Methods for Mapping Demographic Indexes for the Casco Bay Watershed

July 27, 2022

Contents

Introduction **1**

Load Libraries **2**

Set Graphics Theme **3**

Load Data **3**

Folder References 3

Tabular Data 3

Geospatial Data 3

Initial Plot **4**

Merge Tablular Data and Geospatial Data **6**

Revised Plot **7**

Creating A Faceted Map Display **8**

Building the Tibble 8

Facet Plot Demo 9

National Percentiles of Indicators **10**

Zoom in on Portland Region 12

Ranks of Indicators **13**

Zoom in on Portland Region 14

Introduction

CBEP, like other National Estuary Programs will receive additional funding to support our programs via the "Bipartisan Infrastructure Law" signed into law last December.

EPA has recently released guidance for applying for those funds. A core component of the guidance is that overall, the NEP program should comply with the White House's "Justice 40" initiative, which requires that "at least 40% of the benefits and investments from BIL funding flow to disadvantaged communities."

EPA suggested that we use the National-scale EJSCREEN tools to help identify "disadvantaged communities" in our region. The EPA guidance goes on to suggest we focus on five demographic indicators:

- Percent low-income;
- Percent linguistically isolated;
- Percent less than high school education;
- Percent unemployed; and
- Low life expectancy.

This notebook builds on the work in "Calc_Indexes.pdf" to calculate data for the Casco Bay Watershed Census Tracts, and calculates how Casco Bay Census tracts compare at National, Statewide, and Regional scales.

Load Libraries

```
library(tidyverse)
#> -- Attaching packages --------------------------------------- tidyverse 1.3.1 --
#> v ggplot2 3.3.6     v purrr   0.3.4
#> v tibble  3.1.7     v dplyr   1.0.9
#> v tidyr   1.2.0     v stringr 1.4.0
#> v readr   2.1.2     v forcats 0.5.1
#> -- Conflicts ------------------------------------------ tidyverse_conflicts() --
#> x dplyr::filter() masks stats::filter()
#> x dplyr::lag()    masks stats::lag()
library(GGally)
#> Registered S3 method overwritten by 'GGally':
#>   method from
#>   +.gg   ggplot2
library(readr)
library(rgdal)
#> Loading required package: sp
#> Please note that rgdal will be retired by the end of 2023,
#> plan transition to sf/stars/terra functions using GDAL and PROJ
#> at your earliest convenience.
#>
#> rgdal: version: 1.5-32, (SVN revision 1176)
#> Geospatial Data Abstraction Library extensions to R successfully loaded
#> Loaded GDAL runtime: GDAL 3.4.3, released 2022/04/22
#> Path to GDAL shared files: C:/Users/curtis.bohlen/AppData/Local/R/win-library/4.2/rgdal/gdal
#> GDAL binary built with GEOS: TRUE
#> Loaded PROJ runtime: Rel. 7.2.1, January 1st, 2021, [PJ_VERSION: 721]
#> Path to PROJ shared files: C:/Users/curtis.bohlen/AppData/Local/R/win-library/4.2/rgdal/proj
#> PROJ CDN enabled: FALSE
#> Linking to sp version:1.5-0
#> To mute warnings of possible GDAL/OSR exportToProj4() degradation,
```

```
#> use options("rgdal_show_exportToProj4_warnings"="none") before loading sp or rgdal.
library(sf)    # automatically loads `sp` and `rgdal`
#> Linking to GEOS 3.9.1, GDAL 3.4.3, PROJ 7.2.1; sf_use_s2() is TRUE
library(broom)  # used to tidy geospatial info to dataframe for ggplot (could use fortify())
#library(rgeos)
```

Set Graphics Theme

This sets `ggplot()`graphics for no background, no grid lines, etc. in a clean format suitable for (some) publications.

```
theme_set(theme_classic())
```

Load Data

Folder References

I use folder references to allow limited indirection, thus making code from GitHub repositories more likely to run "out of the box".

```
data_folder <- "Original_Data"
gis_folder <- "GIS_Data"
dir.create(file.path(getwd(), 'figures'), showWarnings = FALSE)
```

I use the "Original_Data" folder to retain data in the form originally downloaded. That minimizes the chances of inadvertently modifying the source data. All data was accessed via EJScreen. The 2021 EJSCREEN Data was accessed on July 26, 2022, at https://gaftp.epa.gov/EJSCREEN/2021/. I downloaded geodatabases, and open the geospatial data they contained in ArcGIS and exported the tabular attribute data to CSV files. That tabular CSV data is provided in the "Original Data" folder here.

The "figures" folder isolates "final" versions of any graphics I produce. That just makes it a bit easier to find final products in what can sometimes be fairly large GitHub Repositories (although not here).

Tabular Data

The Tabular data is quite extensive (over 165 MB), which poses some data access challenges. The raw CSV file contains 74001 records, and 166 columns. Most, but not all are numeric. The Health data is slightly smaller in length, and has only a handful of relevant data columns, but it DOES include State ad County names, which are more convenient that the GEOID10 values to search.

```
cb_data <- read_csv("cb_tracts_indexes.csv",
                    col_types = paste0('ccddd--', rep('d', 41)))
```

Geospatial Data

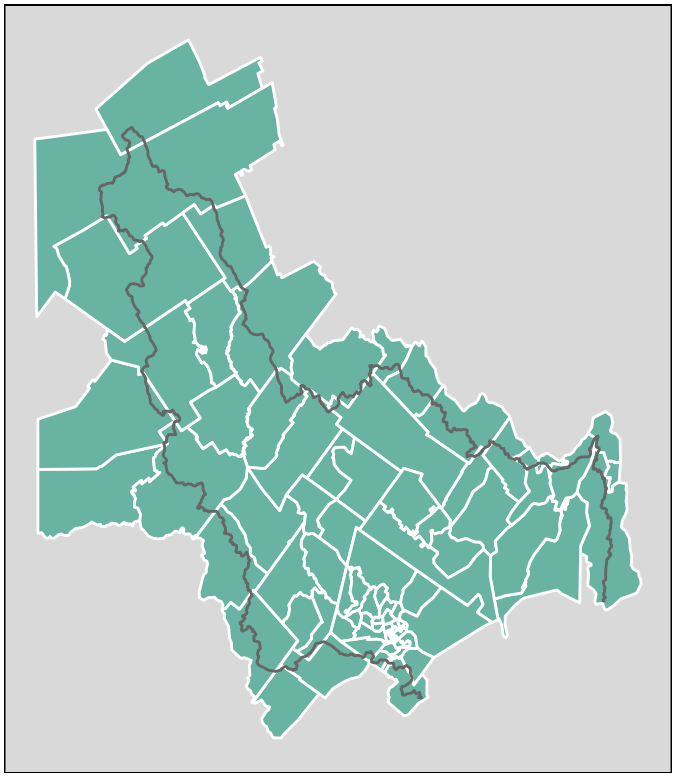
We have geospatial data, in UTM coordinates.

```
the_file_name <- 'casco_ejscreen_utm.shp'
the_path <- file.path(gis_folder, the_file_name)
cb_geospatial <- readOGR(the_path )
#> OGR data source with driver: ESRI Shapefile
#> Source: "C:\Users\curtis.bohlen\Documents\Data Analysis\Casco_EJScreen\GIS_Data\casco_ejscreen_utm.s
#> with 80 features
#> It has 8 fields
#> Integer64 fields read as strings:  InPoly_FID

the_file_name <- 'casco_watershed_utm.shp'
the_path <- file.path(gis_folder, the_file_name)
cb_watershed <- readOGR(the_path )
#> OGR data source with driver: ESRI Shapefile
#> Source: "C:\Users\curtis.bohlen\Documents\Data Analysis\Casco_EJScreen\GIS_Data\casco_watershed_utm.
#> with 817 features
#> It has 9 fields
#> Integer64 fields read as strings:  OBJECTID InPoly_FID

the_file_name <- 'watershed_outline.shp'
the_path <- file.path(gis_folder, the_file_name)
cb_watershed_outline <- readOGR(the_path )
#> OGR data source with driver: ESRI Shapefile
#> Source: "C:\Users\curtis.bohlen\Documents\Data Analysis\Casco_EJScreen\GIS_Data\watershed_outline.sh
#> with 1 features
#> It has 1 fields
#> Integer64 fields read as strings:  FID_Mainla

the_file_name <- 'Maine.shp'
the_path <- file.path(gis_folder, the_file_name)
Maine <- readOGR(the_path )
#> OGR data source with driver: ESRI Shapefile
#> Source: "C:\Users\curtis.bohlen\Documents\Data Analysis\Casco_EJScreen\GIS_Data\Maine.shp", layer: "
#> with 1 features
#> It has 5 fields
#> Integer64 fields read as strings:  InPoly_FID
```

Initial Plot

Note that if you pass the geospatial layer to `ggplot` without manually converting it (using `tidy()` from `broom`, or `fortify()`, buried in `ggplot2`, `ggplot()` still figures out what to do, but makes some default decisions about how to define the polygons that may or may not be correct. For a sample map, as here, they appear to work pretty well.

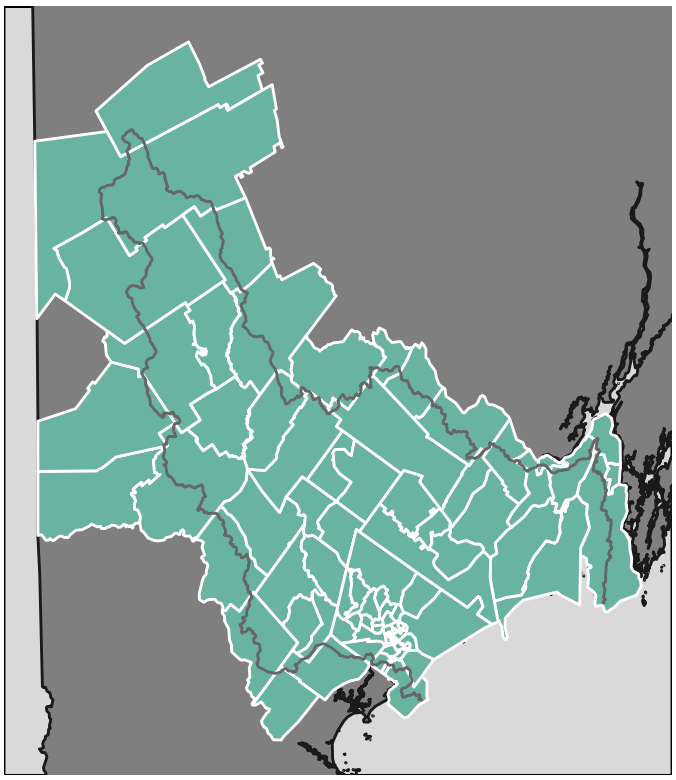
```
plt <- ggplot() +
  geom_polygon(data = cb_geospatial, aes( x = long, y = lat, group = group), fill="#69b3a2", color="whi
  geom_path(data = cb_watershed_outline, aes( x = long, y = lat, group = group),
            color="grey40") +
  theme_void() +
  theme(panel.background = element_rect(fill = 'grey85')) +
  coord_equal()
#> Regions defined for each Polygons
plt
```



That's a good start, but it would be nice to provide a Maine state background. To do that, we need to find the current coordinates and apply them after we add Maine (which is larger) to the map.

```
xlims <- layer_scales(plt)$x$range$range
ylims <- layer_scales(plt)$y$range$range
```

```
plt <- ggplot() +
  geom_polygon(data = Maine, aes( x = long, y = lat, group = group),
               fill = 'grey50', color='grey10')  +
  geom_polygon(data = cb_geospatial, aes( x = long, y = lat, group = group),
               fill="#69b3a2", color="white") +
  geom_path(data = cb_watershed_outline, aes( x = long, y = lat, group = group),
               fill = NA, color="grey40") +

  theme_void() +
  theme(panel.background = element_rect(fill = 'grey85')) +
  coord_equal(xlim = xlims, ylim = ylims)
#> Regions defined for each Polygons
#> Regions defined for each Polygons
#> Warning: Ignoring unknown parameters: fill
plt
```



Merge Tablular Data and Geospatial Data

The object the shapefile has become (Technically, a "SpatialPolygonsDataFrame") contains a dataframe in its `@data` slot, but manipulating it directly appears to lead to unpredictable behavior. However, `sp` includes a `merge()` method for these S4 objects, when paired with a dataframe. This works as one might expect. (Note that `sp` also contains an `as.data.frame()` method that extracts this information without requiring access to the slot directly.)

```
names(cb_geospatial@data)
#> [1] "GEOID10"    "County"     "Shape_Leng" "Shape_Area" "InPoly_FID"
#> [6] "SimPgnFlag" "MaxSimpTol" "MinSimpTol"
```

```
tmp <- cb_data %>%
  select(c('GEOID10', "LIFEEXP":"UNEMPPCT", -LIFEEXP_SE, NEG_LIFEEXP, Index_1:p_Index_2))
```

But we run into problems creating a "fortified" data frame for ggplot that contains the identifier we need in the next step to allow us to merge the geospatial data with the tabular data.

```
cb_geospatial_df <- tidy(cb_geospatial, region = "GEOID10")
```

```
cb_geospatial_df <- tidy(cb_geospatial, region = "GEOID10")
```

A little Googling revealed that `maptools` (apparently called by `sp`, which is called by either `sf` or `rgdal`) relies on other packages to handle polygon geometries. Calling either `tidy()` or `fortify()` on a polygon map layer with a `region` argument triggers calls to functions from one of these other packages.

The help files for `maptools::checkPolygonHoles()` provides a bit more detail.

Apparently, `gpclip` has a restrictive license, so you need to formally grant permission to `gpclip`. but the help file says use of the `rgeos` package functions is preferred.

So, after I installed and loaded the `rgeos` package, all is well... But note the warning about future changes. Since I have never called any `rgeos` functions directly, I'm not sure how we will need to change code in the future.

```
library(rgeos)
#> rgeos version: 0.5-9, (SVN revision 684)
#>  GEOS runtime version: 3.9.1-CAPI-1.14.2
#>  Please note that rgeos will be retired by the end of 2023,
#> plan transition to sf functions using GEOS at your earliest convenience.
#>  GEOS using OverlayNG
#>  Linking to sp version: 1.5-0
#>  Polygon checking: TRUE

cb_geospatial_df <- tidy(cb_geospatial, region = "GEOID10")
```

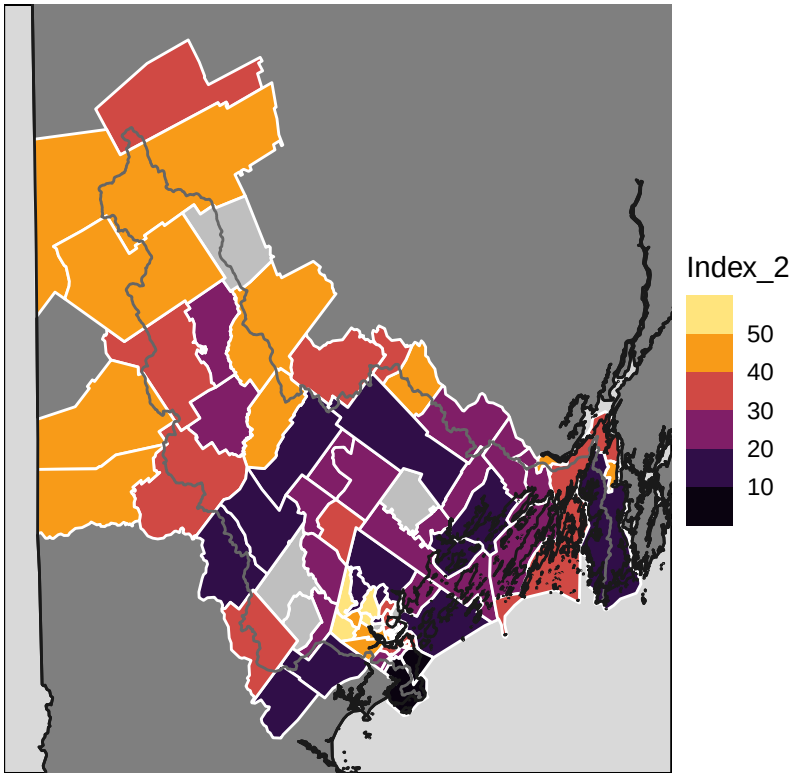
```
cb_geospatial_df <- cb_geospatial_df %>%
  left_join(tmp, by = c('id' = 'GEOID10'))
```

```
names(cb_geospatial_df)
#>  [1] "long"        "lat"         "order"       "hole"        "piece"
#>  [6] "group"       "id"          "LIFEEXP"     "LOWINCPCT"   "LESSHSPCT"
#> [11] "LINGISOPCT"  "UNEMPPCT"    "NEG_LIFEEXP" "Index_1"     "Index_2"
#> [16] "p_Index_1"   "p_Index_2"
```

Revised Plot

With severa lother improvements...

```
plt <- ggplot() +
  geom_polygon(data = Maine, aes( x = long, y = lat, group = group),
               fill = 'grey50', color='grey10')  +
  geom_polygon(data = cb_geospatial_df, aes( x = long, y = lat, group = group,
                                             fill = Index_2), color="white") +
  geom_path(data = cb_watershed_outline, aes( x = long, y = lat, group = group),
              fill = NA, color="grey40") +
  geom_path(data = Maine, aes( x = long, y = lat, group = group),
               fill = 'grey50', color='grey10')  +
  scale_fill_viridis_b(option = 'B', na.value = "grey75") +
  theme_void() +
  theme(panel.background = element_rect(fill = 'grey85')) +
  coord_equal(xlim = xlims, ylim = ylims)
#> Regions defined for each Polygons
#> Warning: Ignoring unknown parameters: fill
#> Regions defined for each Polygons
#> Warning: Ignoring unknown parameters: fill
plt
```



Creating A Faceted Map Display

Building the Tibble

To create a `facet_wrap()` display on multiple panels, I need to build a suitable long tibble. It's not entirely obvious how to go about that.

We start with a map showing which Census Tracts in the Casco Bay region exceed the (regional) 80th percentile on each of my proposed thresholds.

Only two census Tracts exceed the Maine 80th percentile threshold, and none exceed the Federal 80th percentile thresholds. That makes maps of those values fairly useless. Fourteen census tracts exceed the regional 80th percentile threshold (by definition of the 80th percentile).

```
tmp <- cb_data %>%
  select(GEOID10, Ex_CB_Index_1:Ex_CB_PCA_Index_V2)
```

First, we create a "fortified" tibble, as required for mapping with `ggplot()`.

```
cb_geospatial_df <- tidy(cb_geospatial, region = "GEOID10")
cb_geospatial_df <- cb_geospatial_df %>%
  left_join(tmp, by = c('id' = 'GEOID10'))
```

Then we simplify the column names.

```
cb_geospatial_df_2 <- cb_geospatial_df %>%
  rename_with(function(x) sub('Ex_CB_', '', x))
```

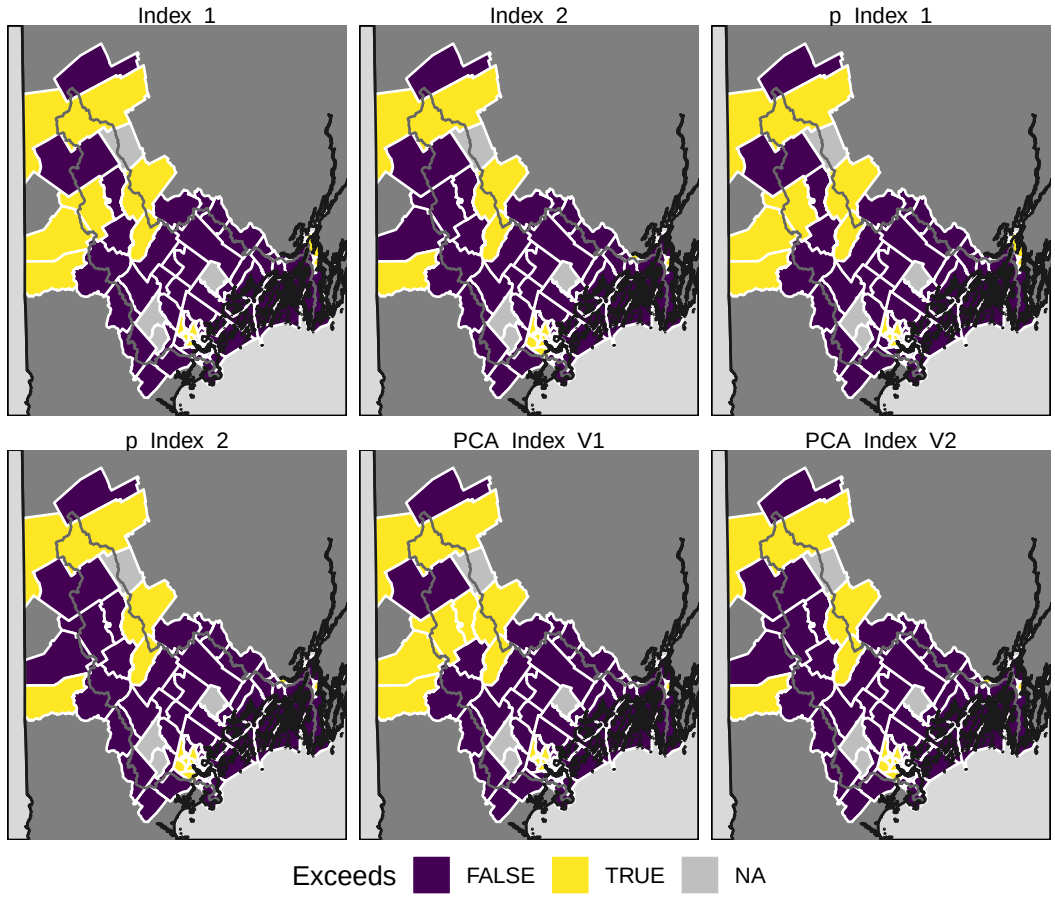
And finally, pivot the fortified data frame into long format. Note that this creates a **large** data frame which we probably don't want to keep around any longer than necessary.

```
cb_geospatial_df_2 <- cb_geospatial_df_2 %>%
  pivot_longer(Index_1:PCA_Index_V2,
               names_to = 'Threshold', values_to = 'Value')
```

Facet Plot Demo

This ggplot object is fairly complex, so it takes some time to render. Rendering time can be shortened somewhat by using simpler polygon geometries.

```
plt <- ggplot() +
  geom_polygon(data = Maine, aes( x = long, y = lat, group = group),
                fill = 'grey50', color='grey10')  +
  geom_polygon(data = cb_geospatial_df_2, aes( x = long, y = lat, group = group,
                                               fill = Value), color="white") +
  geom_path(data = cb_watershed_outline, aes( x = long, y = lat, group = group),
               fill = NA, color="gray40") +
  geom_path(data = Maine, aes( x = long, y = lat, group = group),
                fill = 'grey50', color='grey10')  +

  facet_wrap(~ Threshold, nrow = 2) +

  scale_fill_viridis_d(option = 'D', na.value = "grey75", name = 'Exceeds') +
  theme_void() +
  theme(panel.background = element_rect(fill = 'grey85'),
        legend.position = 'bottom') +
  coord_equal(xlim = xlims, ylim = ylims)
#> Regions defined for each Polygons
#> Warning: Ignoring unknown parameters: fill
#> Regions defined for each Polygons
#> Warning: Ignoring unknown parameters: fill
plt
```

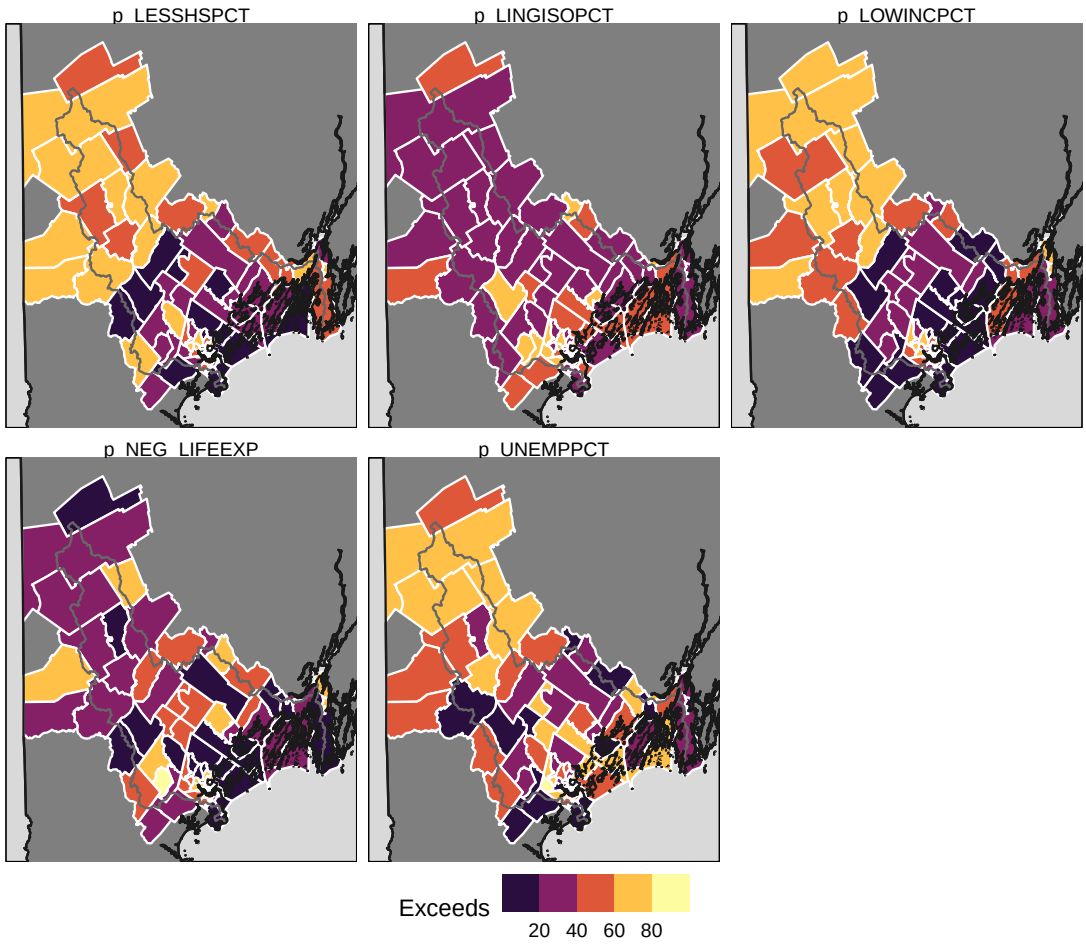


What that shows is that thresholds based on indexes constructed by looking at raw scores generally identify more mid-watershed areas, while indexes based on national percentiles tend to identify more locations in Portland and South Portland. This presumably reflects the greater weighting of income in the former, compared to other metrics of disadvantage.

National Percentiles of Indicators

```
tmp <- cb_data %>%
  select(GEOID10, p_NEG_LIFEEXP:p_UNEMPPCT) %>%
  mutate(across(p_NEG_LIFEEXP:p_UNEMPPCT, rank,
         na.last = TRUE, ties.method = 'average'))

cb_geospatial_df <- tidy(cb_geospatial, region = "GEOID10")
cb_geospatial_df <- cb_geospatial_df %>%
  left_join(tmp, by = c('id' = 'GEOID10'))

cb_geospatial_df_2 <- cb_geospatial_df %>%
  pivot_longer(p_NEG_LIFEEXP:p_UNEMPPCT,
               names_to = 'Indicator', values_to = 'Value')
```

```
plt <- ggplot() +
  geom_polygon(data = Maine, aes( x = long, y = lat, group = group),
               fill = 'grey50', color='grey10')  +
  geom_polygon(data = cb_geospatial_df_2, aes( x = long, y = lat, group = group,
                                               fill = Value), color="white") +
    geom_path(data = cb_watershed_outline, aes( x = long, y = lat, group = group),
              fill = NA, color="grey40") +
  geom_path(data = Maine, aes( x = long, y = lat, group = group),
               fill = 'grey50', color='grey10')  +

  facet_wrap(~ Indicator, nrow = 2) +

  scale_fill_viridis_b(option = 'B', na.value = "grey75", name = 'Exceeds') +
  theme_void() +
  theme(panel.background = element_rect(fill = 'grey85'),
        legend.position = 'bottom') +
  coord_equal(xlim = xlims, ylim = ylims)
#> Regions defined for each Polygons
#> Warning: Ignoring unknown parameters: fill
#> Regions defined for each Polygons
#> Warning: Ignoring unknown parameters: fill
plt
```

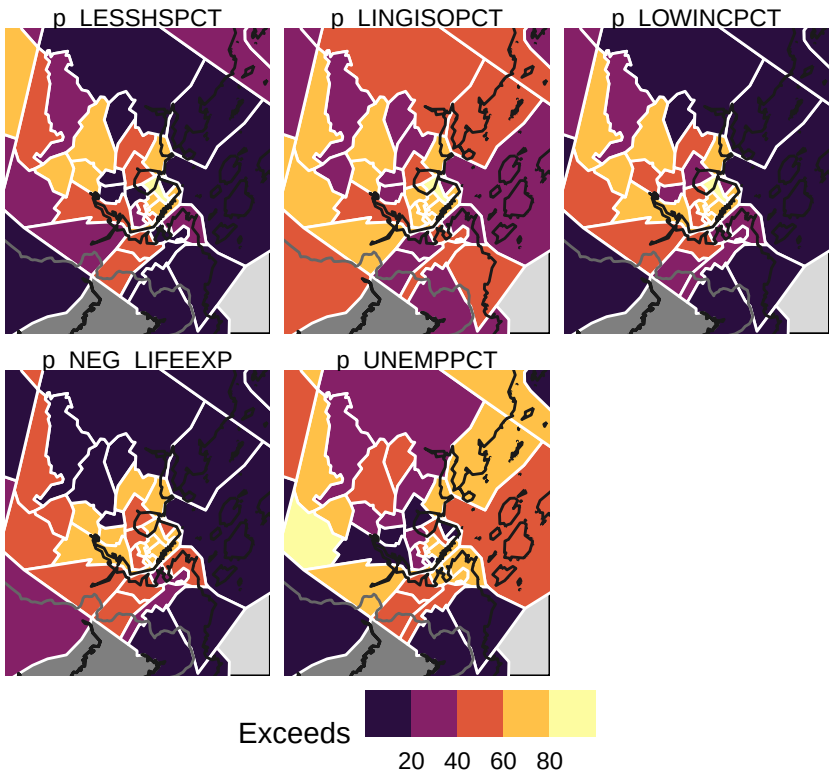


Zoom in on Portland Region

```
xlims
#> [1] 339455.5 438494.2
ylims
#> [1] 4817293 4931318
```

```
new_x_low <- xlims[1] + 3* (xlims[2]-xlims[1])/6
new_x_high <- xlims[1] + 4* (xlims[2]-xlims[1])/6

new_y_low <- ylims[1] +  (ylims[2]-ylims[1])/12
new_y_high <- ylims[1] + 3*(ylims[2]-ylims[1])/12

new_xlims <- c(new_x_low, new_x_high)
new_ylims <- c(new_y_low, new_y_high)
```

```
 plt +
  coord_equal(xlim = new_xlims, ylim = new_ylims)
#> Coordinate system already present. Adding new coordinate system, which will replace the existing one
```



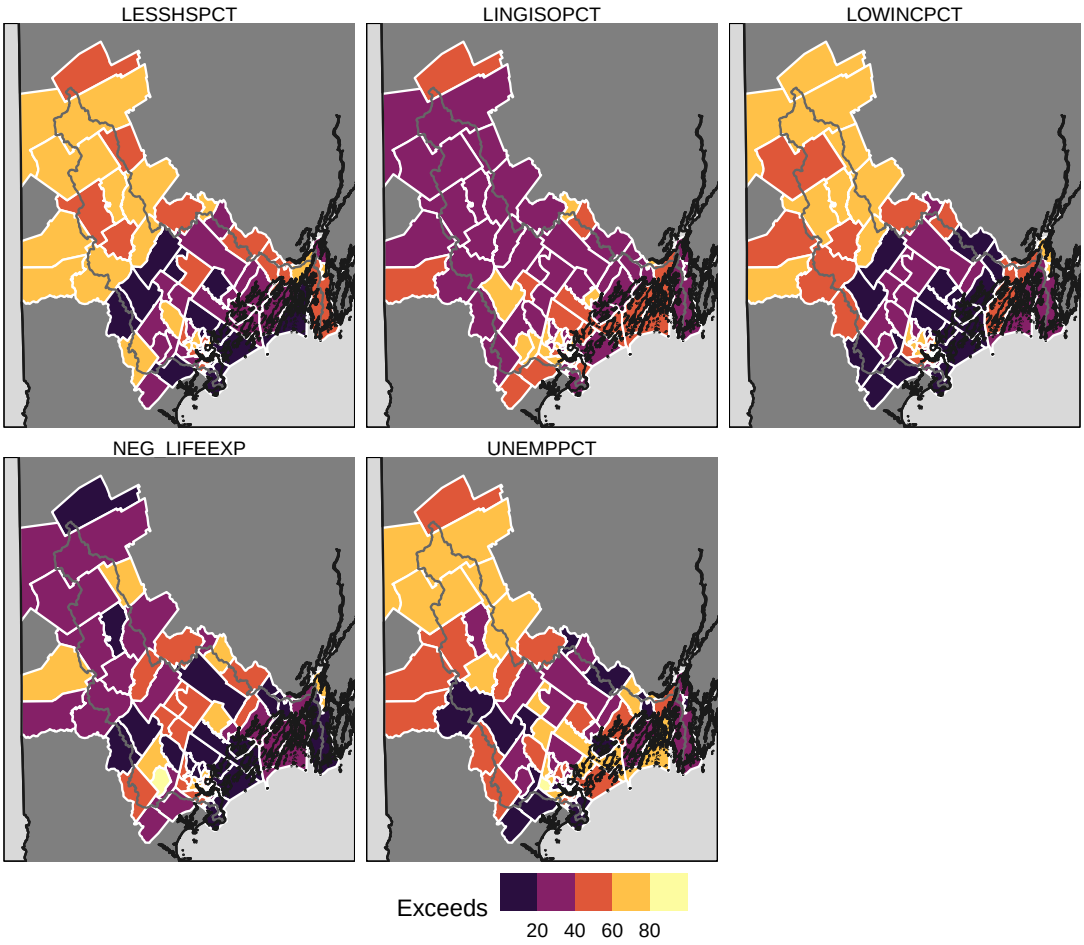
So, several of these pick up the subsidized housing and homeless community in Portland's Back Cove neighborhood. Other than that, what is striking is the fact that the ranks of the different metrics are not all that correlated in this smaller sub-region, which may help explain why few locations are flagged by any of the candidate indicators.

Ranks of Indicators

Let's look at rank ordering within each raw indicators.

```
tmp <- cb_data %>%
  select(GEOID10, LOWINCPCT:UNEMPPCT, NEG_LIFEEXP) %>%
  mutate(across(LOWINCPCT:NEG_LIFEEXP, rank,
         na.last = TRUE, ties.method = 'average'))

cb_geospatial_df <- tidy(cb_geospatial, region = "GEOID10")
cb_geospatial_df <- cb_geospatial_df %>%
  left_join(tmp, by = c('id' = 'GEOID10'))

cb_geospatial_df_2 <- cb_geospatial_df %>%
  pivot_longer(LOWINCPCT:NEG_LIFEEXP,
               names_to = 'Indicator', values_to = 'Value')
```

```
plt <- ggplot() +
  geom_polygon(data = Maine, aes( x = long, y = lat, group = group),
                fill = 'grey50', color='grey10')  +
  geom_polygon(data = cb_geospatial_df_2, aes( x = long, y = lat, group = group,
                                          fill = Value), color="white") +
  geom_path(data = cb_watershed_outline, aes( x = long, y = lat, group = group),
               fill = NA, color="grey40") +
  geom_path(data = Maine, aes( x = long, y = lat, group = group),
                fill = 'grey50', color='grey10')  +

  facet_wrap(~ Indicator, nrow = 2) +

  scale_fill_viridis_b(option = 'B', na.value = "grey75", name = 'Exceeds') +
  theme_void() +
  theme(panel.background = element_rect(fill = 'grey85'),
        legend.position = 'bottom') +
  coord_equal(xlim = xlims, ylim = ylims)
#> Regions defined for each Polygons
#> Warning: Ignoring unknown parameters: fill
#> Regions defined for each Polygons
#> Warning: Ignoring unknown parameters: fill
plt
```

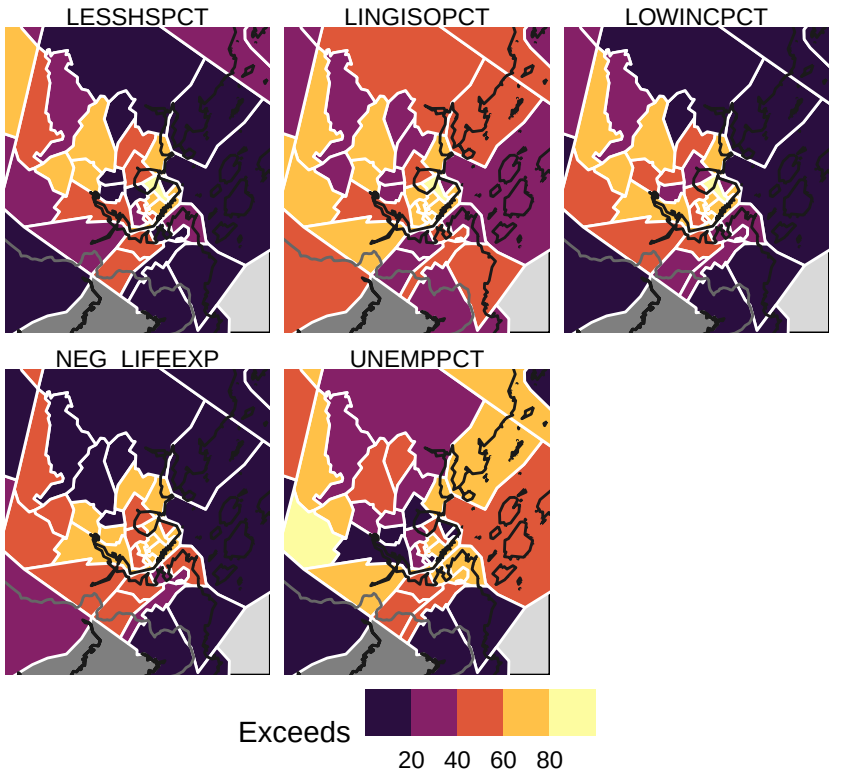


Zoom in on Portland Region

```
xlims
#> [1] 339455.5 438494.2
ylims
#> [1] 4817293 4931318
```

```
new_x_low <- xlims[1] + 3* (xlims[2]-xlims[1])/6
new_x_high <- xlims[1] + 4* (xlims[2]-xlims[1])/6

new_y_low <- ylims[1] +  (ylims[2]-ylims[1])/12
new_y_high <- ylims[1] + 3*(ylims[2]-ylims[1])/12

new_xlims <- c(new_x_low, new_x_high)
new_ylims <- c(new_y_low, new_y_high)
```

```
 plt +
  coord_equal(xlim = new_xlims, ylim = new_ylims)
#> Coordinate system already present. Adding new coordinate system, which will replace the existing one
```



So, several of these pick up the subsidized housing and homeless community in Portland's Back Cove neighborhood. Other than that, what is striking is the fact that the ranks of the different metrics are not all that correlated in this smaller sub-region, which may help explain why few locations are flagged by any of the candidate indicators.