

Review: Load and clean data

With minimal commentary, below is code to load and clean two datasets for visual analysis in the rest of the post. The first dataset is a .geojson file containing geospatial descriptions of Philadelphia's neighborhoods, courtesy of [OpenDataPhilly](#). This dataset is polygon data and will form our basemap for layering on additional, more interesting, features.

The second dataset, also courtesy of [OpenDataPhilly](#) is our dataset of interest: a 2016 inventory of all of the trees in Philadelphia. This dataset is point data, where each tree has associated coordinates for identifying its precise location. Don't ask who collected this data or why; just be thankful that it exists. As for its accuracy, I can attest that when I step out on my stoop and look up and down the street, what I see is spot on with the city's data. At any rate, it is more than sufficient for illustrative purposes.

Without further ado, let's get the data loaded into our workspace.

```

# SETUP      #####
library(dplyr)
library(sf)
library(ggplot2)

# LOAD DATA #####
## ## Simple features data: Philadelphia neighborhoods
# Source: OpenDataPhilly. https://www.opendataphilly.org/dataset/philadelphia-neighborhoods
neighborhoods_geojson <- "https://raw.githubusercontent.com/azavea/geo-data/master/Neighborhoods\_Philadelphia/Neighborhoods\_Philadelphia.geojson"
neighborhoods_raw <- sf::read_sf(neighborhoods_geojson)

head(neighborhoods_raw)
#> Simple feature collection with 6 features and 8 fields
#> geometry type:  MULTIPOLYGON
#> dimension:      XY
#> bbox:           xmin: -75.28027 ymin: 39.96271 xmax: -75.01684 ymax: 40.09464
#> geographic CRS: WGS 84
#> # A tibble: 6 x 9
#>   name listname mapname shape_leng shape_area cartodb_id created_at
#>
#> 1 PENN~ Pennypa~ Pennyp~      87084.    60140756.           9 2013-03-19
13:41:50
#> 2 OVER~ Overbro~ Overbr~      57005.    76924995.          138 2013-03-19
13:41:50
#> 3 GERM~ Germant~ Southw~      14881.    14418666.           59 2013-03-19
13:41:50
#> 4 EAST~ East Pa~ East P~      10886.     4231000.          129 2013-03-19
13:41:50
#> 5 GERM~ Germany~ German~      13042.     6949968.           49 2013-03-19
13:41:50
#> 6 MOUN~ Mount A~ East M~      28846.    43152470.           6 2013-03-19
13:41:50
#> # ... with 2 more variables: updated_at , geometry

## ## Simple features data: Philadelphia urban forest
# Source: OpenDataPhilly. https://www.opendataphilly.org/dataset/philadelphia-street-tree-inventory
trees_geojson <- "http://data.phl.opendata.arcgis.com/datasets/957f032f9c874327a1ad800abd88717\_0.geojson"
trees_raw <- sf::read_sf(trees_geojson)

head(trees_raw)
#> Simple feature collection with 6 features and 4 fields
#> geometry type:  POINT
#> dimension:      XY
#> bbox:           xmin: -75.15902 ymin: 39.9944 xmax: -75.15441 ymax: 39.99517
#> geographic CRS: WGS 84
#> # A tibble: 6 x 5

```

```
#>   OBJECTID SPECIES STATUS DBH          geometry
#>
#> 1         1          (-75.15608 39.99506)
#> 2         2          (-75.1578 39.99459)
#> 3         3          (-75.15441 39.99517)
#> 4         4          (-75.15446 39.99495)
#> 5         5          (-75.15752 39.9944)
#> 6         6          (-75.15902 39.9946)
```

```
# CLEAN DATA #####
neighborhoods <- neighborhoods_raw %>%
  dplyr::transmute(
    NEIGHBORHOOD_ID = cartodb_id,
    LABEL = mapname,
    AREA = shape_area/43560 # sq. ft to acres
  )

trees <- trees_raw %>%
  dplyr::select(TREE_ID = OBJECTID)
```

As you can see, we now have two objects—`neighborhoods` and `trees`—prepared for our analysis. Since the original data was relatively clean, the only modifications we have made are to drop columns, rename columns, and convert square feet into acres.

Geospatial layers in ggplot2

Your first map

To draw static maps in R, we will use `ggplot2`, which is not only the standard package for drawing traditional bar plots, line plots, histograms, and other standard visualizations of discrete or continuous data, but is also the standard package for drawing maps. A few other packages are excellent alternatives including `sf` and `maps`. We prefer `ggplot2` because it has a consistent grammar of graphics between its various applications and offers a robust set of geospatial graphing functions.

Let's take a moment to refresh ourselves on `ggplot2`'s functionality. To make a plot, you need three steps: (1) initiate the plot, (2) add as many data layers as you want, and (3) adjust plot aesthetics, including scales, titles, and footnotes.

To (1) initiate the plot, we first call `ggplot()`, and to (2) add data layers, we next call `geom_sf()` once for each layer. We have the option to add `data = neighborhoods` to provide simple features data to our plot either in the `ggplot()` call or in the `geom_sf()` call. In `ggplot2`, functions inherit from functions called higher up. Thus, if we call `ggplot(data = neighborhoods)`, we do not need to subsequently specify `geom_sf(data = neighborhoods)`, as the `data` argument is inherited in `geom_sf()`. The same goes for aesthetics. If we specify `ggplot(data = neighborhoods, aes(fill = FILL_GROUP))`, the subsequent `geom_sf()` call will already know how we want to color our plot. The behavior of inheriting aesthetics is convenient for drawing graphs that only use one dataset, which is most common in non-geospatial visualization. A line plot might call both `geom_line()` and `geom_point()` and want both the lines and points to inherit the same dataset. Since maps frequently have many layers from different sources, we will elect to specify our data and aesthetics within the `geom_sf()` calls and not in the `ggplot()` calls.

Finally, to (3) adjust overall aesthetics, we will use a range of functions, such as the `theme_*` family of

functions, the `scale_fill_*` family of functions, and the `coord_sf()` function. Before we get there, however, we will start with a simple plot with just our two data layers.

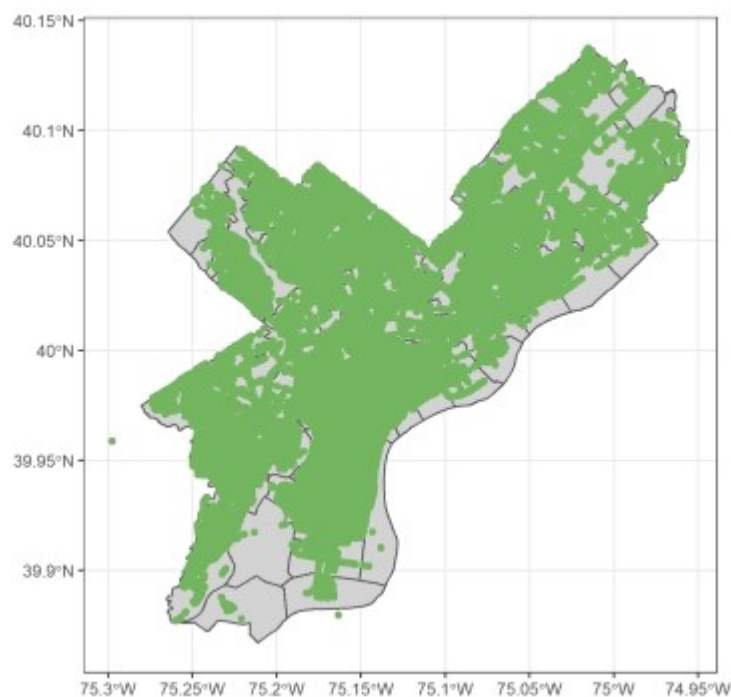
```
ggplot2::ggplot() +  
  ggplot2::geom_sf(data = neighborhoods) +  
  ggplot2::geom_sf(data = trees)
```



Simple formatting adjustments

Let's clean up the colors to make this a little bit easier on the eyes.

```
ggplot2::ggplot() +  
  ggplot2::geom_sf(data = neighborhoods, fill = "#d3d3d3") +  
  ggplot2::geom_sf(data = trees, color = "#74b560") +  
  ggplot2::theme_bw()
```

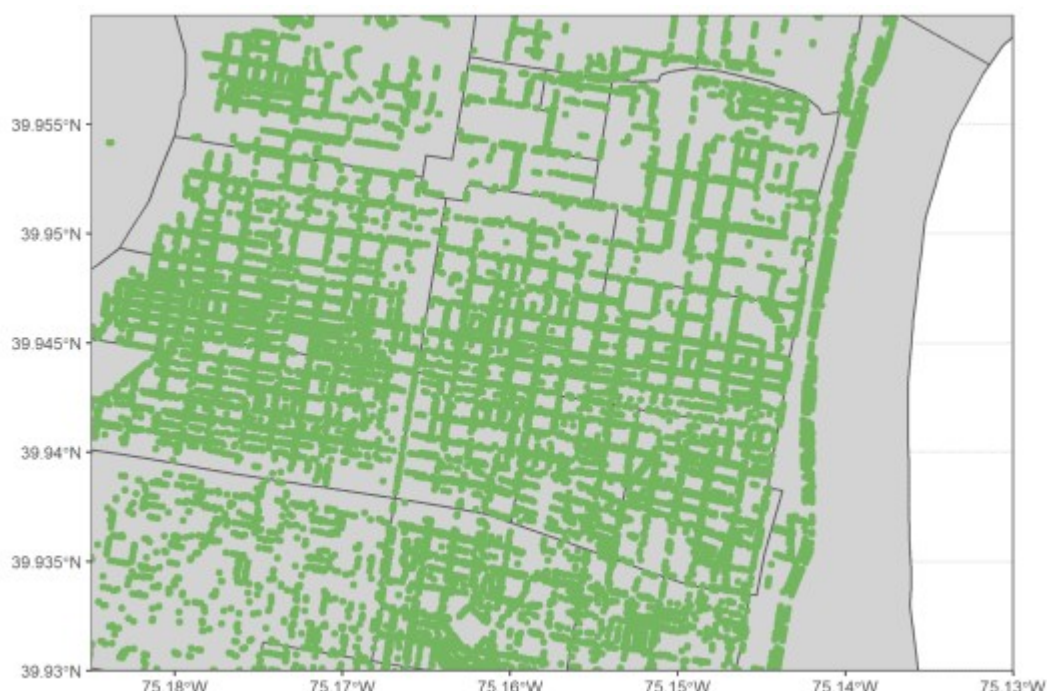


Zoom in on a region of interest

By default, `ggplot2` will zoom out so that all of the mapping objects are in the image. Suppose, however that we are interested in a smaller region of the map: Center City Philadelphia. We can use `ggplot2::coord_sf()` to specify the coordinates to display. By default, `geom_sf()` calls `coord_sf()` in the background, but by explicitly calling it ourselves, we can override the default parameters. Below, we will specify our latitude and longitude, and set `expand = FALSE`. By default, `expand` is true, which puts a small buffer around the coordinates we specify. It's an aesthetic choice.

While we're here, let's make a brief comment on CRSs in `ggplot2` mapping. If you recall from [Part 1](#) of this series, the CRS is the ellipsoid and datum used to reference points on the globe. `ggplot2` will take the first CRS provided (in this case, in our `neighborhoods` dataset) and ensure that all subsequent layers use the same CRS. It automatically converts any mismatched CRSs to the first one provided. Using `coord_sf()`, we have options to change the CRS and the datum. Changing the datum won't affect plotting, but will affect where graticules (latitude/longitude lines) are drawn if you choose to include them. By default, `ggplot2` draws graticules using WGS 84 (EPSG: 4326), which happens to be the CRS of our two datasets. If we had needed to, we could have changed to NAD 83 (EPSG: 4269) using `datum = sf::st_crs(4269)`.

```
ggplot2::ggplot() +  
  ggplot2::geom_sf(data = neighborhoods, fill = "#d3d3d3") +  
  ggplot2::geom_sf(data = trees, color = "#74b560") +  
  ggplot2::coord_sf(  
    xlim = c(-75.185, -75.13), ylim = c(39.93, 39.96),  
    expand = FALSE  
  ) +  
  ggplot2::theme_bw()
```

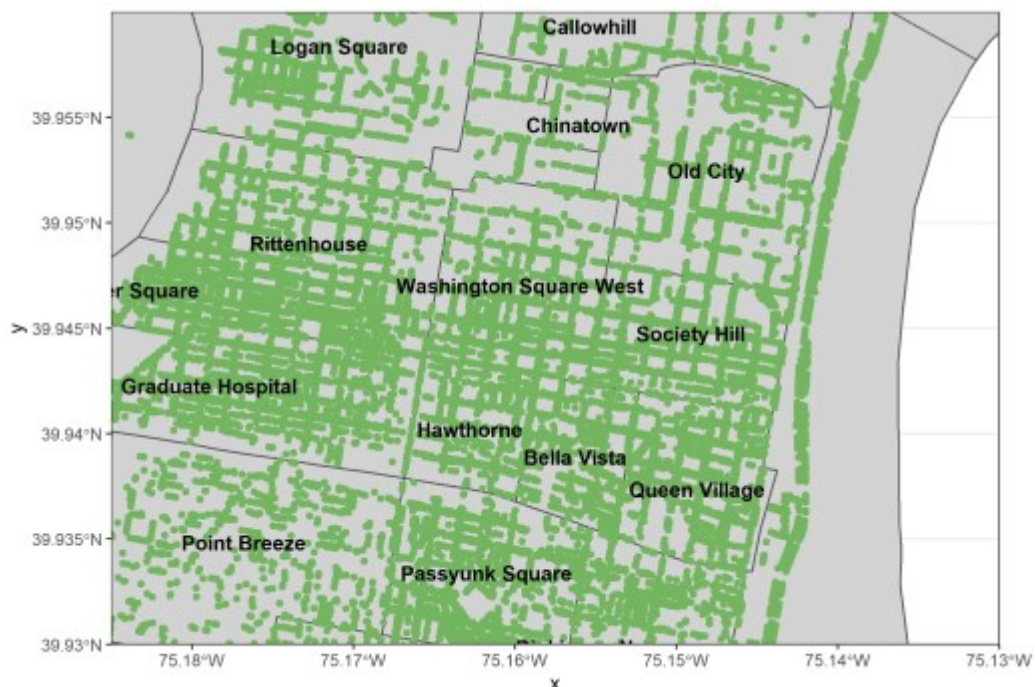


Add labels for clearer communication

Now that we have zoomed in to a smaller region, we have space on our map to add labels. Here, we use the function, `geom_sf_text()`, specifying the label as an aesthetic (`aes(...)`). We use the aesthetic because the label is coming from within our data object and changes with each row in the object, as

opposed to being a single value specified at time of plotting, as we had done with fill and color above. Notice that by adding the labels after the “trees” data, the label appears on top. `geom_sf_text()` adds text, and `geom_sf_label()` adds text enclosed in a small box for easier reading. It’s an aesthetic choice.

```
ggplot2::ggplot() +
  ggplot2::geom_sf(data = neighborhoods, fill = "#d3d3d3") +
  ggplot2::geom_sf(data = trees, color = "#74b560") +
  ggplot2::geom_sf_text(
    data = neighborhoods, aes(label = LABEL),
    fontface = "bold", check_overlap = TRUE
  ) +
  ggplot2::coord_sf(
    xlim = c(-75.185, -75.13), ylim = c(39.93, 39.96),
    expand = FALSE
  ) +
  ggplot2::theme_bw()
```



Add highlights and annotations

I want to highlight and annotate my favorite tree. Since the highlight rule needs to be determined tree-by-tree, we need to adjust our simple features object and add an appropriate aesthetic call to our plot. First we adjust our simple features object by adding columns for the color group and the label text. Then, we adjust our plot by including `aes(color = COLOR)` to define color groups and simultaneously adding `scale_color_manual()` to specify the colors we want for each group. At the same time, we optionally set `show.legend = FALSE` to hide the legend. We also add the label text using `geom_sf_label()` using the `aes(label = LABEL)` to specify the text and other parameters to adjust how it appears on the plot.

```
trees_highlight <- trees %>%
  dplyr::mutate(
    HIGHLIGHT_IND = dplyr::if_else(TREE_ID == 39547, 1, 0),
    COLOR = as.factor(HIGHLIGHT_IND),
```



```

    LABEL = dplyr::if_else(HIGHLIGHT_IND == 1, "My favorite", NULL)
  ) %>%
  dplyr::select(-HIGHLIGHT_IND)

ggplot2::ggplot() +
  ggplot2::geom_sf(data = neighborhoods, fill = "#d3d3d3") +
  ggplot2::geom_sf(
    data = trees_highlight, aes(color = COLOR),
    show.legend = FALSE
  ) +
  ggplot2::geom_sf_text(
    data = neighborhoods, aes(label = LABEL),
    fontface = "bold", check_overlap = TRUE
  ) +
  ggplot2::geom_sf_label(
    data = trees_highlight, aes(label = LABEL),
    color = "#cb7123", fontface = "bold",
    nudge_x = 0.005, na.rm = TRUE
  ) +
  ggplot2::coord_sf(
    xlim = c(-75.185, -75.13), ylim = c(39.93, 39.96),
    expand = FALSE
  ) +
  ggplot2::scale_color_manual(values = c("#74b560", "#cb7123")) +
  ggplot2::theme_bw()

```



Final beautification

The options for beautifying your map are endless. Some people like to add a scale and north arrow using the `ggspatial` package. I prefer to leave it off but to add axis labels, a title and subtitle, a source note, and to make a few additional adjustments using the `theme()` function. The aesthetic choice is yours.

```

ggplot2::ggplot() +

```

```

ggplot2::geom_sf(data = neighborhoods, fill = "#d3d3d3") +
ggplot2::geom_sf(
  data = trees_highlight, aes(color = COLOR),
  show.legend = FALSE
) +
ggplot2::geom_sf_text(
  data = neighborhoods, aes(label = LABEL),
  fontface = "bold", check_overlap = TRUE
) +
ggplot2::geom_sf_label(
  data = trees_highlight, aes(label = LABEL),
  color = "#cb7123", fontface = "bold",
  nudge_x = 0.006, na.rm = TRUE
) +
ggplot2::coord_sf(
  xlim = c(-75.185, -75.13), ylim = c(39.93, 39.96),
  expand = FALSE
) +
ggplot2::xlab("Longitude") +
ggplot2::ylab("Latitude") +
ggplot2::labs(
  title = "The Urban Forest of Center City Philadelphia",
  subtitle = "2016 virtual assessment of Philadelphia's street trees",
  caption = "Source: OpenDataPhilly"
) +
ggplot2::scale_color_manual(values = c("#74b560", "#cb7123")) +
ggplot2::theme_bw() +
ggplot2::theme(
  panel.grid.major = ggplot2::element_line(
    color = gray(0.5), linetype = "dashed", size = 0.5
  ),
  panel.background = ggplot2::element_rect(fill = gray(0.75))
)

```



Choropleths in ggplot2

We must cover one final topic in `ggplot2` before wrapping up this article. This is the concept of a “choropleth” map, which colors regions to represent a statistical variable. For instance, we may want to color our neighborhoods by the number of trees in each, or (more appropriately) the number of trees per acre.

Merge and clean data

To color our neighborhoods, we must first merge what had been two distinct layers of neighborhoods and trees into a single simple features object. By combining, we lose our point data and convert it into a statistic (the count of trees and trees per acre) associated with each neighborhood.

The `sf::st_join()` function has options for the join method. By default, it uses `st_intersects`, but we can change it to other options such as `st_contains` or `st_covered_by`, depending on our needs. When joining polygon data and point data, we may have some unusual cases for trees that are on the exact border between neighborhoods. For this example, we can safely ignore those edge cases.

The join will result in a very long object, with one row for every combination of neighborhood and tree. Thus, we need to `group_by` and `summarise` as quickly as we can to return to the proper-sized data frame.

```
tree_count <- sf::st_join(neighborhoods, trees) %>%
  dplyr::group_by(NEIGHBORHOOD_ID, LABEL, AREA) %>%
  dplyr::summarise(
    COUNT = n(),
    .groups = "drop"
  ) %>%
  dplyr::mutate(DENSITY = COUNT/AREA) %>%
  dplyr::select(NEIGHBORHOOD_ID, LABEL, DENSITY)

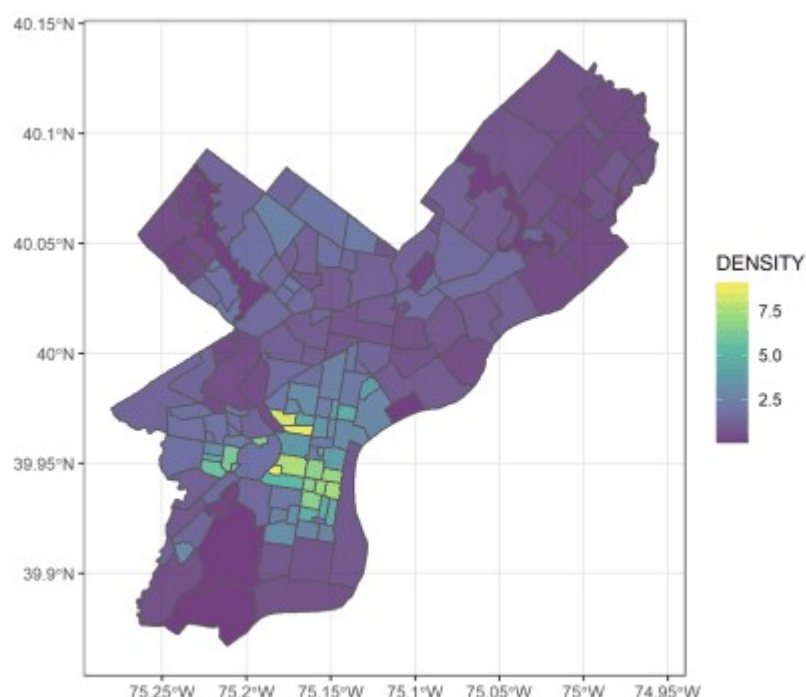
head(tree_count)
#> Simple feature collection with 6 features and 3 fields
#> geometry type:  MULTIPOLYGON
#> dimension:      XY
#> bbox:           xmin: -75.23049 ymin: 39.9849 xmax: -75.0156 ymax: 40.1126
#> geographic CRS: WGS 84
#> # A tibble: 6 x 4
#>   NEIGHBORHOOD_ID LABEL      DENSITY
#>   <geometry>
#>
#> 1              1 Bridesbu~  0.459 (((-75.06773 40.0054, -75.06765
40.0052, -7~
#> 2              2 Bustleton  0.714 (((-75.0156 40.09487, -75.01768
40.09276, --
#> 3              3 Cedarbro~  1.37  (((-75.18848 40.07273, -75.18846
40.07275, ~
#> 4              4 Chestnut~  1.34  (((-75.21221 40.08603, -75.21211
40.08603, ~
#> 5              5 East Fal~  1.55  (((-75.18479 40.02837, -75.18426
40.02789, ~
#> 6              6 East Mou~  2.50  (((-75.18088 40.04325, -75.18096
40.04317, ~
```

Draw the choropleth

Now we are ready to draw our map. After the exercises above, this map will seem surprisingly simple, since we have only one data object. Since we have only one object, we can safely place it in the initial `ggplot()` call and allow our subsequent functions to inherit the data and `aes`.

In order to color our object, we need to specify a scale. In this case, we will use `scale_fill_viridis_c` (where “c” stands for “continuous”), with an `alpha` (opacity) of 0.75. This choropleth, as opposed to the previous graphs of the tree locations, immediately shows the reader how variable tree density is across the city, with Center City, Fairmount, and University City much more tree-filled than all other regions.

```
ggplot2::ggplot(data = tree_count, aes(fill = DENSITY)) +  
  ggplot2::geom_sf() +  
  ggplot2::scale_fill_viridis_c(alpha = 0.75) +  
  ggplot2::theme_bw()
```



Final beautification

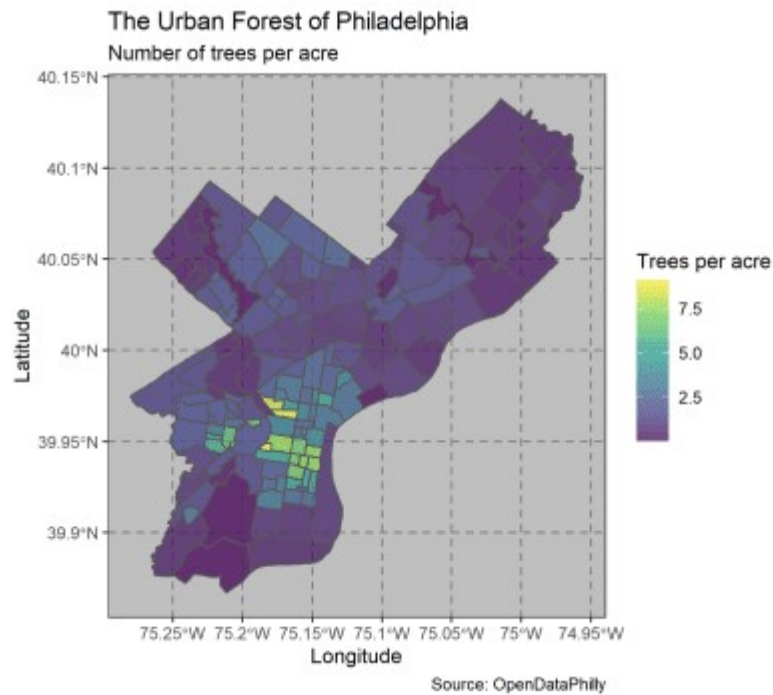
As before, the options for beautifying your map are endless. We now have the added option of formatting our legend title with `labs(fill = "Trees per acre")`. If you desire, you can apply the same skills of zooming, labeling, and annotating as we used above in our non-choropleth maps to highlight particular regions of interest.

```
ggplot2::ggplot(data = tree_count, aes(fill = DENSITY)) +  
  ggplot2::geom_sf() +  
  ggplot2::scale_fill_viridis_c(alpha = 0.75) +  
  ggplot2::xlab("Longitude") +  
  ggplot2::ylab("Latitude") +  
  ggplot2::labs(  
    title = "The Urban Forest of Philadelphia",  
    subtitle = "Number of trees per acre",  
    caption = "Source: OpenDataPhilly",  
    fill = "Trees per acre"
```

```

) +
ggplot2::theme_bw() +
ggplot2::theme(
  panel.grid.major = ggplot2::element_line(
    color = gray(0.5), linetype = "dashed", size = 0.5
  ),
  panel.background = ggplot2::element_rect(fill = gray(0.75))
)

```



Conclusion

To conclude, we have now seen how to add multiple layers of simple features data to a single `ggplot2` plot and adjust the aesthetics of those layers to suit our needs. This includes the ability to zoom, add labels, and highlight geospatial features. We have also seen how to join two geospatial datasets together in order to create a choropleth map, coloring regions according to a desired summary statistic.