…Let’s start at the beginning. In general, countries have multiple levels of provincial government. For example, any neighborhood in Canada is going to have three layers/levels of government: a Federal government, based in Ottawa, Level 0; a pronvincial government, let’s say Alberta, which is Level 1; and a municipal government, with a town council and the like. Let’s say we start with Edmonton, which in this case is going to be Level 2. What does a map of Canada look like at that level of granularity? First, download the data from GADM as follows. I also download Level 1 and Level 2 :

```
library(rgeos)
library(maptools)
library(ggmap)
library(ggplot2)
library(sp)
library(rgdal)
library(raster) #Various packages for GIS and mapping
library(plyr)
#This one is for database manipulations like "join"
library(nasapower)
#To get the solar insolation and other meteorological data
library(scales)
#This one allows us to do things like have nice scales for the legends
on the images we create


#Download the files for Levels 1 and 2 SPDFs covering Canada
can_level1 = getData("GADM", country = "CA", level = 1)
can_level2 = getData("GADM", country = "CA", level =2)
```

Importantly, my two objects above “can_level1” and “can_level2” are SpatialPolygonsDataFrame objects, and they will inherit some features of the DataFrame object’s behaviour. In the above, I also called all of the packages I might need to run all of the codes below, just to get things out of the way. So what does a Province-level map of Canada look like, when printed as a familiar pink?

```
plot(can_level1, col = "pink", bg = "lightblue")
```

The picture is not too surprising when you post it.

…and just in case my super exact, very academic description of where Alberta lies didn't ring any bells, here we're going to select the bits of Canada that are Alberta, and then just plot them. It helps at first to know what the internal structure of the Canada Level 1 object is.

```
> names(can_level1)
 [1] "GID_0"     "NAME_0"    "GID_1"     "NAME_1"    "VARNAME_1"
"NL_NAME_1" "TYPE_1"
 [8] "ENGTYPE_1" "CC_1"      "HASC_1"
```

As you may have guessed, the "Name 1" metadata column fills in the names of the provinces.

```
> can_level1$NAME_1
 [1] "Alberta"                   "British Columbia"          "Manitoba"
 [4] "New Brunswick"             "Newfoundland and Labrador" "Northwest
Territories"
 [7] "Nova Scotia"               "Nunavut"                   "Ontario"
[10] "Prince Edward Island"      "Québec"
"Saskatchewan"
[13] "Yukon"
```

We want to select all the bits of Canada which belong to Alberta and, since the SPDFs inherit DataFrame behaviour, we can do this the same way we subset a dataframe.

```
> alberta_level1 = can_level1[which(can_level1$NAME_1 == "Alberta"),]
```

Finally, we would like to plot Alberta just to make sure that we got it right. The only difference from the previous plotting is that I use the "add = TRUE" to the plot function to make sure that I don't remove all the other provinces. It makes sense also to specify a new colour, just to be sure that Alberta can be clearly seen.

```
> plot(alberta_level1, col = "green", add = TRUE)
```

The very straightforward result is below.

What good is any of this data though if we can visualise something useful with it? First off, I'm going to stretch my maps into a proper ellipsoid shape; with Alberta being so far from the equator, it makes sense to understand that a geographic shape region which might seem like a perfect square in the above map is actually a bit more "squished". (You can read more about projections, here). For the sake of our discussion, I will use the WGS84 transformation, which is widely used for things like Google Maps (discussions about which projections may or may not be better for a given location are a bit ahead right now; but yes, you might want to choose a different projection). The below code shows how it's done:

```
WGS84 <- CRS("+proj=longlat +ellps=WGS84 +datum=WGS84 +no_defs")
canada_level2_ellipsoid = spTransform(can_level2, WGS84)
```

Next, we want to only select the bits of that projection which fit our needs:

```
alberta_ellipsoid = canada_level2_ellipsoid[which(
canada_level2_ellipsoid$NAME_1 == "Alberta"),]
```

If you play around a bit with these objects, you find an easy way to access the longitude and latitude of each sub-division, using the gCentroid function (wtihin the sp package). To collect all of the longitudes and latitudes associated with each of the sub-divisions in a single data frame, use the following:

```
alberta_coordinates = data.frame(matrix(ncol = 2, nrow =
length(alberta_ellipsoid$NAME_2)))
for(i in 1:length(alberta_ellipsoid$NAME_2))
{
  alberta_coordinates[i,] = data.frame(gCentroid(alberta_
ellipsoid[i,]))
}
```

After that, the stuff we came here for: we want not only a data.frame containing the solar insolation at each centroid. I've written a bit about NASAPOWER, possibly my favourite CRAN package bar none, over here, but you can also just read the CRAN description: https://cran.r-project.org/web/packages/nasapower/index.html More immediately, you could try looking at the resulting data.frame resulting from the following line of code:

```
ghi_retrieved = nasapower::get_power(community = "SSE", pars =
"ALLSKY_SFC_SW_DWN", lonlat = c(alberta_coordinates[1,1],
alberta_coordinates[1,2]), temporal_average = "CLIMATOLOGY")
```

Here, "ALLSKY_SFC_SW_DWN" retrieves the GHI or "Global Horizontal Irradiation; and "Climatology" defines the inbuilt timeframe from 1-1-1984 to 31-12-2013. One important thing you'll notice is that the average daily insolation on a GHI basis is stored in column 16 of the returned data.frame (actually a tibble in the first instance from the above line). So we will build an all-Alberta data frame containing the average yearly insolation during the "CLIMATOLOGY" timeframe thus:

```
alberta_solar_insolation_annual = data.frame(matrix(ncol = 1, nrow =
nrow(alberta_coordinates)))
for(i in 1:nrow(alberta_coordinates))
{
  ghi_retrieved = nasapower::get_power(community = "SSE", pars =
"ALLSKY_SFC_SW_DWN", lonlat = c(alberta_coordinates[i,1],
alberta_coordinates[i,2]), temporal_average = "CLIMATOLOGY")
  alberta_solar_insolation_annual[i,] = data.frame(ghi_retrieved[16])*
365
  print("One more Census Division")
}
```

So, now we have a dataframe which contains the average insolation over a year (for the selected time frame) based on NASA satellite observations for the relevant centroids (longitude and latitude pairs as far as we're concerned). We're going to want to match up each insolation value to its relevant sub-district, so we are going to have attach a column with the names of the districts (Never mind that for Alberta, GADM seems to only have names like "Census Division X" etc):

```
alberta_solar_insolation_annual = cbind(unique(alberta_
ellipsoid@data$NAME_2), alberta_solar_insolation_annual)
#We set also the metadata
colnames(alberta_solar_insolation_annual) = c("NAME_2", "Insolation")
```

Now, notice that our ellipsoid object has an attached dataframe with all sorts of valuable information. It's accessed like so:

```
alberta_ellipsoid@data
```

You probably don't want to do that just yet; instead, just get an idea of the metadata of that dataframe:

```
names(alberta_ellipsoid@data)
 [1] "GID_0"       "NAME_0"      "GID_1"       "NAME_1"
"NL_NAME_1"   "GID_2"       "NAME_2"      "VARNAME_2"
 [9] "NL_NAME_2"   "TYPE_2"      "ENGTYPE_2"   "CC_2"        "HASC_2"
```

```
"GID_0.1"      "NAME_0.1"     "GID_1.1"
[17] "NAME_1.1"     "NL_NAME_1.1" "GID_2.1"      "VARNAME_2.1"
"NL_NAME_2.1" "TYPE_2.1"     "ENGTYPE_2.1" "CC_2.1"
[25] "HASC_2.1"
```

Some thing to make sure of is that you've got a "NAME_2" column in this dataframe. Make sure it contains the same values in the insolation dataframe, but note that they will generally not be in the same order. We can now join the insolation data to the dataframe for the Alberta ellipsoid. After that, we want to "fortify" that data to convert it into a data.frame, again, one which ggplot2 is able to deal with, and to lay it over a polygon:

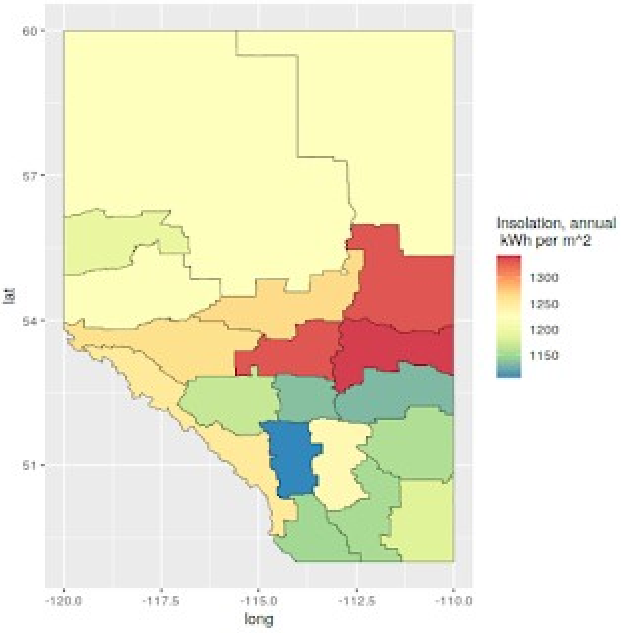
```
alberta_ellipsoid@data = join(alberta_ellipsoid@data, alberta_solar_df,
by = "NAME_2")

alberta_solar_df_fortified_B = fortify(alberta_ellipsoid)
```

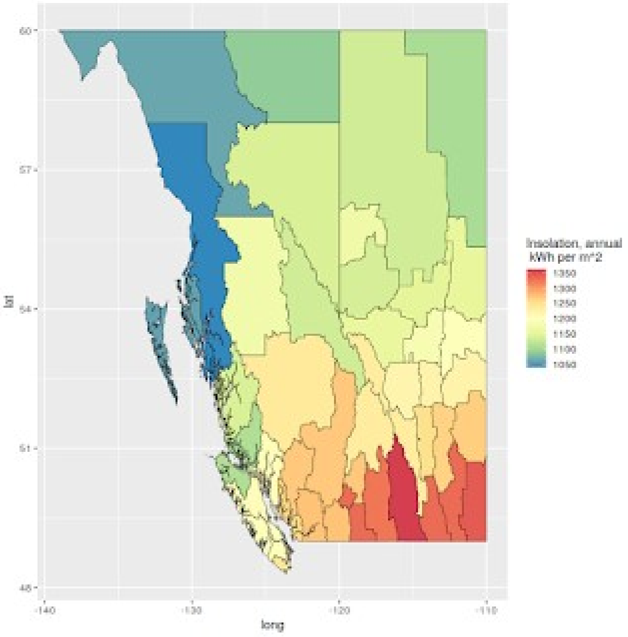
Check the names of this resulting object. You should notice, among other things, longitude, latitude and a NAME_2 as well as an Insolation value. Pro tip: the next step, when we produce a heatmap based on insolation, is going to be much, much easier if you make sure that the "Insolation" values are in a column with a one-word name. If you were to put them in a column named, say, "Annual insolation" you'd have to refer to it in the call to ggplot2 as "Annual insolation" and R will think you're trying to use factors as quantitative values, throwing up a "Discrete values to continuous scale" error which is not going to be easy to diagnose. Instead, try something like the following:

```
ggplot() + geom_polygon(data = alberta_solar_df_fortified_B, aes(x =
long, y = lat, group = group, fill = Insolation), size = 0.12) +
scale_fill_distiller(name="Insolation, annual \n kWh per m^2", palette
= "Spectral", breaks = 7)
```

I played around with the sizes until I settled on 0.12. I also knew a little bit about the data to begin with; I think 7 breaks for the pretty breaks (defined in the scale/legend filler, the third part of the call) is about right in this case. The resulting plot should look like this:



This looks pretty and all, but it's also not entirely conclusive of anything if it's not put into perspective. So I repeated the steps above, but instead of Level 2 districts in Alberta alone, I mapped them for the two provinces of Western Canada: Alberta and British Columbia, the latter of which is to the west (left) of the image. As can be clearly seen, Alberta is pretty much consistently more sunny than British Columbia.



Coming up in future posts, I will move on to try and map not only insolation but also more decision variable type values, such as PV yield. A yet more sophisticated approach involves also the intermittency of solar PV and how it matches, or doesn't match, electricity demand (i.e., without storage, PV electricity is no good if people only play video games and turn the lights on at night). That's for much later down the road!