Spatial Data Analysis for Multnomah County, OR

Installing and Loading Packages

```
# Ensure to install these packages if you haven't already
# install.packages("sf")
# install.packages("tigris")
# install.packages("tmap")
# install.packages("RColorBrewer")

# Load necessary libraries
library(tidyverse)
library(tidycensus)
library(flextable)
library(sf)
library(tigris)
library(tmap)
library(RColorBrewer)
```

Setting up Census API Key

```
# Load in your Census API key (replace YOUR_API_KEY with your actual key)
census_api_key("YOUR_API_KEY", install = TRUE)
```

Retrieving Spatial Data for Multnomah County, OR

Then use the `get_acs()` command to bring in Oregon tract-level median household income, total foreign-born population, and total population from the 5-year 2018-2022 American Community Survey (ACS). Remember that "E" at the end of the variable indicates "Estimate" and "M" indicates margin of errors:

```
# Use tidycensus to get ACS data for Multnomah County, OR
or_tracts <- get_acs(geography = "tract",
                     year = 2022,
                     variables = c(medincome = "B19013_001",
                                   fb = "B05012_003",
                                   totp = "B05012_001"),
                     state = "OR",
                     county = "Multnomah",
                     survey = "acs5",
                     output = "wide",
                     geometry = TRUE)
```

```
## Getting data from the 2018-2022 5-year ACS
```

```
## Downloading feature geometry from the Census website.  To cache shapefiles for use in future sessions

##   |                                                                              |
```

```r
# View the first few rows of the data
head(or_tracts)
```

```
## Simple feature collection with 6 features and 8 fields
## Geometry type: MULTIPOLYGON
## Dimension:     XY
## Bounding box:  xmin: -122.7437 ymin: 45.46746 xmax: -122.4966 ymax: 45.55462
## Geodetic CRS:  NAD83
##         GEOID                                          NAME medincomeE
## 1 41051003402 Census Tract 34.02; Multnomah County; Oregon     100000
## 2 41051004700    Census Tract 47; Multnomah County; Oregon      69524
## 3 41051006601 Census Tract 66.01; Multnomah County; Oregon     128173
## 4 41051008302 Census Tract 83.02; Multnomah County; Oregon      70572
## 5 41051008400    Census Tract 84; Multnomah County; Oregon      47172
## 6 41051009101 Census Tract 91.01; Multnomah County; Oregon      59167
##   medincomeM  fbE fbM totpE totpM                       geometry
## 1      18318  569 410  4549   671 MULTIPOLYGON (((-122.6755 4...
## 2      21397  612 523  4108   732 MULTIPOLYGON (((-122.7127 4...
## 3      40990  160  84  2754   386 MULTIPOLYGON (((-122.7437 4...
## 4      22759 1121 309  5231   773 MULTIPOLYGON (((-122.5658 4...
## 5       8762 1237 328  4626   650 MULTIPOLYGON (((-122.548 45...
## 6      13991 1818 548  6261   696 MULTIPOLYGON (((-122.5237 4...
```

Exploring and Manipulating Spatial Data

```r
# Check the class of the retrieved object
class(or_tracts)
```

```
## [1] "sf"         "data.frame"
```

```r
# Example of data manipulation: Drop margin of error columns, rename variables, and calculate percent f
or_tracts <- or_tracts %>%
  select(-ends_with("M")) %>%
  rename(medincome = medincomeE, fb = fbE, totp = totpE) %>%
  mutate(pfb = 100 * (fb / totp))

# View the updated data
head(or_tracts)
```

```
## Simple feature collection with 6 features and 6 fields
## Geometry type: MULTIPOLYGON
## Dimension:     XY
## Bounding box:  xmin: -122.7437 ymin: 45.46746 xmax: -122.4966 ymax: 45.55462
## Geodetic CRS:  NAD83
##         GEOID                                          NAME medincome   fb totp
## 1 41051003402 Census Tract 34.02; Multnomah County; Oregon    100000  569 4549
```

```
## 2 41051004700    Census Tract 47; Multnomah County; Oregon     69524  612 4108
## 3 41051006601 Census Tract 66.01; Multnomah County; Oregon    128173  160 2754
## 4 41051008302 Census Tract 83.02; Multnomah County; Oregon     70572 1121 5231
## 5 41051008400    Census Tract 84; Multnomah County; Oregon     47172 1237 4626
## 6 41051009101 Census Tract 91.01; Multnomah County; Oregon     59167 1818 6261
##                          geometry       pfb
## 1 MULTIPOLYGON (((-122.6755 4... 12.508244
## 2 MULTIPOLYGON (((-122.7127 4... 14.897760
## 3 MULTIPOLYGON (((-122.7437 4...  5.809731
## 4 MULTIPOLYGON (((-122.5658 4... 21.429937
## 5 MULTIPOLYGON (((-122.548 45... 26.740164
## 6 MULTIPOLYGON (((-122.5237 4... 29.036895
```

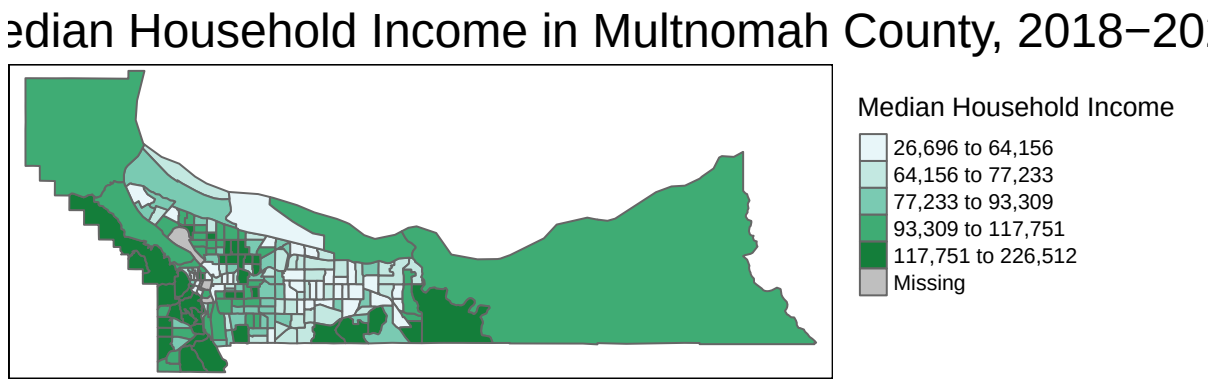
Mapping in R

Now that you've got your spatial data in and wrangled, the next natural step is to map something. There are several functions in R that can be used for mapping. The package we'll use today is tmap.

The foundation function for mapping in tmap is tm_shape(). You then build on tm_shape() by adding one or more elements, all taking on the form of tm_. Let's make a choropleth map of median housing values.

Mapping Median Household Income in Multnomah County

```
# Create a choropleth map of median household income
tm_shape(or_tracts) +
  tm_polygons(col = "medincome", style = "quantile", palette = "BuGn", title = "Median Household Income
  tm_layout(main.title = "Median Household Income in Multnomah County, 2018-2022",
            main.title.position = "center",
            legend.outside = TRUE)
```



You first put the dataset `or_tracts` inside `tm_shape()`. Because you are plotting polygons, you use `tm_polygons()` next. If you are plotting points, you will use `tm_dots()`. If you are plotting lines, you will use `tm_lines()`. The argument `col = "medincome"` tells R to shade the tracts by the variable `medincome`. `tmap` allows users to specify the classification style with the style argument. The argument `style = "quantile"` tells R to break up the shading into quantiles, or equal groups of 5. Seven of the most useful classification styles are described in the bullet points below:

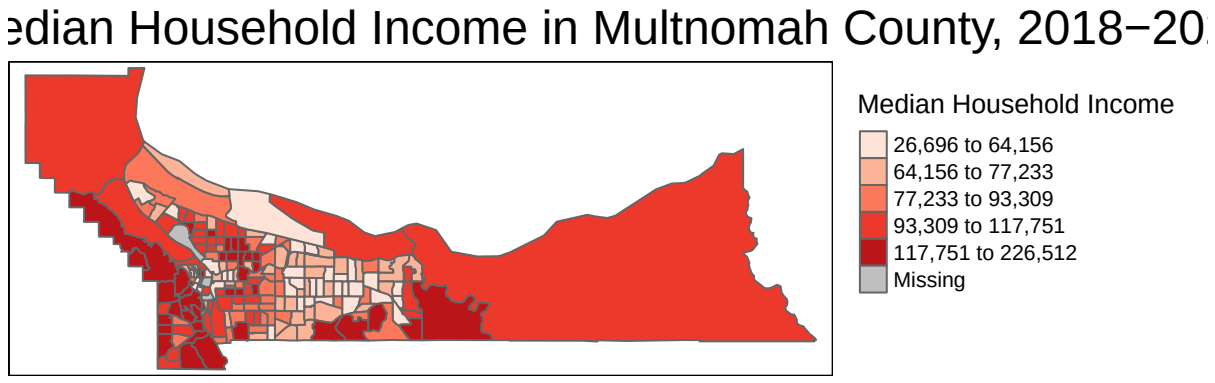
- `style = pretty`, the default setting, rounds breaks into whole numbers where possible and spaces them evenly
- `style = equal` divides input values into bins of equal range, and is appropriate for variables with a uniform distribution (not recommended for variables with a skewed distribution as the resulting map may end-up having little color diversity)
- `style = quantile` ensures the same number of observations fall into each category (with the potential down side that bin ranges can vary widely)
- `style = jenks` identifies groups of similar values in the data and maximizes the differences between categories
- `style = cont` (and order) present a large number of colors over continuous color field, and are particularly suited for continuous rasters (order can help visualize skewed distributions)
- `style = sd` divides the values by standard deviations above and below the mean.
- `style = cat` was designed to represent categorical values and assures that each category receives a unique color

Note that `tmap` is smart enough to detect the presence of missing values, and shades them gray and labels them on the legend.

Color Scheme

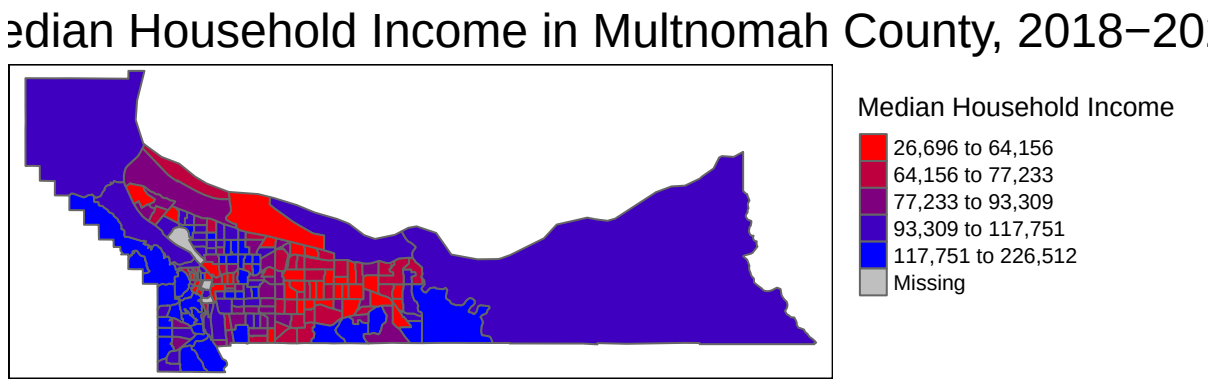
Don't like the color scheme? We can change the color scheme using the argument `palette =` within `tm_polygons()`. The argument `palette =` defines the color ranges associated with the bins as determined by the style argument. Below we use the color scheme "Reds" using `palette = "Reds"`.

```
tm_shape(or_tracts) +
  tm_polygons(col = "medincome", style = "quantile", palette = "Reds", title = "Median Household Income
  tm_layout(main.title = "Median Household Income in Multnomah County, 2018-2022",
            main.title.position = "center",
            legend.outside = TRUE)
```



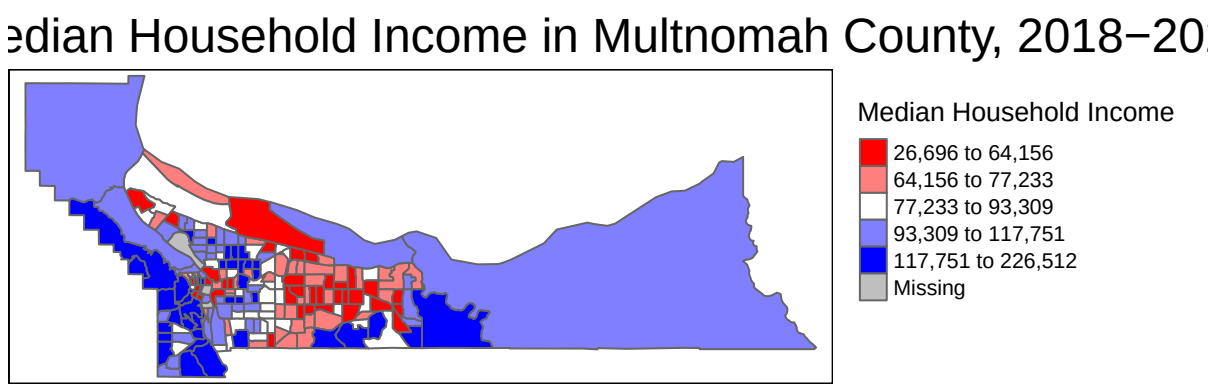
In addition to the built-in palettes, customized color ranges can be created by specifying a vector with the desired colors as anchors. This will create a spectrum of colors in the map that range between the colors specified in the vector. For instance, if we used `c("red","blue")`, the color spectrum would move from red to purple, then to blue, with in between shades. In our example:

```
tm_shape(or_tracts) +
  tm_polygons(col = "medincome", style = "quantile", palette = c("red","blue"), title = "Median Househol
  tm_layout(main.title = "Median Household Income in Multnomah County, 2018-2022",
            main.title.position = "center",
            legend.outside = TRUE)
```



Not exactly a pretty picture. In order to capture a diverging scale, we insert "white" in between red and blue.

```
tm_shape(or_tracts) +
  tm_polygons(col = "medincome", style = "quantile", palette = c("red","white","blue"), title = "Median
  tm_layout(main.title = "Median Household Income in Multnomah County, 2018-2022",
            main.title.position = "center",
            legend.outside = TRUE)
```



A preferred approach to select a color palette is to chose one of the schemes contained in the `RColorBrewer` package. These are based on the research of cartographer Cynthia Brewer (see the `colorbrewer2` web site for details). `RColorBrewer` makes a distinction between sequential scales (for a scale that goes from low to high), diverging scales (to highlight how values differ from a central tendency), and qualitative scales (for categorical variables). For each scale, a series of single hue and multi-hue scales are suggested. In the `RColorBrewer` package, these are referred to by a name (e.g., the "Reds" palette we used above is an example). The full list is contained in the `RColorBrewer` documentation.

There are two very useful commands in this package. One sets a color palette by specifying its name and the number of desired categories. The result is a character vector with the hex codes of the corresponding colors.

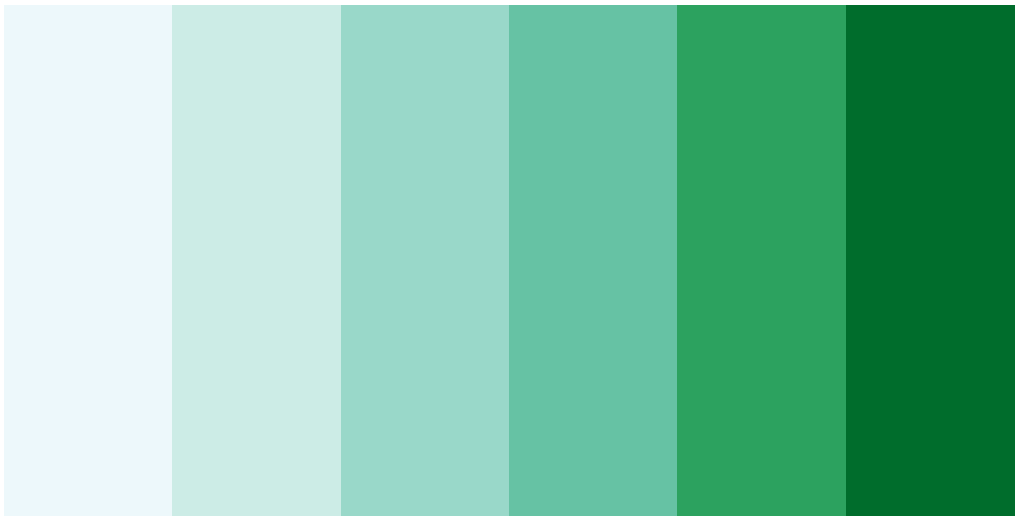
For example, we select a sequential color scheme going from blue to green, as BuGn, by means of the command `brewer.pal`, with the number of categories (6) and the scheme as arguments. The resulting vector contains the HEX codes for the colors.

```
pal <- brewer.pal(6,"BuGn")
pal
```

```
## [1] "#EDF8FB" "#CCECE6" "#99D8C9" "#66C2A4" "#2CA25F" "#006D2C"
```

The command `display.brewer.pal()` allows us to explore different color schemes before applying them to a map. For example:

```
display.brewer.pal(6,"BuGn")
```



BuGn (sequential)

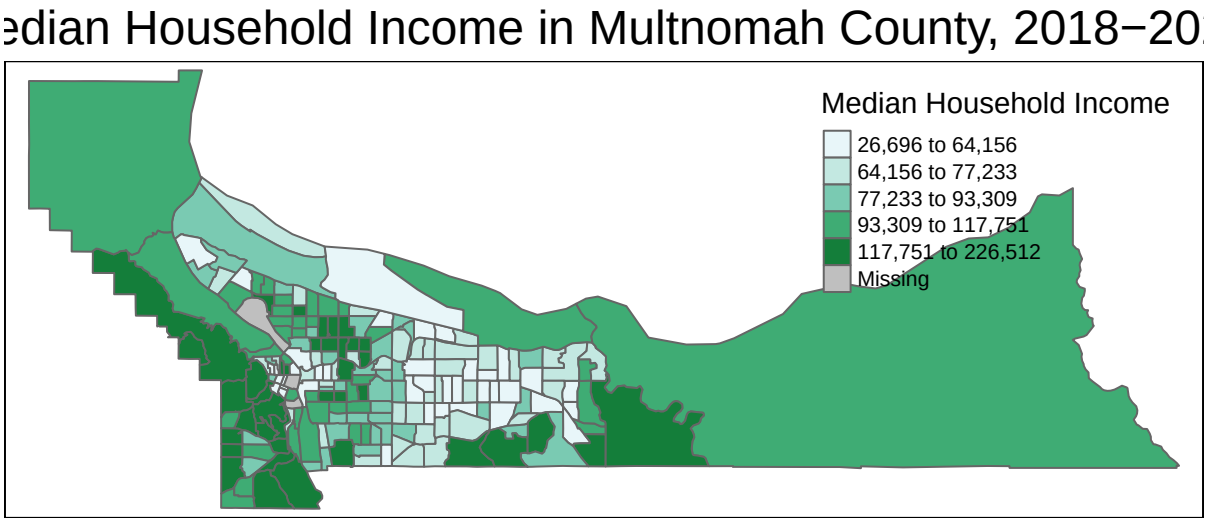
Legend

There are many options to change the formatting of the legend. Often, the automatic title for the legend is not intuitive, since it is simply the variable name (in our case, medincome). This can be customized by setting the title argument in `tm_polygons()`.

Another important aspect of the legend is its positioning. This is handled through the `tm_layout()` function. This function has a vast number of options, as detailed in the documentation.

The default is to position the legend inside the map. Often, this default solution is appropriate, but sometimes further control is needed. The legend.position argument in the `tm_layout()` function moves the legend around the map, and it takes on a vector of two string variables that determine both the horizontal position ("left", "right", or "center") and the vertical position ("top", "bottom", or "center"). The default is "right" and "bottom". But, we can change it to, say, top right.

```
# Create a choropleth map of median household income
tm_shape(or_tracts) +
  tm_polygons(col = "medincome", style = "quantile", palette = "BuGn", title = "Median Household Income
  tm_layout(main.title = "Median Household Income in Multnomah County, 2018-2022",
            main.title.position = "center",
            legend.position = c("right", "top"))
```

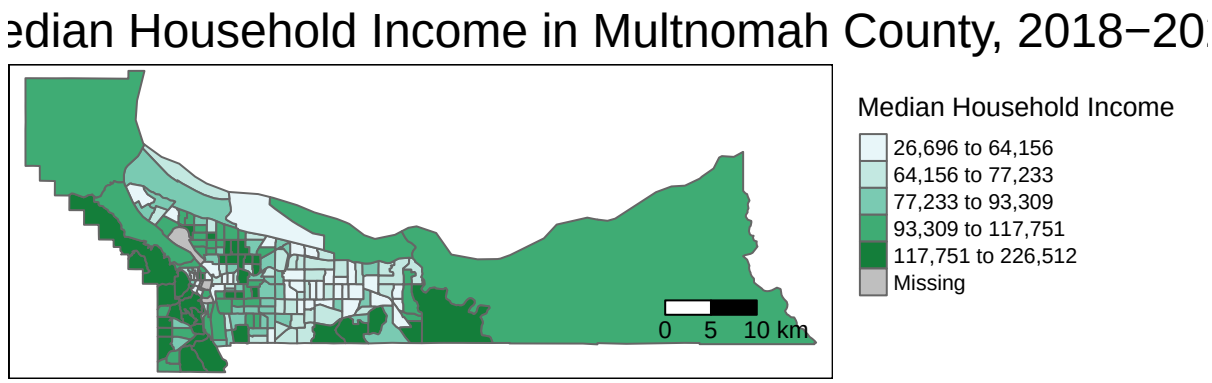


See that compared to the previous plots, the legend is now overlapping with the map! That is why I was using the option to position the legend outside the frame of the map. This is accomplished by setting `legend.outside = TRUE`

Scale bar and arrow

Here is where we start adding more layout functions after `tm_polygons()` using the `+` operator. First, the scale bar, which you can add using the `function tm_scale_bar()`

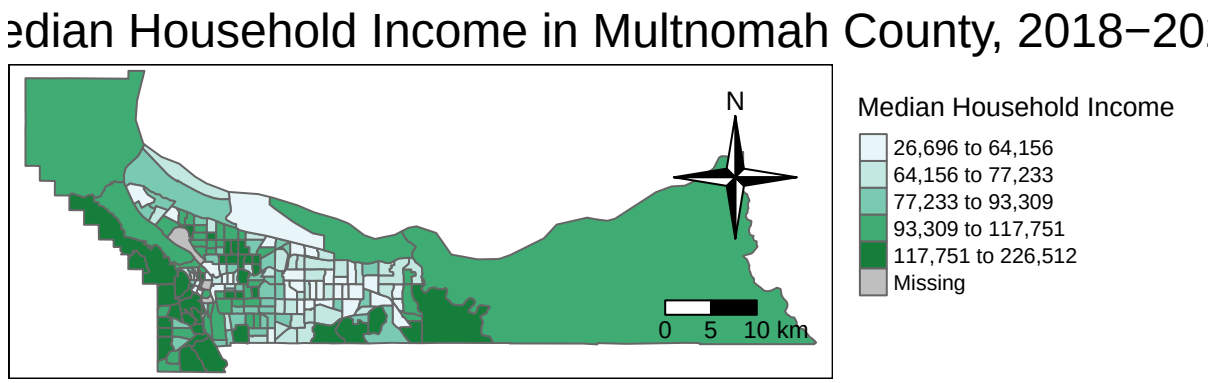
```
# Create a choropleth map of median household income
tm_shape(or_tracts) +
  tm_polygons(col = "medincome", style = "quantile", palette = "BuGn", title = "Median Household Income
  tm_layout(main.title = "Median Household Income in Multnomah County, 2018-2022",
            main.title.position = "center",
            legend.outside = TRUE) +
  tm_scale_bar(breaks = c(0, 5, 10), text.size  = 0.75,
               position = c("right", "bottom"))
```



The argument breaks within `tm_scale_bar()` tells R the distances to break up and end the bar. Make sure you use reasonable break points - the Multnomah county area is not 200 miles wide, so you should not use c(0,100,200) (try it and see what happens. You won't like it). Note that the scale is not in miles, the primary distance unit used here in the US; the default is in kilometers (the rest of the world!). You can specify the units within `tm_shape()` using the argument unit. Try adding `unit = "mi"` to designate distance in the scale bar measured in miles.

The next element is the north arrow, which we can add using the function `tm_compass()`. You can control for the type, size and location of the arrow within this function. We place a 4-star arrow on the top right of the map.

```
# Create a choropleth map of median household income
tm_shape(or_tracts) +
  tm_polygons(col = "medincome", style = "quantile", palette = "BuGn", title = "Median Household Income
  tm_layout(main.title = "Median Household Income in Multnomah County, 2018-2022",
            main.title.position = "center",
            legend.outside = TRUE) +
  tm_scale_bar(breaks = c(0, 5, 10), text.size  = 0.75,
               position = c("right", "bottom"))  +
  tm_compass(type = "4star", position = c("right", "top"))
```

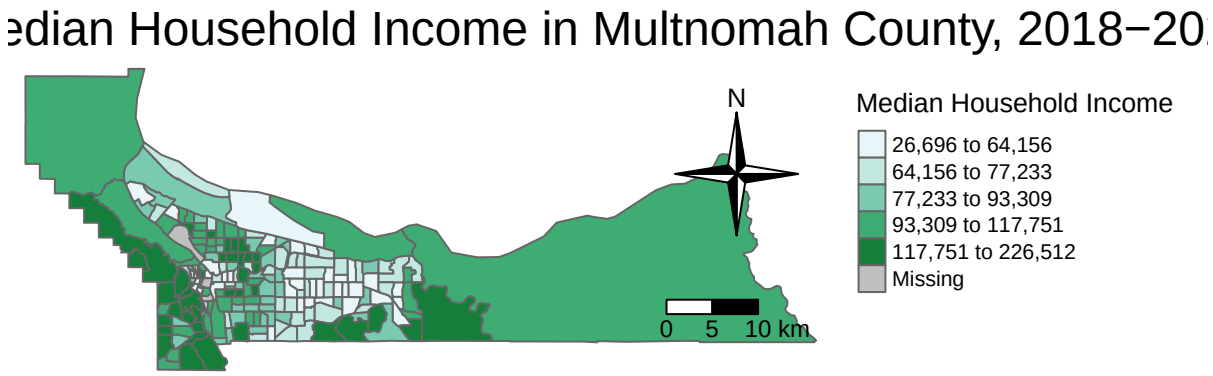


Other features

We can make the map prettier by changing a variety of settings. We can eliminate the frame around the map using the argument `frame = FALSE` with `tm_layout`:

```
# Create a choropleth map of median household income
mult_county_map <- tm_shape(or_tracts) +
  tm_polygons(col = "medincome", style = "quantile", palette = "BuGn", title = "Median Household Income
  tm_layout(main.title = "Median Household Income in Multnomah County, 2018-2022",
            main.title.position = "center",
            legend.outside = TRUE, frame = FALSE) +
  tm_scale_bar(breaks = c(0, 5, 10), text.size  = 0.75,
               position = c("right", "bottom"))  +
  tm_compass(type = "4star", position = c("right", "top"))

mult_county_map
```



Notice that we stored the map into an object called `mult_county_map`. R is an object-oriented language, so everything you make in R are objects that can be stored for future manipulation. This includes maps. You should see `mult_county_map` in your `Environment` window. By storing the map, you can access it anytime during your current R session.

Multiple map objects can be arranged in a single `metaplot` with `tmap_arrange()`. For example, let's map median household income and percent of the population foreign born in Multnomah county. Let's create the percent foreign born map, and save it into an object named `mult_pfb_map`:

```
mult_pfb_map <- tm_shape(or_tracts, unit = "mi") +
    tm_polygons(col = "pfb", style = "quantile",palette = "Reds",
                title = "% foreign-born") +
    tm_layout(main.title = "2018-22 Percent Foreign Born in Multnomah County",
              main.title.size = 0.95, main.title.position="center",
              legend.outside = TRUE, frame = FALSE, ) +
    tm_scale_bar(breaks = c(0, 5, 10), text.size  = 0.75,
                 position = c("right", "bottom"))  +
    tm_compass(type = "4star", position = c("left", "top")) +
    tm_polygons(col = "green")
```

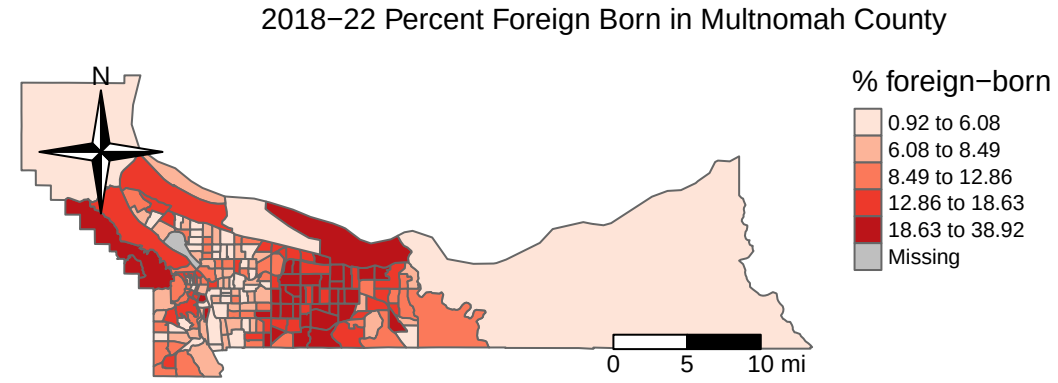
Then use `tmap_arrange()` to map `mult_pfb_map` and `mult_county_map` side by side:

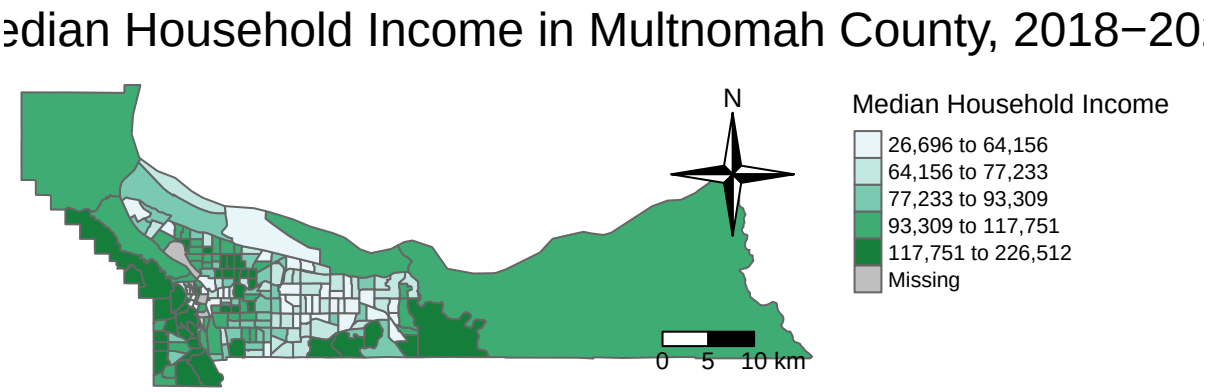
```
tmap_arrange(mult_pfb_map, mult_county_map)
```

```
## Warning: One tm layer group has duplicated layer types, which are omitted. To
## draw multiple layers of the same type, use multiple layer groups (i.e. specify
```

```
## tm_shape prior to each of them).

## Warning: One tm layer group has duplicated layer types, which are omitted. To
## draw multiple layers of the same type, use multiple layer groups (i.e. specify
## tm_shape prior to each of them).
```





Saving maps

You can save your maps using the function `tmap_save()`

```
tmap_save(mult_county_map, "map_multnomah_medianIncome.png")
```

```
## Map saved to /Users/aallorant/Desktop/math241s24/activity/map_multnomah_medianIncome.png

## Resolution: 3401.57 by 1296.46 pixels

## Size: 11.33857 by 4.321534 inches (300 dpi)
```

Specify the tmap object and a filename with an extension. It supports pdf, eps, svg, wmf, png, jpg, bmp and tiff. The default is png. Also make sure you've set your directory to the folder that you want your map to be saved in.

Interactive maps

So far we've created static maps. That is, maps that don't "move". But, we're all likely used to Google maps - which we can move around and zoom into. You can make interactive maps in R using the package `tmap`:

To make your tmap object interactive, use the function `tmap_mode()`. Type in "view" inside this function.

Now that the interactive mode has been 'turned on', all maps produced with `tm_shape()` will launch. Let's view our saved `mult_pfb_map` interactively:

Click on the further-left icon at the top-right of your chunk and a larger window should open up.

Besides interactivity, another important benefit of `tmap_mode()` is that it provides a basemap. The function of a basemap is to provide background detail necessary to orient the location of the map. In the static maps we produced earlier, Multnomah county was sort of floating in white space. As you can see in the interactive map above we've added geographic context to the surrounding area.

The default basemap in `tmap_mode()` is `CartoDB.Positron`. You can change the basemap through the `tm_basemap()` function. For example, let's change the basemap to an `OpenStreetMap`:

You can save your interactive map using the same methods described for static maps. To switch back to plotting mode (noninteractive), type in:

tigris package

Another package that allows us to bring in census geographic boundaries is `tigris`. Here is a list of all the geographies you can download through this package. Let's bring in the boundaries for Portland. Cities are designated as places by the Census. Use the `places()` function to get all places in Oregon.

```
pl <- places(state = "OR", cb = TRUE, year=2022)
```

```
##   |                                                                              |
```

The `cb = TRUE` argument tells R to download a generalized cartographic boundary file, which drastically reduces the size of the data (compare the file size when you don't include `cb = TRUE`). For example, it eliminates all areas that are strictly covered by water (e.g. lakes). The argument `year=2022` tells R to bring in the boundaries for that year (census geographies can change from year to year). When using the multi-year ACS, best to use the end year of the period. In the `get_acs()` command above we used `year=2022`, so also use `year=2022` in the places() command. Note that unlike the `tidycensus` package, `tigris` does not allow you to attach attribute data (e.g. Hispanic, total population, etc.) to geometric features.

Take a glimpse of pl

```
glimpse(pl)
```

```
## Rows: 426
## Columns: 13
## $ STATEFP    <chr> "41", "41", "41", "41", "41", "41", "41", "41", "41", "41",~
## $ PLACEFP    <chr> "49600", "81300", "50050", "76250", "33250", "49150", "6170~
## $ PLACENS    <chr> "02411135", "02412250", "02411174", "02412133", "02410735",~
## $ AFFGEOID   <chr> "1600000US4149600", "1600000US4181300", "1600000US4150050",~
## $ GEOID      <chr> "4149600", "4181300", "4150050", "4176250", "4133250", "414~
## $ NAME       <chr> "Monroe", "Wheeler", "Mosier", "Unity", "Helix", "Mitchell"~
## $ NAMELSAD   <chr> "Monroe city", "Wheeler city", "Mosier city", "Unity city",~
## $ STUSPS     <chr> "OR", "OR", "OR", "OR", "OR", "OR", "OR", "OR", "OR", "OR",~
```

```
## $ STATE_NAME <chr> "Oregon", "Oregon", "Oregon", "Oregon", "Oregon", "Oregon",~
## $ LSAD       <chr> "25", "25", "25", "25", "25", "25", "25", "25", "25", "25",~
## $ ALAND      <dbl> 1332983, 1351171, 1181456, 1658349, 328347, 3309008, 273227~
## $ AWATER     <dbl> 0, 0, 455449, 20553, 0, 318, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ~
## $ geometry   <MULTIPOLYGON [°]> MULTIPOLYGON (((-123.3067 4..., MULTIPOLYGON (~
```

We see a `geometry` column, which indicates we have a spatial data frame.

We can use `filter()` to keep Portland. We will filter on the variable NAME to keep Portland.

```
pdx <- filter(pl, NAME == "Portland")
```

The argument `NAME == "Portland"` tells R to keep cities with the exact city name "Portland".

```
glimpse(pdx)
```

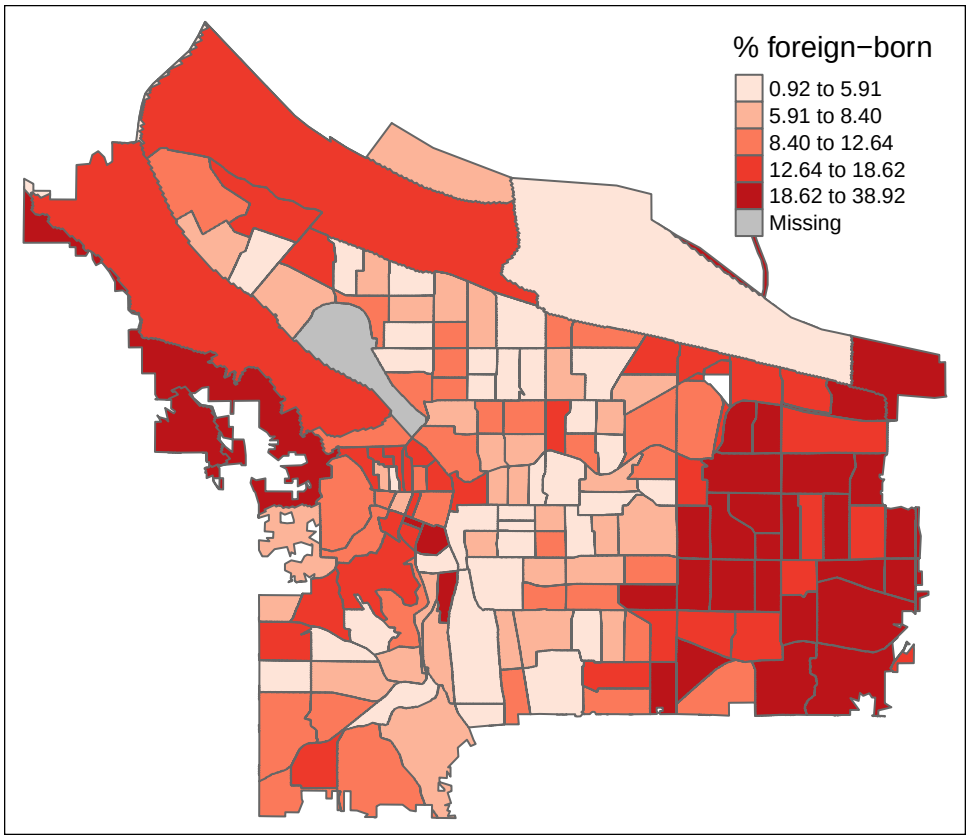
```
## Rows: 1
## Columns: 13
## $ STATEFP    <chr> "41"
## $ PLACEFP    <chr> "59000"
## $ PLACENS    <chr> "02411471"
## $ AFFGEOID   <chr> "1600000US4159000"
## $ GEOID      <chr> "4159000"
## $ NAME       <chr> "Portland"
## $ NAMELSAD   <chr> "Portland city"
## $ STUSPS     <chr> "OR"
## $ STATE_NAME <chr> "Oregon"
## $ LSAD       <chr> "25"
## $ ALAND      <dbl> 345732842
## $ AWATER     <dbl> 29817691
## $ geometry   <MULTIPOLYGON [°]> MULTIPOLYGON (((-122.8365 4...
```

Let's use use the `pdx` object to **crop** our `or_tracts` and focus only on those census tracts that are located in Portland:

```
# cropped version of main spatial object focusing on Portland:
pdx_tracts <- or_tracts %>%
    st_buffer(1e-5) %>%
    st_intersection(pdx)
```

```
## Warning: attribute variables are assumed to be spatially constant throughout
## all geometries
```

```
tm_shape(pdx_tracts) +
    tm_polygons(col = "pfb", style = "quantile",palette = "Reds",
                title = "% foreign-born")
```



Your turn!

1. Using `tidycensus`, bring in 2018-2022 total, non-Hispanic white, non-Hispanic black, non-Hispanic Asian, and Hispanic populations for census tracts in your birth county or your county of choice! Check Activity for the appropriate variable IDs. Make sure to bring in spatial data by using geometry = TRUE. Also make sure to clean the data.

2. Calculate the percent Asian, White, Black and Hispanic variables.

3. Create static presentation-ready maps of percent non-Hispanic black, non-Hispanic white, non-Hispanic Asian, and Hispanic using quantile breaks.

4. Bring in another variable from the ACS 2018-2022 that you would be interested in investigating in relationship to race/ethnicity. This could be median housing value (`B25077_001`), median household income (`B19013_001`), proportion of foreign born individuals (`B05012_003`), or commuting using public transportation (`DP03_0021P`), to name a few variables that we have already encountered. You can also use the `load_variables` function to get the variable IDs of other variables you would be interested in.

5. Create a static presentation-ready map of the variable you selected.

6. Based on a visual comparison of the maps you created in (3) and (5), summarize in your own words the association between racial/ethnic composition and your variable of interest.