**Basic Functions**

* Importing/uploading boundary files – with tigris, set year to 2018, cb = TRUE, etc.

devtools::install\_github("UrbanInstitute/urbnmapr") # don't update others

library(urbnmapr) # for states or counties only

library(tigris) # for states, counties, tracts, urban\_areas, etc.

library(sf)

library(tidyverse)

# Showing both urbnmapr and tigris options (for states)

object <- urbnmapr::get\_urbn\_map(“states”, sf = TRUE) %>%

shift\_geometry()

object <- tigris::states(cb = TRUE) %>%

filter(!STATE\_NAME %in% c(“Guam”, “American Samoa”,

“United States Virgin Islands”,

“Commonwealth of the Northern Mariana Islands”,

“Puerto Rico”)) %>%

shift\_geometry()

# You can filter during the download in tigris – particularly useful when getting small features like tracts. \*\*\*\*Remember to set year to 2018 (something other than the default 2020) when getting something other than states or counties.

object <- tigris::tracts(state = “Colorado”, cb = TRUE, year = 2018)

# If uploading your own shapefile, go to Files – Upload (zipfile of shapefile components); then read into your project.

object <- st\_read(“shapefilename.shp”)

* Joining data with dplyr’s left\_join

# Be sure to View() and glimpse() each of the two data sets you want to join, identifying the common variable (even if they have different names). Make sure they are both of the same class type. If not, reclassify as needed (most likely using Base R).

df1$var <- as.numeric(df1$var)

# Then you can join using dplyr’s left\_join(). \*\*\*Generally df1 is your boundary file. You can join time-series data to boundary files.

object <- left\_join(df1, df2, by = c(“df1var” = “df2var”)

* Filtering for particular cases: a) with dplyr’s filter if the filter attribute is in the data; b) with sf()’s st\_intersection if based on spatial location

# When you can filter by an existing variable/attribute in a larger data set:

smalldf <- df %>%

filter(var == condition) # ==, >=, <=, &, |, etc.

Or, instead of creating a separate object/df, you can pipe your filter into ggplot…

df %>%

filter(var == condition) %>%

ggplot(……)

* Plotting using geom\_sf()

ggplot(data = df, aes(fill = var)) +

geom\_sf() +

theme\_void()

**Additional Mapping Options**

**statebins**

<https://cran.r-project.org/web/packages/statebins/statebins.pdf>

additional: <https://github.com/hrbrmstr/statebins>

# Two ways to create state bins – 1) through ggplot; 2) through statebins function.

library(statebins)

# ggplot route

ggplot(data = df, aes(state = statevar, fill = var)) +

geom\_statebins() +

coord\_equal() +

theme(legend.title = “title”) +

viridis::scale\_fill\_viridis() +

labs(title = “title”, caption = “caption”) +

theme\_void()

Or…

#statebins route

statebins(state\_data = df, state\_col = “stateabbr”,

value\_col = “var”, name = “name for legend”,

font\_size = 3) +

labs(title = “title”, caption = “caption”) +

theme\_statebins(legend\_position = “bottom”)

**Small Multiple Maps**

Create a small multiple maps plot of opiate deaths following Healy’s lead (but try to not create multiple plots along the way as he does. <https://socviz.co/maps.html#small-multiple-maps>

Be sure your data are joined to a boundary file (e.g., states in this case). Additionally, use View() and glimpse() to identify the common variable and ensure they are of the same class type.

df <- left\_join(states, otherdata, by = c(“statesvar” = “otherdatavar”)

ggplot(data = df, aes(fill = var)) +

geom\_sf(size = 0.05) +

scale\_fill\_viridis\_c(option = “plasma”) +

theme\_void() +

facet\_wrap(~year, ncol = 3)

**geofacet**

<https://cran.r-project.org/web/packages/geofacet/vignettes/geofacet.html>

library(geofacet)

ggplot(df, aes(x = year, y = var)) +

geom\_line() + # can also try geom\_area()

facet\_geo(~state, grid = “us\_state\_grid2”) +

theme\_bw()

**Overly complicated for what they’re worth:**

-cartograms

-hexbin maps