coordinates and its coordinate system:

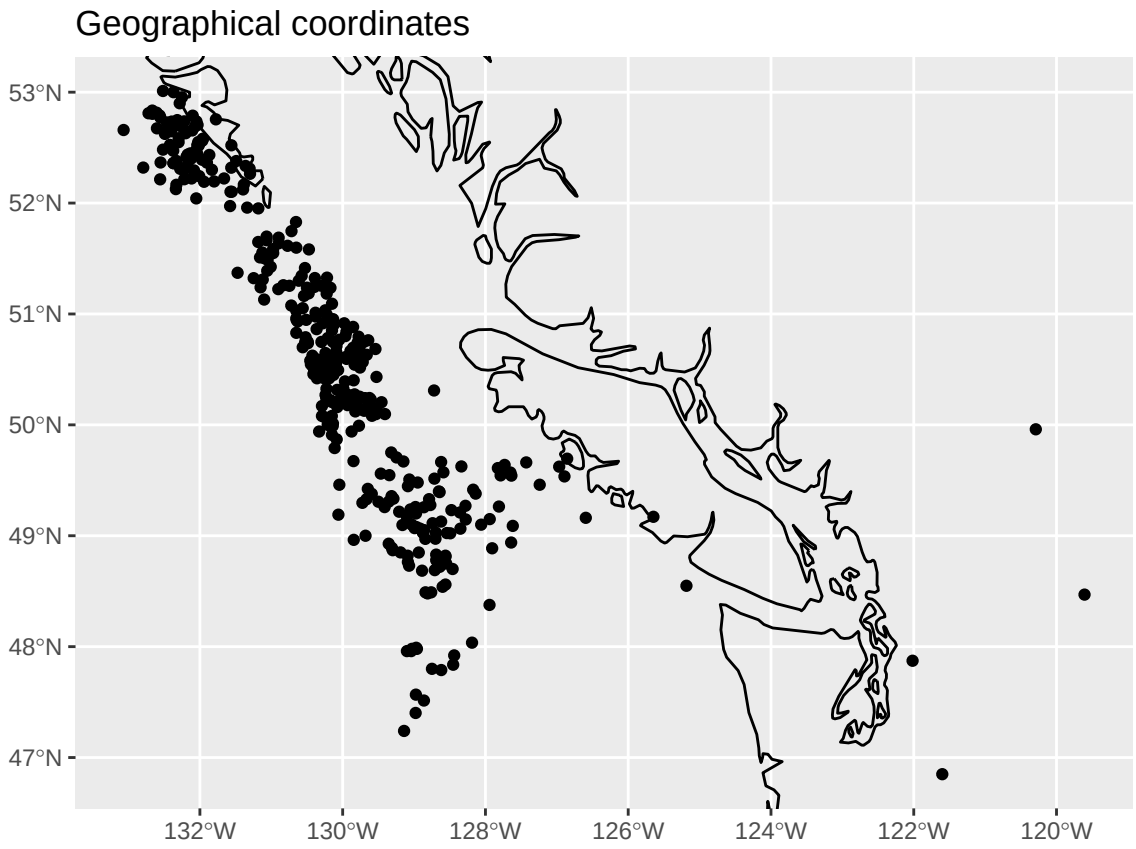
```
earthquakes_sf <- earthquakes |>
  sf::st_as_sf(
    coords = c("longitude", "latitude"),
    crs = "WGS84",
    remove = FALSE
    )
```

```
coastlines <- rnaturalearth::ne_coastline(50, returnclass = "sf")
```

Now the data is georeferenced! To plot the dat awith `ggplot()` we use `geom_sf()` and plot extents can be changed with `coord_sf()`:

```
map1 <- ggplot() +
  geom_sf(data = coastlines) +
  geom_sf(data = earthquakes_sf) +
  coord_sf(
    # extent of the map
    xlim = range(earthquakes_sf$longitude),
    ylim = range(earthquakes_sf$latitude),
  ) +
```

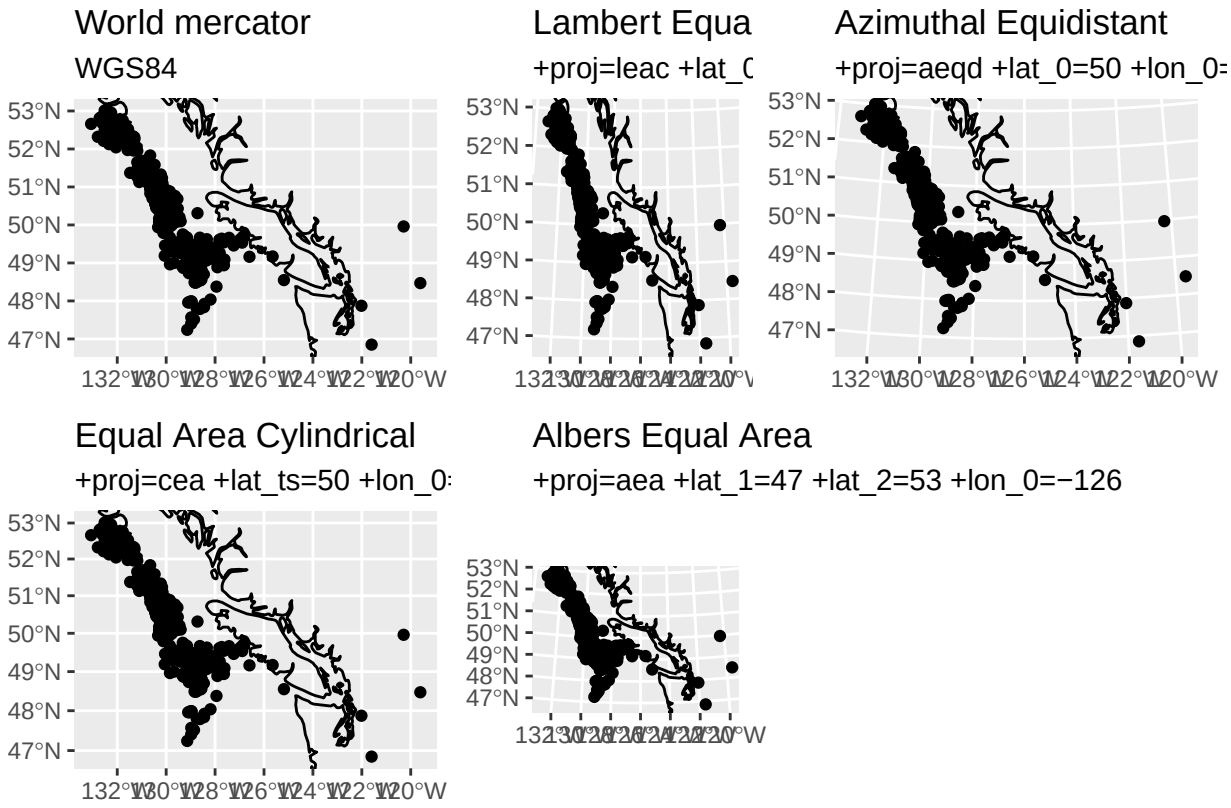
```
  labs(title = "Geographical coordinates")
map1
```



`ggplot()` allows us to reproject our spatial data into any coordinate reference system (CRS) by using `coord_sf(crs = ...)`

```
map1 +
  coord_sf(
    crs = "WGS84",
    default_crs = "WGS84",
    xlim = range(earthquakes_sf$longitude),
    ylim = range(earthquakes_sf$latitude),
  ) + # extent of the map
  labs(
    title = "World mercator",
    subtitle = "WGS84"
  ) +

  map1 +
  coord_sf(
    crs = st_crs("+proj=leac +lat_0=50 +lon_0=-126"),
    default_crs = "WGS84",
    # the projection
    xlim = range(earthquakes_sf$longitude),
    ylim = range(earthquakes_sf$latitude)
  ) + # extent of the map
  labs(
```

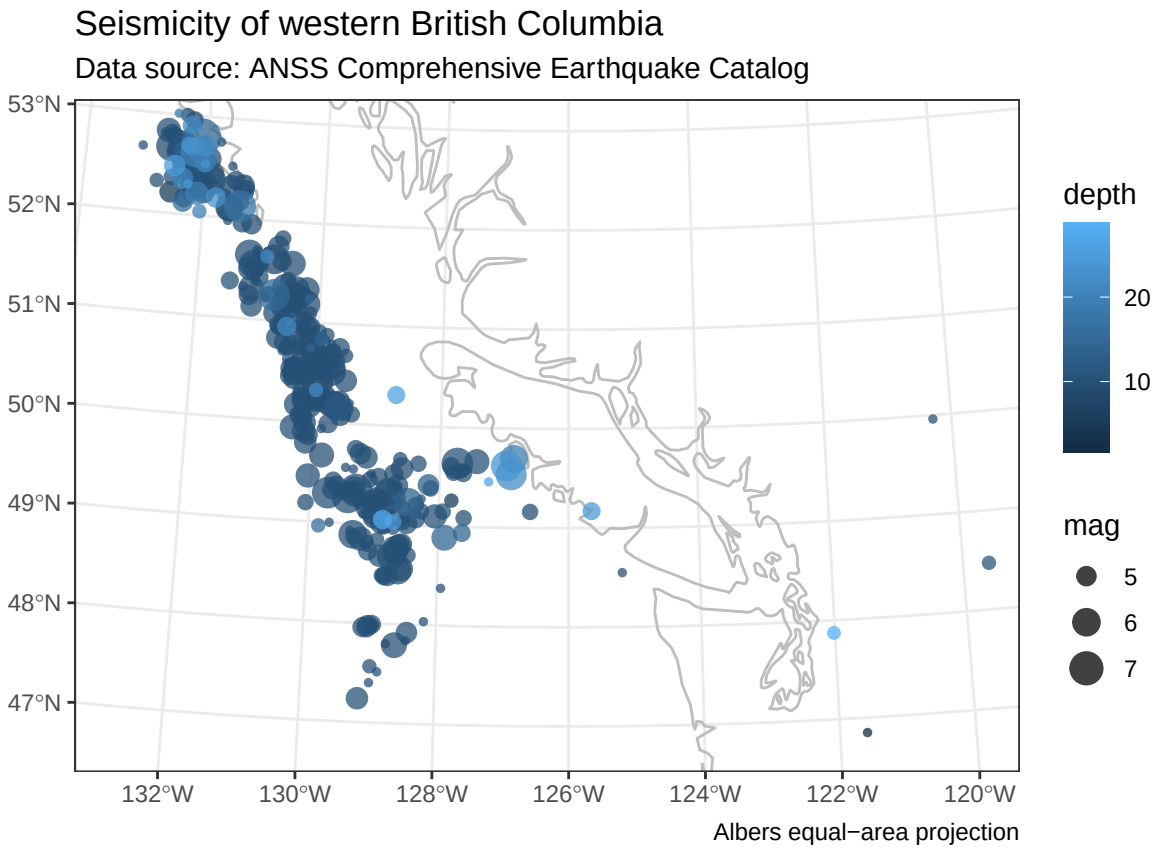
```
  title = "Lambert Equal Area Conic",
  subtitle = "+proj=leac +lat_0=50 +lon_0=-126"
) +

map1 +
coord_sf(
  crs = sf::st_crs("+proj=aeqd +lat_0=50 +lon_0=-126"),
  default_crs = "WGS84",
  # the projection
  xlim = range(earthquakes_sf$longitude),
  ylim = range(earthquakes_sf$latitude)
) + # extent of the map
labs(
  title = "Azimuthal Equidistant",
  subtitle = "+proj=aeqd +lat_0=50 +lon_0=-126"
) +

map1 +
coord_sf(
  crs = st_crs("+proj=cea +lat_ts=50 +lon_0=-126"),
  default_crs = "WGS84",
  # the projection
  xlim = range(earthquakes_sf$longitude),
  ylim = range(earthquakes_sf$latitude)
) + # extent of the map
labs(
  title = "Equal Area Cylindrical",
  subtitle = "+proj=cea +lat_ts=50 +lon_0=-126"
) +

map1 +
coord_sf(
  crs = st_crs("+proj=aea +lat_1=47 +lat_2=53 +lon_0=-126"),
  default_crs = "WGS84",
  xlim = range(earthquakes_sf$longitude),
  ylim = range(earthquakes_sf$latitude)
) + # extent of the map
labs(
  title = "Albers Equal Area",
  subtitle = "+proj=aea +lat_1=47 +lat_2=53 +lon_0=-126"
) +
plot_layout(guides = "collect")
```



R uses the *offical* projection/transformation parameters, either as EPSG, PROJ4 or WKT notation (they are used by ArcGIS or QGIS as well). A list of these CRS parameters can be found at: https://proj.org/en/9.3/operations/index.html

Let's use the Albers equal area projection. In the next step, we will modify the earthquake symbols. We can just add some properties of the data set (columns of the data.frame) into the aesthetic (`aes()`), such as the color, the shape, the size, or the opacity (`alpha`).

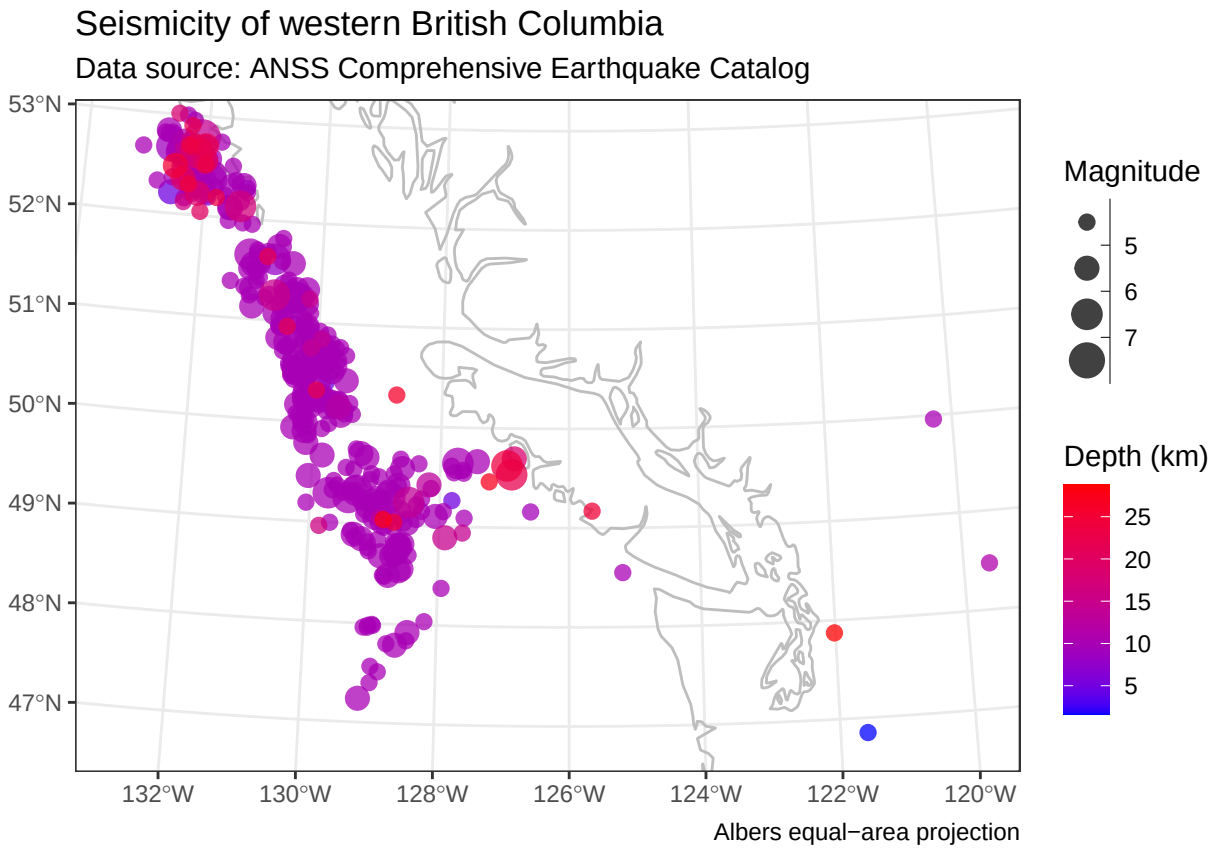
```
map2 <- ggplot() +
  geom_sf(data = coastlines, color = "grey") +
  geom_sf(data = earthquakes_sf, aes(color = depth, size = mag), alpha = .75) +
  coord_sf(
    crs = st_crs("+proj=aea +lat_1=47 +lat_2=53 +lon_0=-126"),
    default_crs = "WGS84",
    xlim = range(earthquakes_sf$longitude),
    ylim = range(earthquakes_sf$latitude)
  ) +
  labs(
    title = "Seismicity of western British Columbia",
    subtitle = "Data source: ANSS Comprehensive Earthquake Catalog",
    caption = "Albers equal-area projection"
  ) +
  theme_bw()
map2
```



The plot above will produce some default outputs (colors).

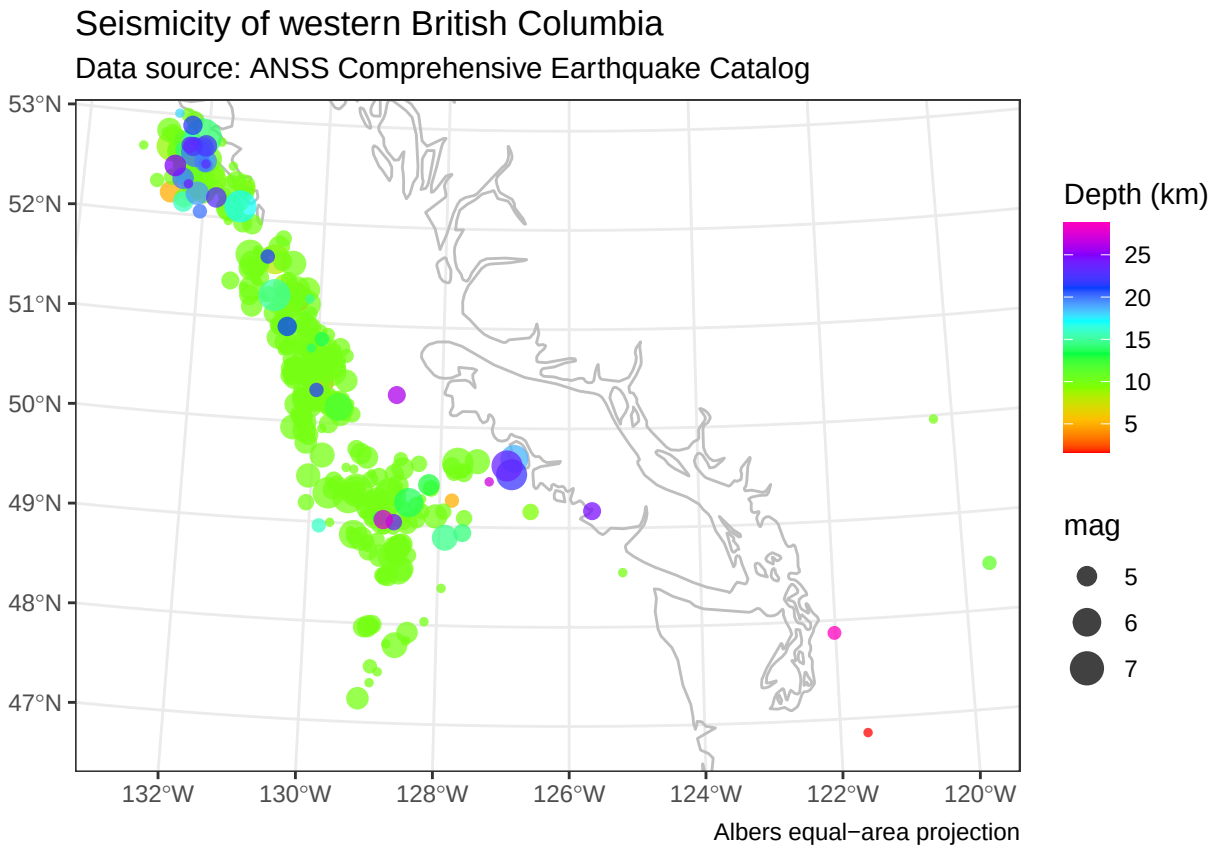
We can modify the colors and aesthetics by using `scale_[aesthetics]`,

```
map2 +
  scale_colour_gradient(
    name = "Depth (km)",
    low = "blue",
    high = "red",
    breaks = seq(0, 30, 5)
  ) +
  scale_size_binned(name = "Magnitude")
```



We do not have to define the color palette. `ggplot()` offers a variety of pre-installed color palettes.

```
map2 +
  scale_color_gradientn(
    colors = rainbow(8),
    name = "Depth (km)",
    breaks = seq(0, 30, 5)
  )
```

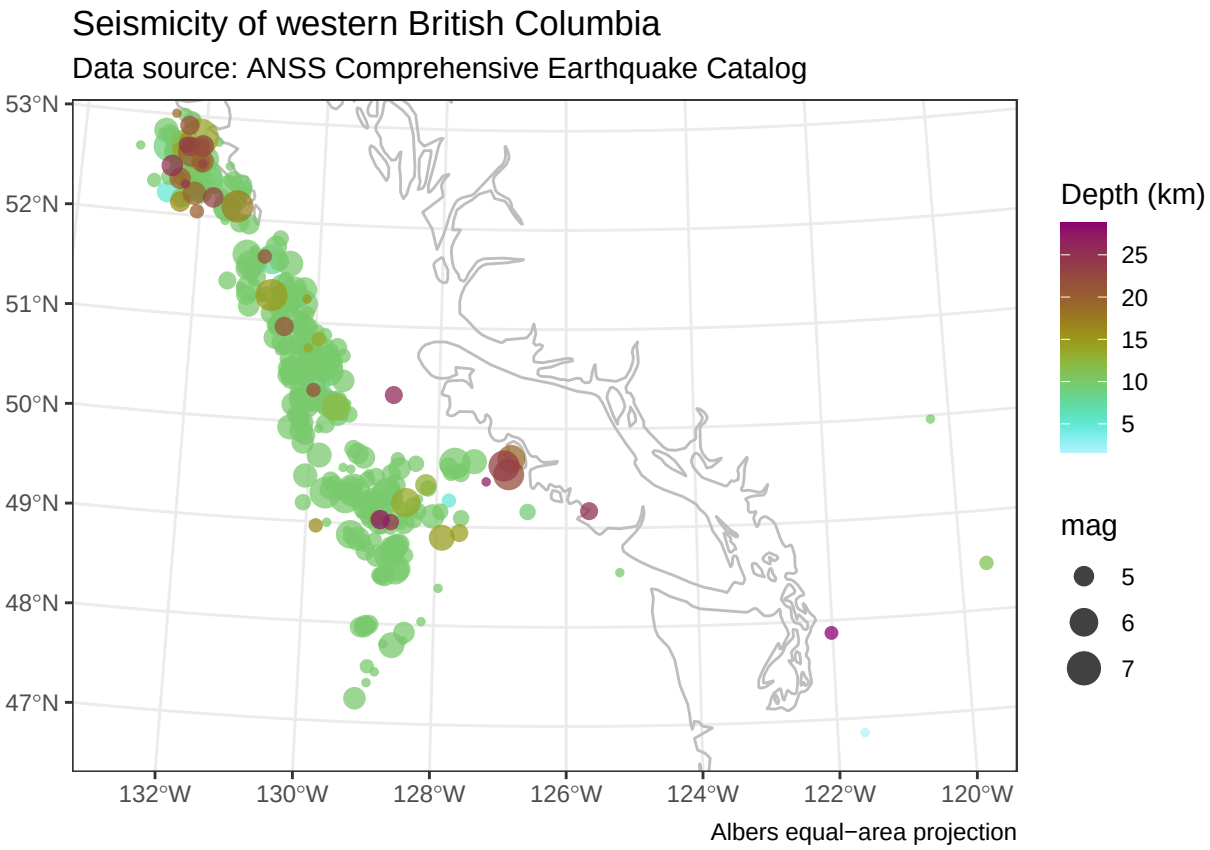


Many scientific journals already require (or at least recommend) to use readable colors for colour-vision deficient or colorblind people.

The package `scico` provides a list of scientific color palettes that are

- colorblind-friendly
- perceptually uniform and ordered to represent data both fairly, without visual distortion, and intuitively, and
- still readable when printed in black and white

```
map2 + scico::scale_color_scico(
  palette = "hawaii",
  name = "Depth (km)",
  direction = -1,
  breaks = seq(0, 30, 5)
)
```



Import Shape-files

Now we want to add the plate boundaries to the map. The plate boundaries are stored in a ESRI shape file that can be imported using `st_read()` from the sf package.

```
# load shape file as 'simple feature'
plates <- st_read("Data/Map/plate_boundaries.shp") |>
  st_cast("MULTILINESTRING")# plate boundaries
```

```
## Reading layer ‘plate_boundaries’ from data source
##   ‘C:\Users\tobis\Documents\GIT_repos\R_Geo_workshop\Data\Map\plate_boundaries.shp’
##   using driver ‘ESRI Shapefile’
## Simple feature collection with 54 features and 3 fields
## Geometry type: MULTIPOLYGON
## Dimension:     XY
## Bounding box:  xmin: -180 ymin: -90 xmax: 180 ymax: 89.99392
## Geodetic CRS:  GCS_unknown
```

```
st_crs(plates) <- "WGS84" # set the CRS
```

Now add the plate boundaries to our map:

```
map3 <- ggplot() +
  geom_sf(
    data = plates,
    lwd = 2,
    color = "#D55E00"
  ) +
  geom_sf(data = coastlines, color = "grey") +
```

```
  geom_sf(data = earthquakes_sf,
          aes(color = depth, size = mag),
          alpha = .75) +
  scico::scale_color_scico(
    palette = "hawaii",
    name = "Depth (km)",
    direction = -1,
    breaks = seq(0, 30, 5)
  ) +
  scale_size_binned("Magnitude") +
  labs(
    title = "Seismicity of western British Columbia",
    subtitle = "Data source: ANSS Comprehensive Earthquake Catalog",
    caption = "Albers equal-area projection"
  ) +
  theme_bw() +
  coord_sf(
    crs = st_crs("+proj=aea +lat_1=47 +lat_2=53 +lon_0=-126"),
    default_crs = "WGS84",
    xlim = range(earthquakes_sf$longitude),
    ylim = range(earthquakes_sf$latitude)
  )
```

Create a spatial object

To create spatial objects, e.g. a point, line etc., we need to specify the coordinates and the CRS. Create a lines object

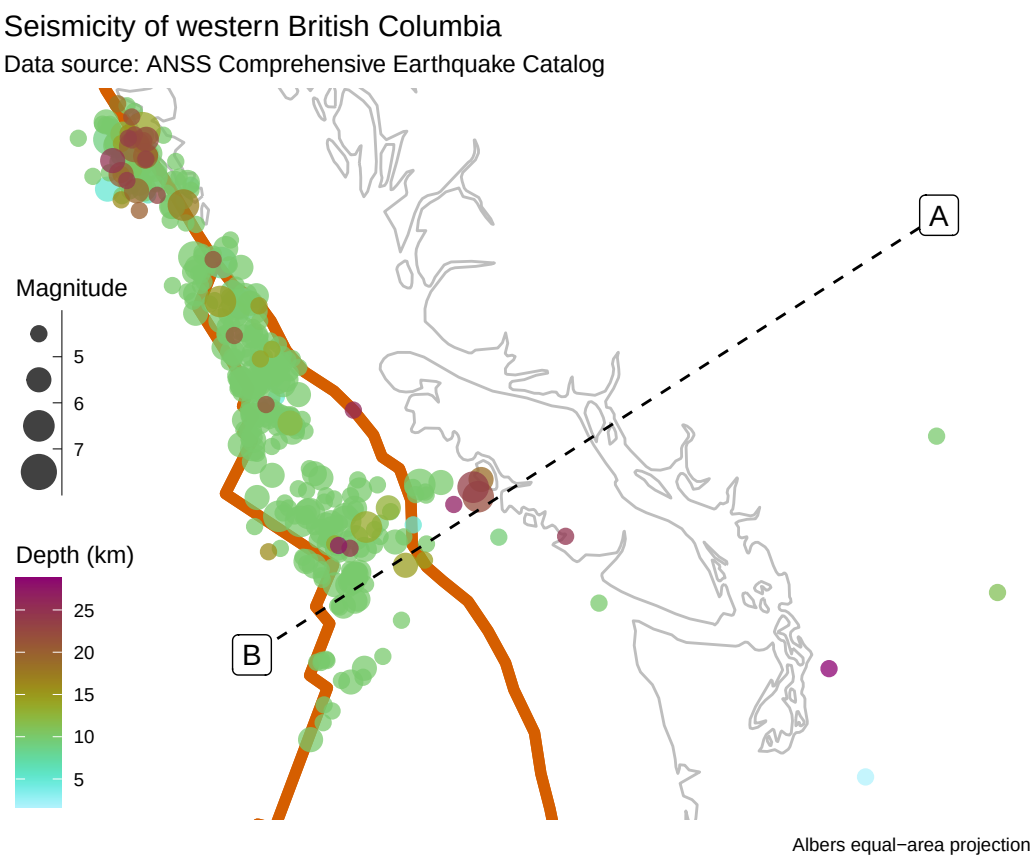
```
# two lines as dataframe
points <- data.frame(
  lon = c(-120, -130),
  lat = c(52, 48),
  name = c("A", "B")
)

points_sf <- st_as_sf(points, coords = c("lon", "lat"), crs = "WGS84")

# as a spatial object
line_sf <- points_sf |>
  summarize() |>
  st_cast("LINESTRING")
```

Plot the new object as a profile line and add to the map:

```
map3 +
  # adds the profile line
  geom_sf(data = line_sf, lty = 2) +

  # end points
  geom_sf(data = points_sf, size = 2) +

  # label the points
  geom_sf_label(data = points_sf, aes(label = name)) +

  # change to a map theme
```

```
  ggthemes::theme_map() +
  coord_sf(
    crs = sf::st_crs("+proj=aea +lat_1=47 +lat_2=53 +lon_0=-126"),
    default_crs = "WGS84",
    xlim = range(earthquakes_sf$longitude),
    ylim = range(earthquakes_sf$latitude)
  )
```

```
## Coordinate system already present. Adding new coordinate system, which will
## replace the existing one.
```



Save ESRI shape file

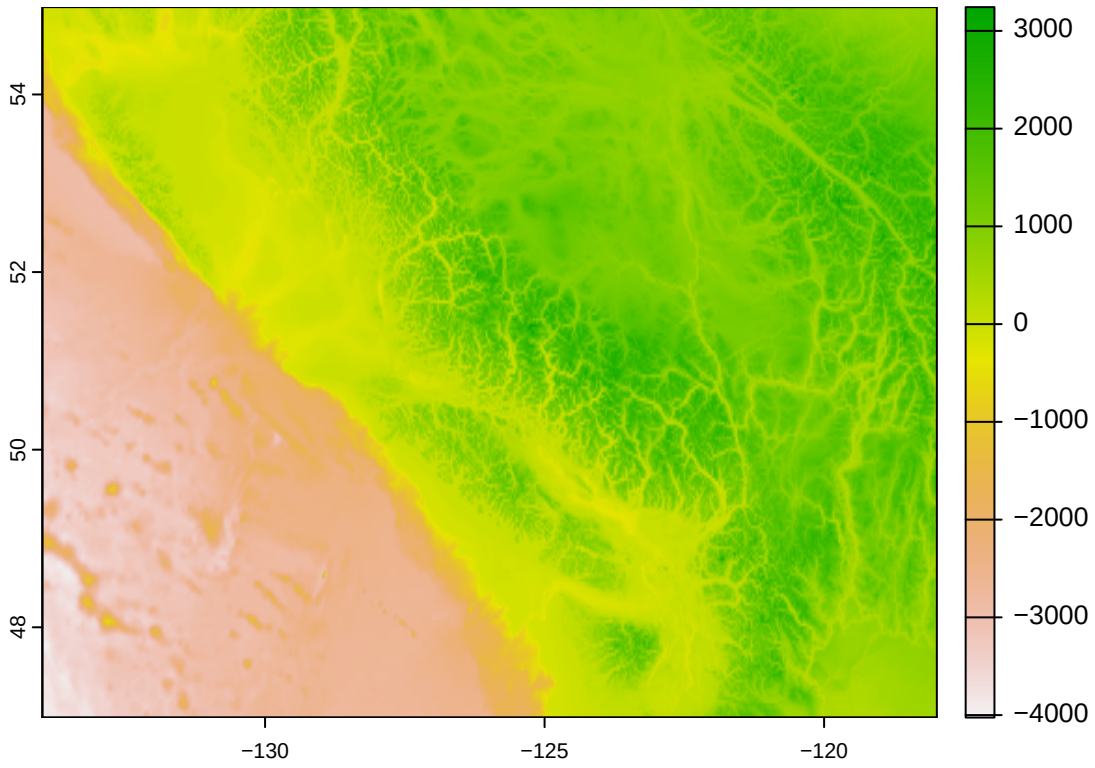
Save line into ESRI shapefile:

```
st_write(line_sf, "Data/Map/my_line.shp")
```

Import Raster

To import and handle raster datasets, we need the **terra** package. The import is done by using `rast()`

```
dem <- terra::rast("Data/Map/ETOPO_vancouver.tif")

# first glimpse
plot(dem)
```



Crop a raster To crop or clip a raster along a spatial feature (e.g. the extent of our earthquake dataset), we can use `crop()`:

```
crop_extent <- st_bbox(earthquakes_sf) + c(-1,-1,1,1)
crop_extent
```

```
##       xmin       ymin       xmax       ymax
## -134.06900   45.84983 -118.60750   54.01100
```

```
dem_cropped <- terra::crop(dem, crop_extent)
```

The raster resolution, i.e. the pixel cell size can be retrieved by `res()`

```
terra::res(dem_cropped)
```

```
## [1] 0.03333333 0.03333333
```

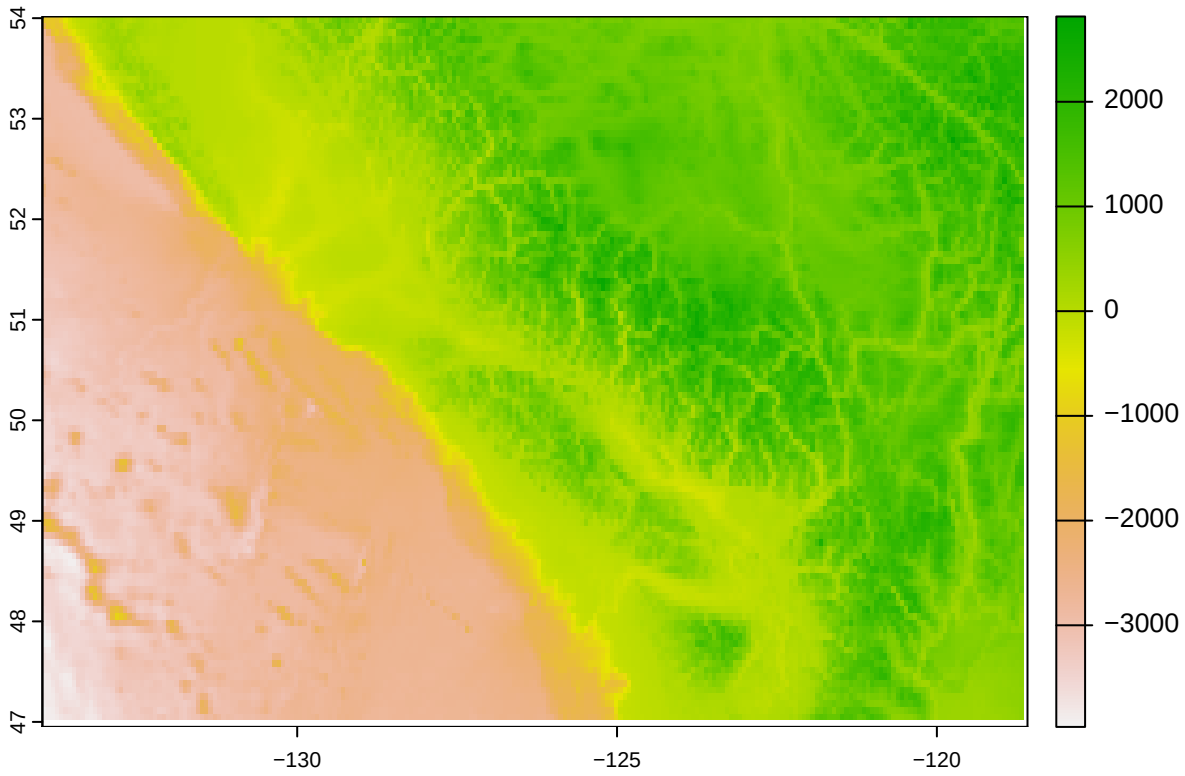
The raster resolution is 0.3 x 0.3 degree.

We can decrease and increase the resolution with `aggregate()` and `disaggregate()`, respectively.

```
# lower the resolution by factor 5:
dem_low <- terra::aggregate(dem_cropped, fact = 2)
terra::res(dem_low)
```

```
## [1] 0.06666667 0.06666667
```

```r
plot(dem_low)
```



Reproject the raster Raster data can also be transformed or reprojected into different CRSs.

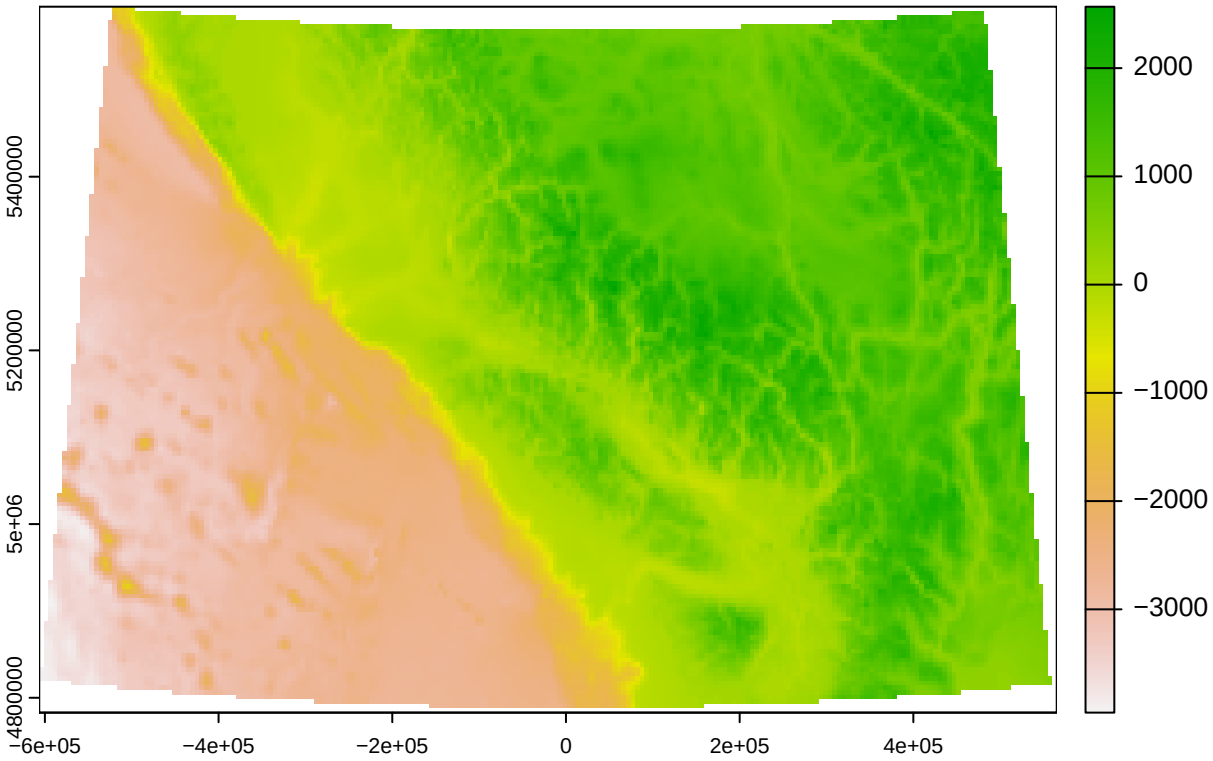
To use the `project()` function, we need to define two things:

- the object we want to reproject and
- the CRS that we want to reproject it to.

The syntax is `terra::project(RasterObject, CRSToReprojectTo)`

Here, we want to transform the raster (in WGS84) into an Albers equal area projection.

```r
dem_aea <- terra::project(
  dem_low,
  terra::crs("+proj=aea +lat_1=47 +lat_2=53 +lon_0=-126")
  )
plot(dem_aea)
```



When we reproject a raster, we move it from one "grid" to another. Thus, we are modifying the data! Keep this in mind as we work with raster data.

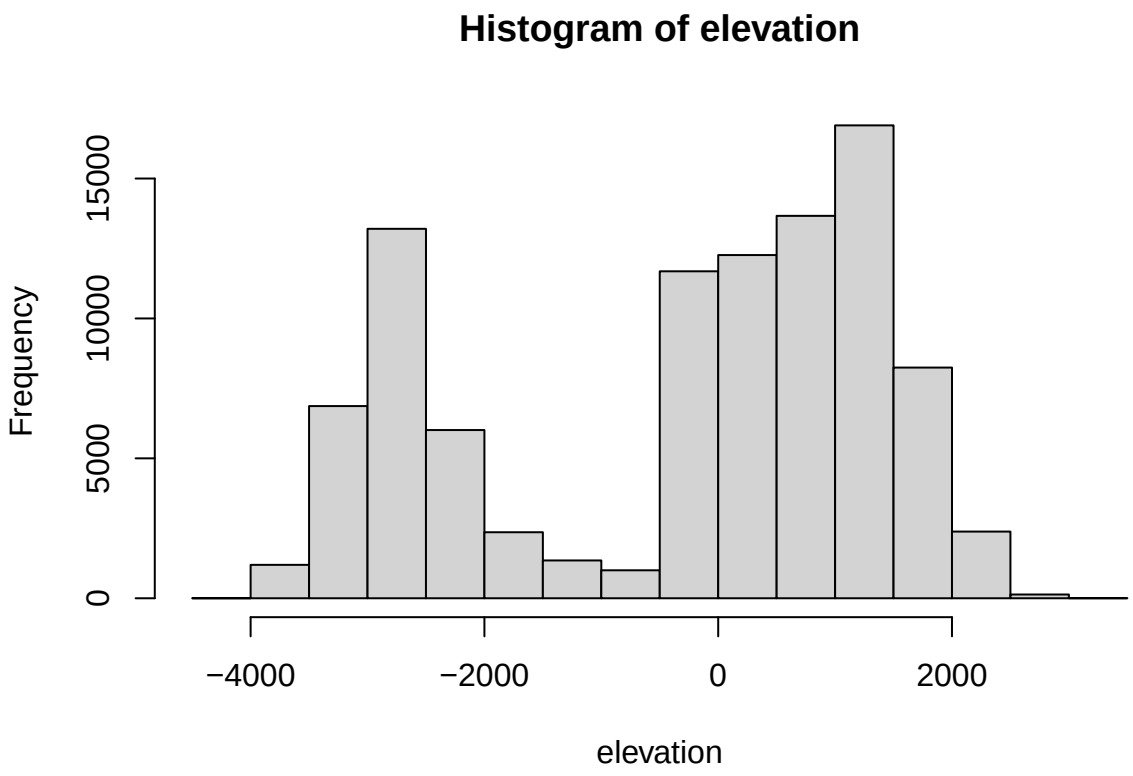
```
terra::writeRaster(dem_aea, "Data/Map/dem_cropped.tif", overwrite = TRUE)
```

Save raster Thereby, we can also do some statistics on the raster's "z" value (here the elevation):

```
# Stats numbers
elevation <- terra::values(dem_cropped)
summary(elevation)
```

```
##  ETOPO_vancouver
##  Min.   :-4025.0
##  1st Qu.:-2334.0
##  Median :  120.0
##  Mean   : -317.8
##  3rd Qu.: 1088.0
##  Max.   : 3243.0
```

```
# Histogram
hist(elevation)
```




```
slope <- terra::terrain(dem_cropped, "slope", unit = "radians")
aspect <- terra::terrain(dem_cropped, "aspect", unit = "radians")

hill <- terra::shade(slope, aspect,
              angle = c(45, 45, 45, 80),
              direction = c(225, 270, 315, 135)
              )
hill <- Reduce(mean, hill)
```

Hillshade

Plot the raster To plot a raster we can use `tidyterra::geom_spatraster()`.

Additionally we will add the city of Vancouver to our map

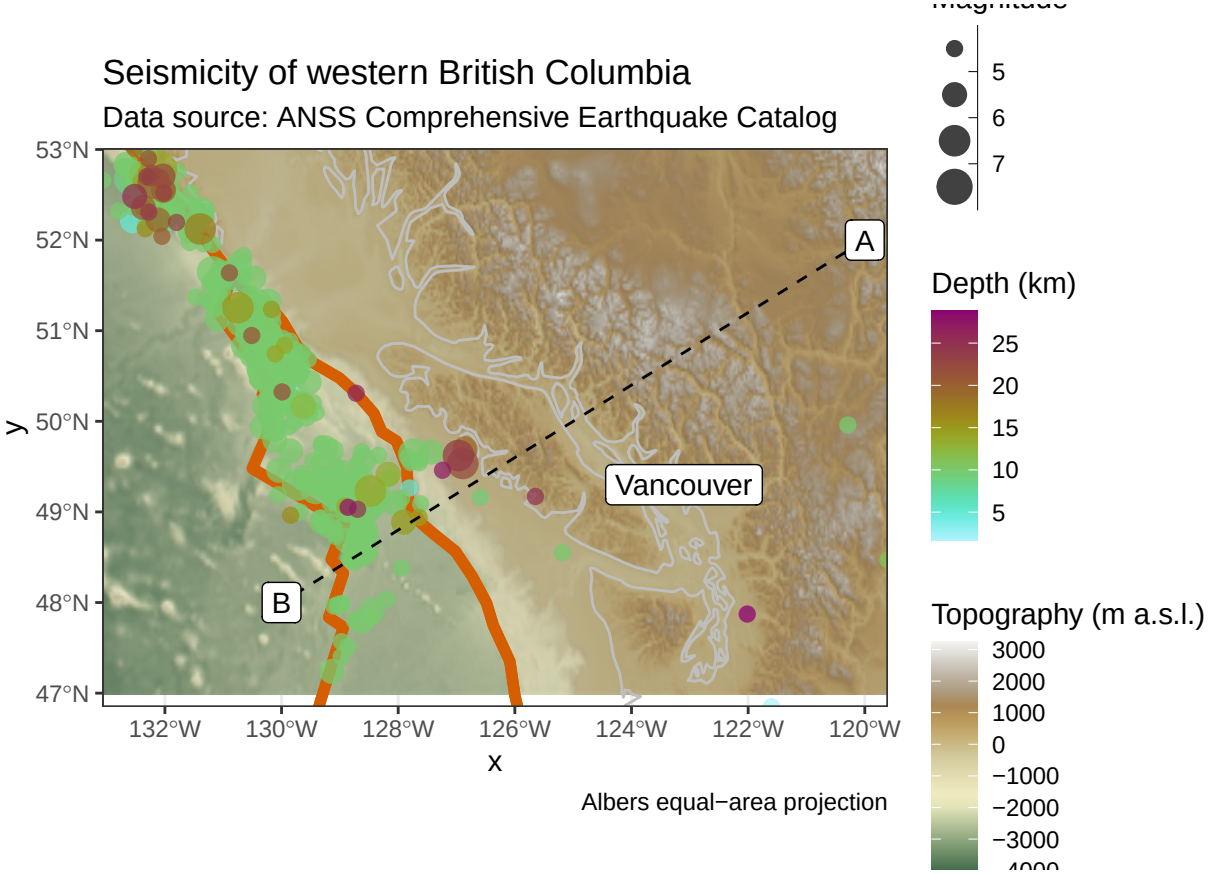
```
vancouver <- data.frame(city = "Vancouver", lat = 49.3, lon = -123.1) |>
  st_as_sf(coords = c("lon", "lat"), crs = "WGS84")
```

Final plot:

```
map4 <- ggplot() +
  tidyterra::geom_spatraster(data = hill, show.legend = FALSE) +
  scico::scale_fill_scico(palette = "grayC") +
  ggnewscale::new_scale_fill() +
  tidyterra::geom_spatraster(data = dem_cropped, alpha = .8) +
  tidyterra::scale_fill_wiki_c(
    name = "Topography (m a.s.l.)",
    limits = range(elevation),
    oob = scales::squish,
    breaks = seq(-5000, 5000, 1000),
```

```
    na.value = NA
  ) +
  geom_sf(
    data = plates,
    lwd = 2,
    color = "#D55E00",
    fill = NA
  ) +
  geom_sf(data = coastlines, color = "grey") +
  geom_sf(data = earthquakes_sf, aes(color = depth, size = mag), alpha = .75) +
  scico::scale_color_scico(
    palette = "hawaii",
    name = "Depth (km)",
    direction = -1,
    breaks = seq(0, 30, 5)
  ) +
  scale_size_binned("Magnitude") +
  geom_sf(data = line_sf, lty = 2) +
  geom_sf(data = points_sf, size = 2) +
  geom_sf_label(data = points_sf, aes(label = name)) +
  geom_sf(data = vancouver) +
  geom_sf_label(data = vancouver, aes(label = city)) +
  coord_sf(
    #crs = sf::st_crs("+proj=aea +lat_1=47 +lat_2=53 +lon_0=-126"),
    # default_crs = "WGS84",
    xlim = range(earthquakes_sf$longitude),
    ylim = range(earthquakes_sf$latitude),
    expand = FALSE
  ) +
  labs(
    title = "Seismicity of western British Columbia",
    subtitle = "Data source: ANSS Comprehensive Earthquake Catalog",
    caption = "Albers equal-area projection"
  ) +
  theme_bw()
map4
```

```
## Warning in st_point_on_surface.sfc(sf::st_zm(x)): st_point_on_surface may not
## give correct results for longitude/latitude data

## Warning in st_point_on_surface.sfc(sf::st_zm(x)): st_point_on_surface may not
## give correct results for longitude/latitude data
```



Elevation profile (Swath)

Swath profiles are used to condense topographic information from a rectangular area (swath) to an elevation profile.

Technically, they are not so simple to create (see here for more details.

Thus, the calculation will be CPU intensive, in particular for large and high resolution raster data! Our example data set should be doable on most computers... (I hope).

`swathR` calculates the elevation data in our swath. To define the swath we have to define `k` – the number of lines parallel to our profile line ("line.pts"), `dist` – the distance of the lines to our profile line, and the coordinate reference system – `crs`.

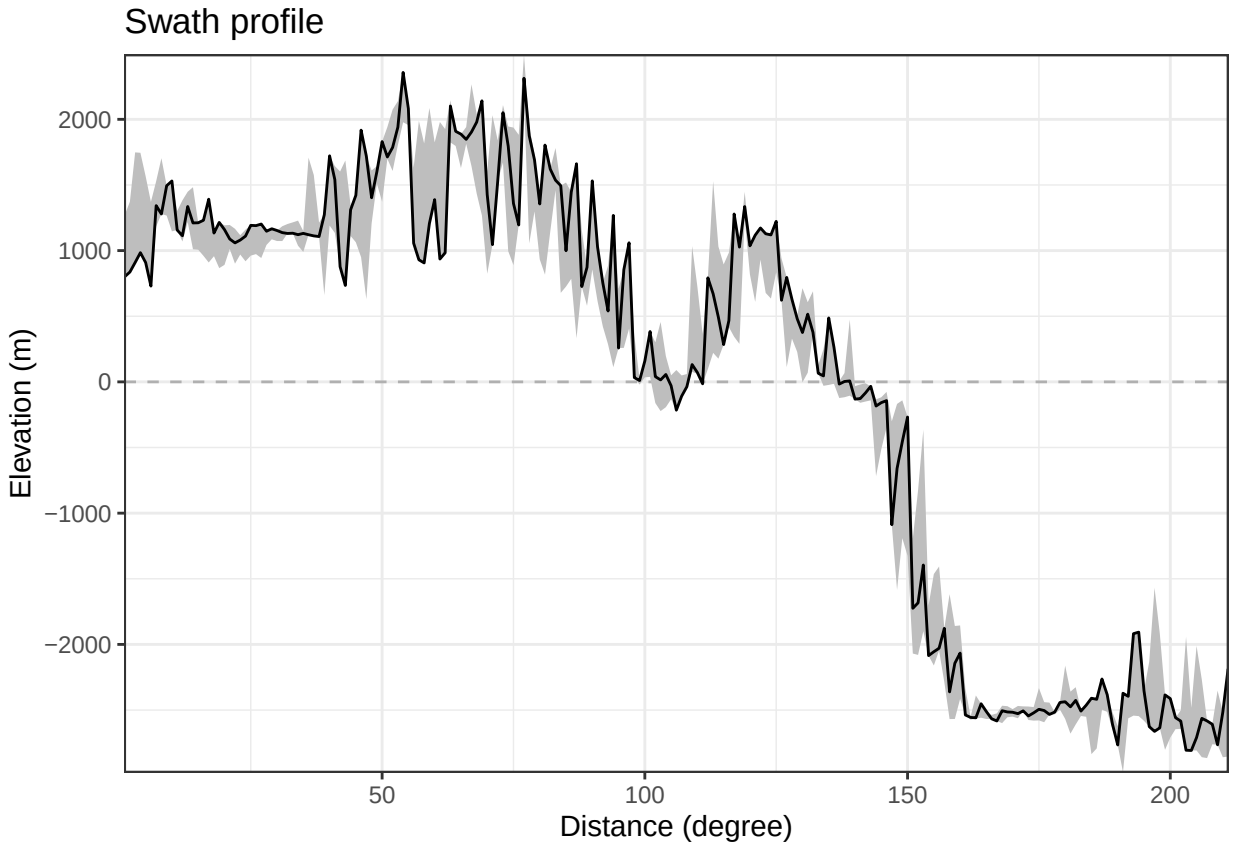
```
# extracts the swath profile and some statistics
swath.list <- swathR(
  coords = as.matrix(points[, 1:2]),
  raster = raster::raster(dem_low)
)
```

The function returns a list of the lines coordinates, some statistics, and the elevation data along the lines. I coded a function `swathR_profile` that extracts the swath profile (i.e. the elevation data along the center line, the minimum and maximum elevation within the swath) from this list.

```
swath.profile <- swath_profile(swath.list)
```

To plot the swath with `ggplot`, we can use the `geom_ribbon()` option and define the min and max values for the elevation:

```r
ggplot(swath.profile, aes(distance, elevation)) +
  coord_cartesian(expand = FALSE) +
  # add a horizontal aline to show the sea level
  geom_hline(
    yintercept = 0,
    lty = 2,
    alpha = .25
  ) +
  # swath
  geom_ribbon(
    aes(
      ymin = min,
      ymax = max
    ),
    fill = "grey"
  ) +
  # elevtion along the center line
  geom_line() +
  labs(title = "Swath profile", x = "Distance (degree)", y = "Elevation (m)") +
  theme_bw()
```



Because our input data is in the geographical coordinate system, the distances are given in degrees. You can use my function `greatcircle_distance()` to get the distances in km along the line:

```r
swath.profile$distance.km <- deg_2_km(swath.profile$distance, c(points[1, 2], points[1, 1]), c(points[2
```

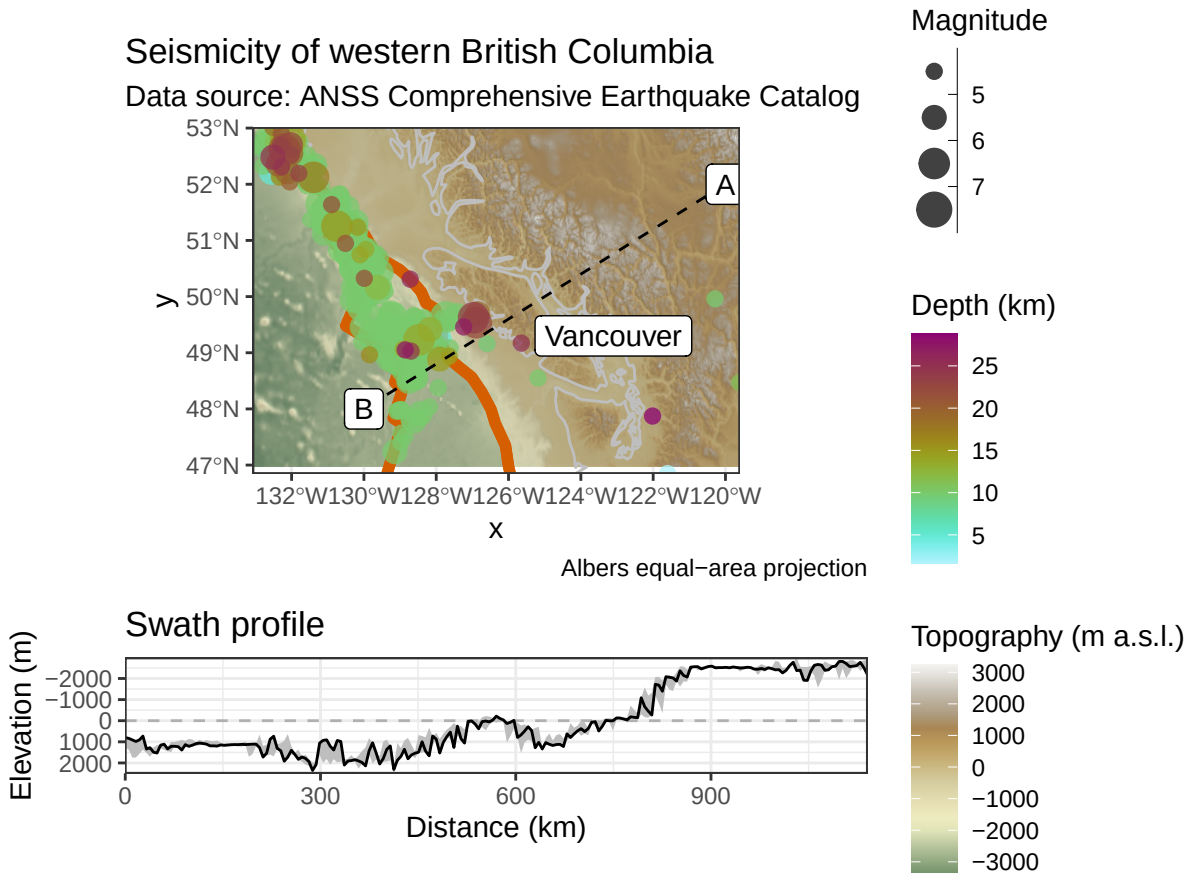
Now we can set up the profile plot ...

```
swath.plot <-
  ggplot(swath.profile, aes(distance.km, elevation)) +
  coord_cartesian(expand = FALSE) +
  geom_hline(
    yintercept = 0,
    lty = 2,
    alpha = .25
  ) +
  geom_ribbon(
    aes(
      ymin = min,
      ymax = max
    ),
    fill = "grey"
  ) +
  geom_line() +
  labs(title = "Swath profile", x = "Distance (km)", y = "Elevation (m)") +
  theme_bw()
```

... and combine it together with our map

```
map.combo <-
  map4 /
  (swath.plot + scale_y_reverse()) +
  plot_layout(heights = c(3, 1), guides = "collect")
map.combo
```

```
## Warning in st_point_on_surface.sfc(sf::st_zm(x)): st_point_on_surface may not
## give correct results for longitude/latitude data

## Warning in st_point_on_surface.sfc(sf::st_zm(x)): st_point_on_surface may not
## give correct results for longitude/latitude data
```



To save a plot, we can use `ggsave()` by giving the file path and the output size. It automatically identifies the file format from your file name.

Save a map

```
ggsave(
  map.combo, filename = "Data/DEM/my_map.pdf",
  width = 11, height = 8.5, scale = 0.5
  )
```

Spatial interpolation

A "heat map" refers to a spatially interpolated value. In our example we want to interpolate the magnitude of the earthquakes for the entire area in derive a somewhat "seismic hazard map".

Next we need to create the interpolation grid where we want to interpolate onto.

```
grid <- st_make_grid(earthquakes_sf, cellsize = .5, what = "centers")
```

Inverse distance weigthing Now we have set up our points and a grid to interpolate onto, we are ready carry out some interpolation. The first method we will try is inverse distance weighting (IDW) as this will not require any special modelling of spatial relationships.

To generate a surface using inverse distance weighting, use the IDW function in gstat. Check the help file for IDW – `?idw` – for information about what this formula is doing.

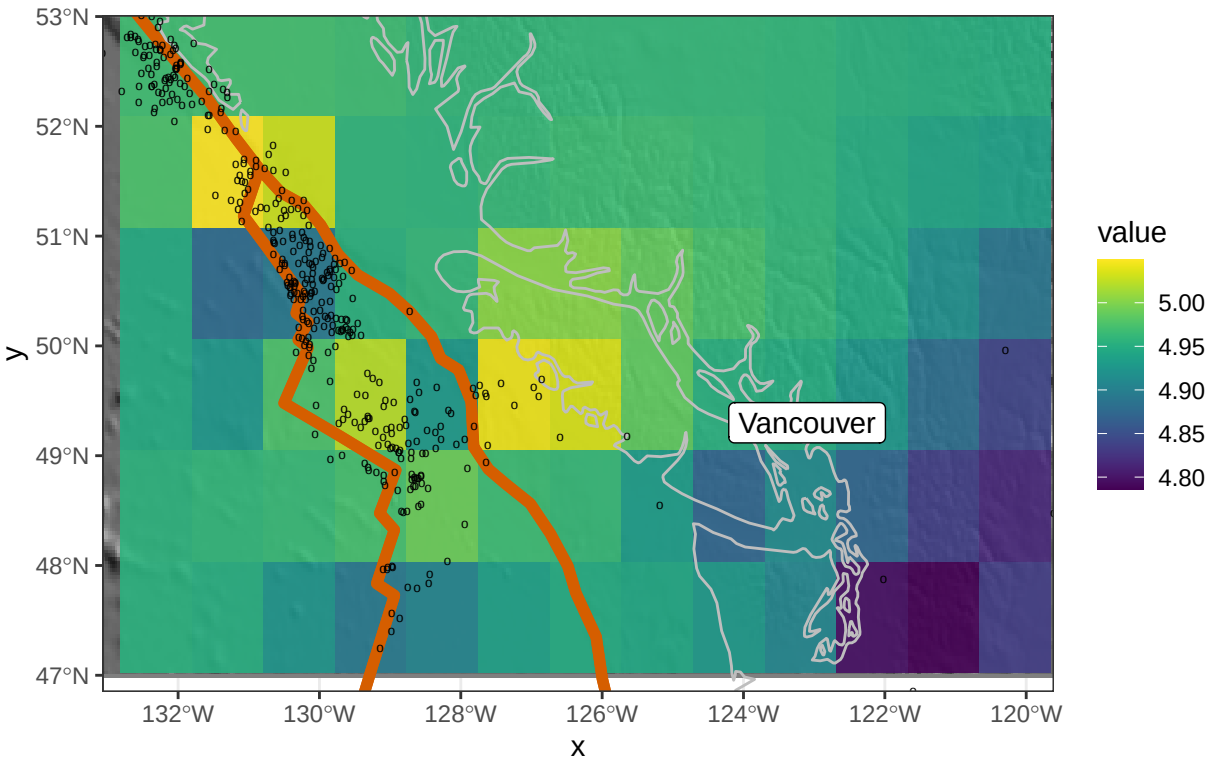
```
idw_res <- gstat::idw(
  formula = mag ~ 1, newdata = grid, locations = earthquakes_sf
  )
```

```
## [inverse distance weighted interpolation]
```

```
idw_rast <- terra::rasterize(
  terra::vect(idw_res),
  terra::rast(terra::vect(grid)),
  field = "var1.pred")
```

Now we have our IDW output we can plot it using ggplot2.

```
ggplot() +
    tidyterra::geom_spatraster(data = hill, show.legend = FALSE) +
  scico::scale_fill_scico(palette = "grayC") +
  ggnewscale::new_scale_fill() +
  tidyterra::geom_spatraster(data = idw_rast, alpha = .9) +
  viridis::scale_fill_viridis() +
  geom_sf(
    data = plates,
    lwd = 2,
    color = "#D55E00",
    fill = NA
  ) +
  geom_sf(data = coastlines, color = "grey") +
  geom_sf(data = earthquakes_sf, shape = "o") +
  geom_sf(data = vancouver) +
  geom_sf_label(data = vancouver, aes(label = city)) +
  coord_sf(
    default_crs = "WGS84",
    xlim = range(earthquakes_sf$longitude),
    ylim = range(earthquakes_sf$latitude),
    expand = FALSE
  ) +
  theme_bw()
```

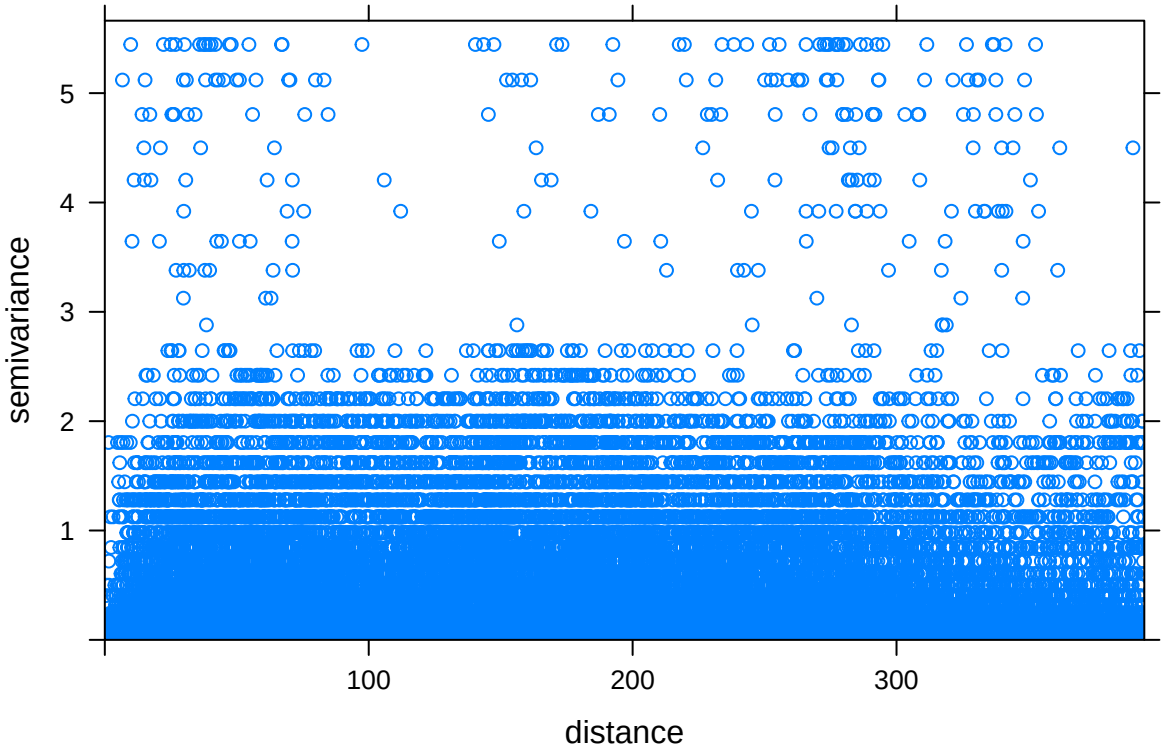


Save the map into png

```
ggsave(map3, filename = "Data/DEM/my_heatmap.png", width = 11, height = 8.5)
```

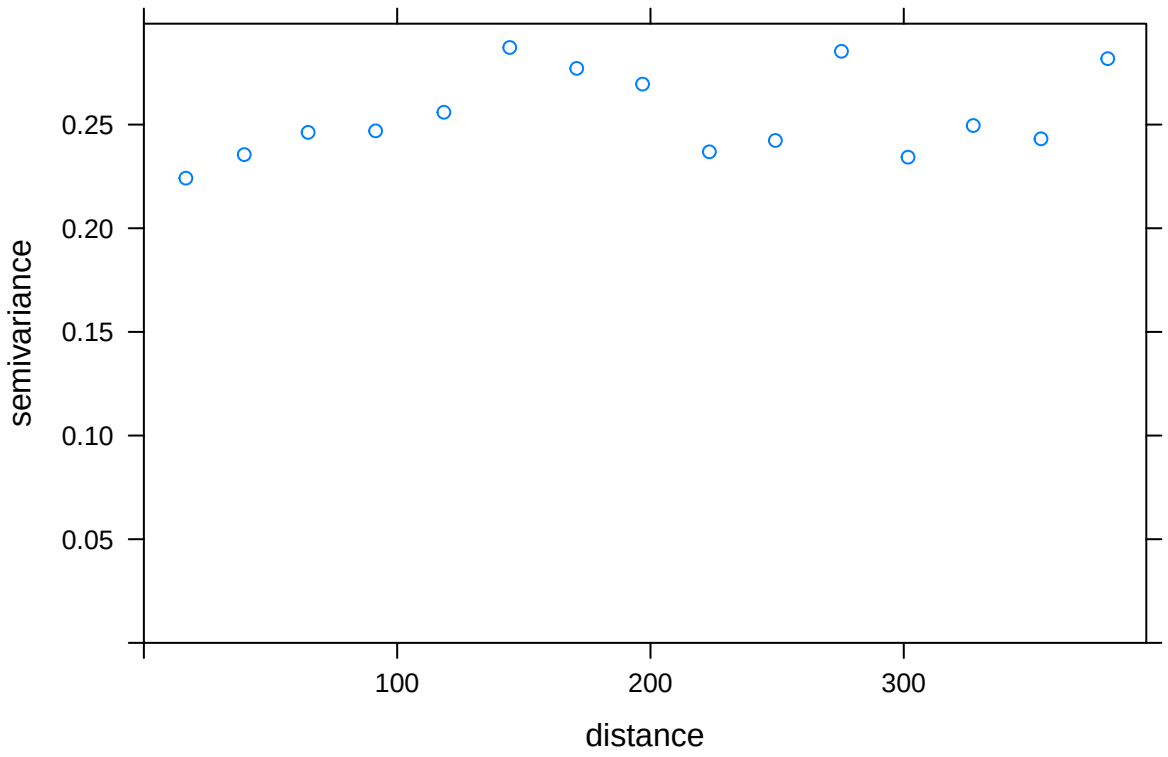
Kriging Kriging is a little more involved than IDW as it requires the construction of a semivariogram model to describe the spatial autocorrelation pattern for your particular variable.

```
variogcloud <- gstat::variogram(
  mag ~ 1, locations = earthquakes_sf, data = earthquakes_sf, cloud = TRUE
  )

plot(variogcloud)
```



The values in the cloud can be binned into lags with and plotted with a very similar function

```
semivariog <- gstat::variogram(
  mag ~ 1, locations = earthquakes_sf, data = earthquakes_sf
  )

plot(semivariog)
```




```
semivariog
```

```
##       np      dist     gamma dir.hor dir.ver   id
## 1   3283  16.57218 0.2241624       0       0 var1
## 2   6327  39.62268 0.2355224       0       0 var1
## 3   5731  64.84868 0.2462459       0       0 var1
## 4   4475  91.49887 0.2469810       0       0 var1
## 5   4281 118.44562 0.2559374       0       0 var1
## 6   4504 144.43792 0.2871903       0       0 var1
## 7   4667 170.88331 0.2771020       0       0 var1
## 8   4748 196.86472 0.2695293       0       0 var1
## 9   4405 223.25544 0.2368593       0       0 var1
## 10 4613 249.30448 0.2423462       0       0 var1
## 11 3895 275.39749 0.2853641       0       0 var1
## 12 3374 301.68595 0.2342840       0       0 var1
## 13 2512 327.41938 0.2495442       0       0 var1
## 14 1783 354.17895 0.2431448       0       0 var1
## 15 1452 380.53028 0.2818113       0       0 var1
```

From the empirical semivariogram plot and the information contained in the 'semivariog" *gstat* object, we can estimate the sill, range and nugget to use in our model semivariogram.

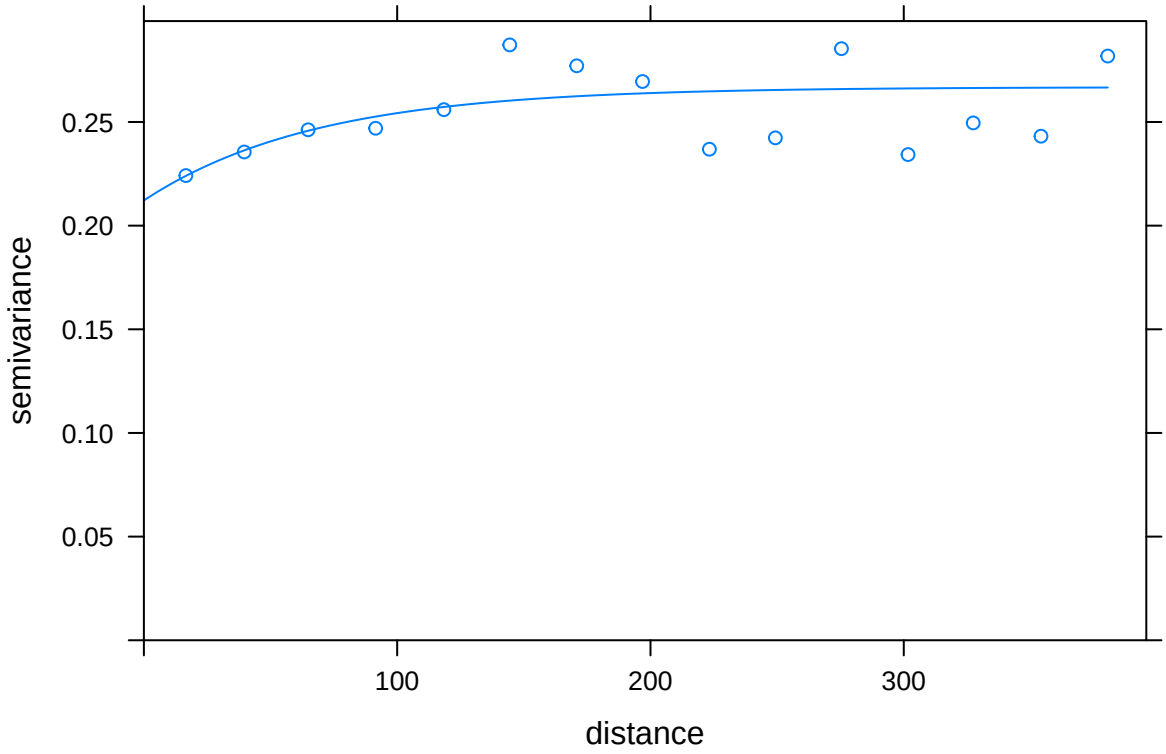
- In this case, the **range** (the point on the distance axis where the semivariogram starts to level off) is around the value of the last lag – 380.53028 – so we'll use Range = 380
- The **Sill** (the point on the y axis where the semivariogram starts to level off) is around 0.25
- The **nugget** looks to be around 0.20 (so the partial sill is around 0.2).

Using this information we'll generate a model semivariogram using the `vgm()` function in *gstat*.

```
model.variog <- gstat::vgm(psill = 0.25, model = "Exp", nugget = 0.2, range = 380)
```

We can now fit this model to a sample variogram to see how well it fits and plot it

```
fit.variog <- gstat::fit.variogram(semivariog, model.variog)
plot(semivariog, fit.variog)
```



Use the `krige()` function in gstat along with the model semivariogram just generated to generate an ordinary/simple Kriged surface – again, check `?krige` to see what the various options in the function are.

```
krige_res <- gstat::krige(
  formula = mag ~ 1, locations = earthquakes_sf, newdata = grid, model = model.variog
  )
```

```
## [using ordinary kriging]
```

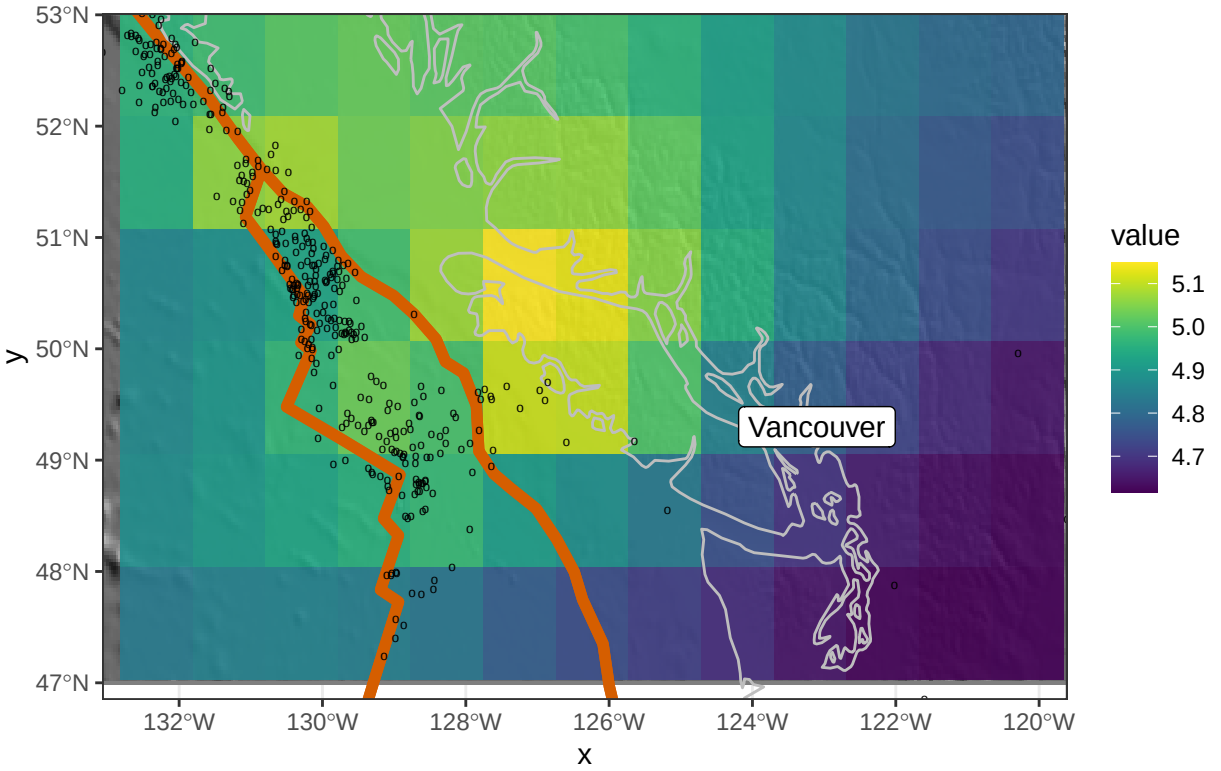
```
krige_rast <- terra::rasterize(
  terra::vect(krige_res),
  terra::rast(terra::vect(grid)),
  field = "var1.pred")
```

Generate a plot of the kriged surface in ggplot2 in a similar way to before

```
ggplot() +
  tidyterra::geom_spatraster(data = hill, show.legend = FALSE) +
  scico::scale_fill_scico(palette = "grayC") +
  ggnewscale::new_scale_fill() +
  tidyterra::geom_spatraster(data = krige_rast, alpha = .9) +
  viridis::scale_fill_viridis() +
  geom_sf(
    data = plates,
    lwd = 2,
    color = "#D55E00",
    fill = NA
  ) +
  geom_sf(data = coastlines, color = "grey") +
```

```
  geom_sf(data = earthquakes_sf, shape = "o") +
  geom_sf(data = vancouver) +
  geom_sf_label(data = vancouver, aes(label = city)) +
  coord_sf(
    default_crs = "WGS84",
    xlim = range(earthquakes_sf$longitude),
    ylim = range(earthquakes_sf$latitude),
    expand = FALSE
  ) +
  theme_bw()
```

```
## Warning in st_point_on_surface.sfc(sf::st_zm(x)): st_point_on_surface may not
## give correct results for longitude/latitude data
```



This completes your very short guide to creating spatial surfaces in R.

Useful resources

- Raster plotting: https://erinbecker.github.io/r-raster-vector-geospatial/02-raster-plot/index.html
- Crop a raster: https://www.earthdatascience.org/courses/earth-analytics/lidar-raster-data-r/crop-raster-data-in-r/
- Raster reprojection: https://datacarpentry.org/r-raster-vector-geospatial/03-raster-reproject-in-r/
- Spatial interpolation with R: https://rspatial.org/raster/analysis/4-interpolation.html
- Kriging with R: https://rstudio-pubs-static.s3.amazonaws.com/46259_d328295794034414944deea60552a942.html
- Swath profile: https://jjvhab.github.io/swathR/
- "Beautify" a map : https://timogrossenbacher.ch/2016/12/beautiful-thematic-maps-with-ggplot2-only/

Content | previous course: Geochronology