Personalized Map Art with OSM and R

Kayla Johnson

April 24th, 2021

Packages I am using

```
library(osmdata)
```

```
## Data (c) OpenStreetMap contributors, ODbL 1.0. https://www.openstreetmap.org/copyright
```

```
library(tigris)
```

```
## To enable
## caching of data, set ‘options(tigris_use_cache = TRUE)‘ in your R script or .Rprofile.
```

```
library(sf)
```

```
## Linking to GEOS 3.8.1, GDAL 3.1.4, PROJ 6.3.1
```

```
library(tidyverse)
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.0 --

## v ggplot2 3.3.3     v purrr   0.3.4
## v tibble  3.1.0     v dplyr   1.0.5
## v tidyr   1.1.3     v stringr 1.4.0
## v readr   1.4.0     v forcats 0.5.1

## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

I have loaded `osmdata`, `tigris`, `sf`, and `tidyverse`. Each of these can be installed from CRAN with `install.packages()` if you do not already have these packages. I will briefly introduce each package here:

- The `tidyverse` package is actually a collection of open source packages for reading, cleaning, manipulating, tidying, and visualizing data. If you are not already familiar with the `tidyverse`, I highly recommend learning it. The `tidyverse` website has learning resources listed here.
- The `sf` package provides simple features access for R. It makes working with irregular geometries in R easy, and here I will be using it to help plot our maps. Learn more about this really cool package on the `sf` website.

- The `osmdata` package is an R package for downloading and using data from OpenStreetMap (OSM). OSM is an open access map of the world created by interested people and is free to use. `osmdata` provides access to the vector data underlying OSM and puts it into a format we can easily use with `sf`. For a more in-depth introduction, see the vignette.
- The `tigris` package allows us to download and use TIGER/Line shapefiles from the US Census Bureau. Shapefiles contain legal boundaries and names as of the first of the year. `tigris` functions return simple features objects with a default year of 2019. There is some overlap between `tigris` and `osmdata` packages in terms of data types you could choose to use. The advantage of `tigris` is that it is pulling from the US Census Bureau data, which is very high quality and tracked by year. Learn more about the `tigris` package here.

Getting started

First let's see what kind of data OSM has. We can use the `available_features` function from `osmdata` or check out the OSM Map features wiki. The wiki includes more information on how features are tagged.

```
available_features()
```

```
##   [1] "4wd_only"                 "abandoned"
##   [3] "abutters"                 "access"
##   [5] "addr"                     "addr:city"
##   [7] "addr:conscriptionnumber" "addr:country"
##   [9] "addr:district"            "addr:flats"
##  [11] "addr:full"                "addr:hamlet"
##  [13] "addr:housename"           "addr:housenumber"
##  [15] "addr:inclusion"           "addr:interpolation"
##  [17] "addr:place"               "addr:postcode"
##  [19] "addr:province"            "addr:state"
##  [21] "addr:street"              "addr:subdistrict"
##  [23] "addr:suburb"              "admin_level"
##  [25] "aeroway"                  "agricultural"
##  [27] "alt_name"                 "amenity"
##  [29] "area"                     "atv"
##  [31] "backward"                 "barrier"
##  [33] "basin"                    "bdouble"
##  [35] "bicycle"                  "bicycle_road"
##  [37] "biergarten"               "boat"
##  [39] "border_type"              "boundary"
##  [41] "bridge"                   "building"
##  [43] "building:fireproof"       "building:flats"
##  [45] "building:levels"          "building:min_level"
##  [47] "building:soft_storey"     "bus_bay"
##  [49] "busway"                   "charge"
##  [51] "construction"             "covered"
##  [53] "craft"                    "crossing"
##  [55] "crossing:island"          "cuisine"
##  [57] "cutting"                  "cycleway"
##  [59] "denomination"             "destination"
##  [61] "diet"                     "direction"
##  [63] "dispensing"               "disused"
##  [65] "disused:shop"             "drink"
##  [67] "drive_in"                 "drive_through"
##  [69] "ele"                      "electric_bicycle"
##  [71] "electrified"            "embankment"
##  [73] "embedded_rails"         "emergency"
##  [75] "end_date"               "entrance"
##  [77] "est_width"              "fee"
##  [79] "fire_object:type"       "fire_operator"
##  [81] "fire_rank"              "foot"
##  [83] "footway"                "ford"
##  [85] "forestry"               "forward"
##  [87] "frequency"              "fuel"
##  [89] "gauge"                  "golf_cart"
##  [91] "goods"                  "hazmat"
##  [93] "healthcare"             "healthcare:counselling"
##  [95] "healthcare:speciality"  "height"
##  [97] "hgv"                    "highway"
##  [99] "historic"               "horse"
## [101] "ice_road"               "incline"
## [103] "industrial"             "inline_skates"
## [105] "inscription"            "internet_access"
## [107] "junction"               "kerb"
## [109] "landuse"                "lanes"
## [111] "lanes:bus"              "lanes:psv"
## [113] "layer"                  "leaf_cycle"
## [115] "leaf_type"              "leisure"
## [117] "lhv"                    "lit"
## [119] "location"               "man_made"
## [121] "maxaxleload"            "maxheight"
## [123] "maxlength"              "maxspeed"
## [125] "maxstay"                "maxweight"
## [127] "maxwidth"               "military"
## [129] "minspeed"               "mofa"
## [131] "moped"                  "motor_vehicle"
## [133] "motorboat"              "motorcar"
## [135] "motorcycle"             "motorroad"
## [137] "mountain_pass"          "mtb_scale"
## [139] "mtb:description"        "mtb:scale:imba"
## [141] "name"                   "name:left"
## [143] "name:right"             "narrow"
## [145] "natural"                "noexit"
## [147] "non_existent_levels"    "note"
## [149] "nudism"                 "office"
## [151] "official_name"          "old_name"
## [153] "oneway"                 "opening_hours"
## [155] "operator"               "organic"
## [157] "oven"                   "overtaking"
## [159] "parking:condition"      "parking:lane"
## [161] "passing_places"         "place"
## [163] "power"                  "priority_road"
## [165] "produce"                "proposed"
## [167] "protected_area"         "psv"
## [169] "public_transport"       "railway"
## [171] "railway:preserved"      "railway:track_ref"
## [173] "recycling_type"         "ref"
## [175] "religion"               "residential"
## [177] "roadtrain"              "route"
## [179] "sac_scale"                "service"
## [181] "service_times"            "shelter_type"
## [183] "shop"                     "sidewalk"
## [185] "site"                     "ski"
## [187] "smoothness"               "social_facility"
## [189] "sorting_name"             "speed_pedelec"
## [191] "start_date"               "step_count"
## [193] "substation"               "surface"
## [195] "tactile_paving"           "tank"
## [197] "tidal"                    "toilets:wheelchair"
## [199] "toll"                     "tourism"
## [201] "tracks"                   "tracktype"
## [203] "traffic_calming"          "traffic_sign"
## [205] "trail_visibility"         "tunnel"
## [207] "turn"                     "type"
## [209] "usage"                    "vehicle"
## [211] "vending"                  "voltage"
## [213] "water"                    "wheelchair"
## [215] "wholesale"                "width"
## [217] "winter_road"              "wood"
```

As you can see, there is a lot here. The tag that is probably going to be one of the most popular is "highway", so we'll use that in a moment. First, we need to declare a "boundary box" to tell `osmdata` which location we want the data pulled from. As I am a member of R-Ladies East Lansing, we'll start with East Lansing, MI.

We can use `getbb()` to define a boundary box for a place, or define it manually with minimum and maximum latitude and longitude values. We will call our boundary box `bbx`.

```
bbx <- getbb("East Lansing, MI")
bbx
```

```
##         min       max
## x -84.51350 -84.43887
## y  42.69726  42.80098
```

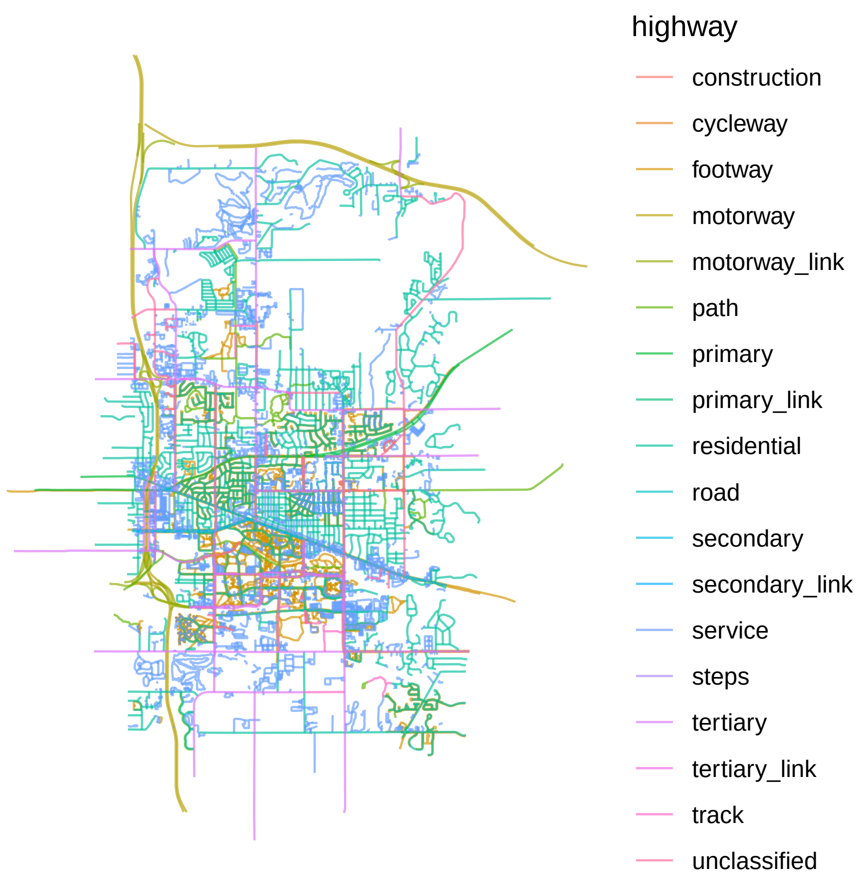
With `bbx` defined, we can now use `opq()` to query OSM for data in that location. The `add_osm_feature()` specifies the type of data we want pulled by the query, and `osmdata_sf()` ensures that the data returned to us is formatted in a way that `sf` can use to help us plot.

Warning: Some queries are larger than others. This particular query would take some time to complete for a large city such as New York City or London. Each feature such as "highway" can have multiple subdivisions and it may be prudent to only select some of these values if you are querying for a large area or large city. I will show an example of this selection in the next query.

```
highways <- bbx %>%
  opq()%>%
  add_osm_feature(key = "highway") %>%
  osmdata_sf()
```

Let's see what this looks like. We need `ggplot()` to create any plot, then we use `geom_sf()` to plot a map rather than say, `geom_bar` for a bar plot. The plotting data is stored in the `highway` object we created, specifically in the `osm_lines` part which we can access with `$`. An example of another option would be `osm_polygons` for storing shapes rather than lines for storing roads. I also colored the roads by `highway`, a variable stored in the OSM data, so we can see the many different types of roads.

```
ggplot() +
  geom_sf(data = highways$osm_lines,
          aes(color=highway),
          size = .4,
          alpha = .65) +
  theme_void() #theme void won't plot non-data (like axes) to give us a blank background
```



The road map is pretty dense using all types of roads. Sometimes selecting only certain values of a feature is better for visualization. First, I can check which values are available for the "highway" feature using `available_tags()`.

```
available_tags("highway")
```

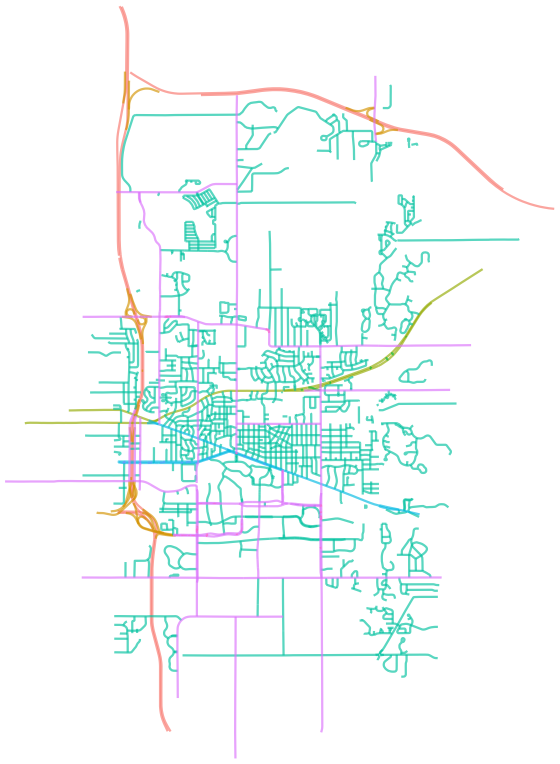
```
##  [1] "bridleway"              "bus_guideway"           "bus_stop"
##  [4] "busway"                 "construction"           "corridor"
##  [7] "crossing"               "cycleway"               "elevator"
## [10] "emergency_access_point" "emergency_bay"          "escape"
## [13] "footway"                "give_way"               "living_street"
## [16] "milestone"              "mini_roundabout"        "motorway"
## [19] "motorway_junction"      "motorway_link"          "passing_place"
## [22] "path"                   "pedestrian"             "platform"
## [25] "primary"                "primary_link"           "proposed"
## [28] "raceway"                "residential"            "rest_area"
## [31] "road"                   "secondary"              "secondary_link"
## [34] "service"                "services"               "speed_camera"
## [37] "steps"                    "stop"                     "street_lamp"
## [40] "tertiary"                 "tertiary_link"            "toll_gantry"
## [43] "track"                    "traffic_mirror"           "traffic_signals"
## [46] "trailhead"                "trunk"                    "trunk_link"
## [49] "turning_circle"           "turning_loop"             "unclassified"
```

I am going to choose the largest/main roads, plus "residential". In a very large city, "residential" may be unnecessary to get the density you want.

```
highways <- bbx %>%
  opq() %>%
  add_osm_feature(key = "highway",
                  value=c("motorway", "trunk",
                          "primary","secondary",
                          "tertiary","motorway_link",
                          "trunk_link","primary_link",
                          "secondary_link",
                          "tertiary_link", "residential")) %>%
  osmdata_sf()
```

We can see the smaller query gives a less crowded map.

```
ggplot() +
  geom_sf(data = highways$osm_lines,
          aes(color=highway),
          size = .4,
          alpha = .65) +
  theme(axis.line=element_blank(),
        axis.text.x=element_blank(),
        axis.text.y=element_blank(),
        axis.ticks=element_blank(),
        axis.title.x=element_blank(),
        axis.title.y=element_blank(),
        legend.position="none",
        panel.background=element_blank(),
        panel.border=element_blank(),
        panel.grid.major=element_blank(),
        panel.grid.minor=element_blank(),
        plot.background=element_blank()) #theme_void made explicit so I can remove legend
```



Getting more specific

Above, I used `getbb()` to set the boundary box for East Lansing, MI. However, we are not always interested in an entire city or an easily named area. In these cases we set our own boundary box. Here I am choosing coordinates that bound MSU's campus, which I have taken from Google Maps.

```
min_lon <- -84.493675; max_lon <- -84.462183
min_lat <- 42.711814; max_lat <- 42.735481
campus_bbx <- rbind(x=c(min_lon,max_lon),y=c(min_lat,max_lat))
colnames(campus_bbx) <- c("min","max")
```

I make two queries so that I can plot roads and sidewalks different sizes and alphas.

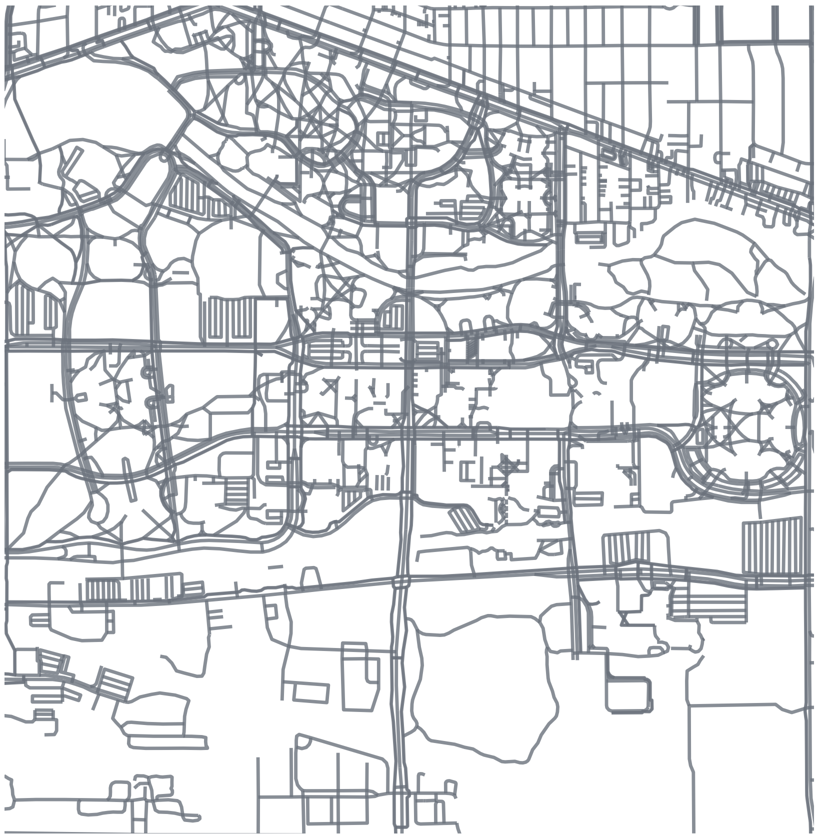
```
campus_sidewalks <- campus_bbx %>%
  opq() %>%
  add_osm_feature(key = "footway") %>%
  osmdata_sf()

campus_roads <- campus_bbx %>%
  opq() %>%
  add_osm_feature(key = "highway") %>%
  osmdata_sf()
```

```r
#setting color for roads
color_roads <- rgb(0.42,0.449,0.488)
```

Plotting roads and sidewalks of campus, using `coord_sf()` to cut off any roads that extend outside the map boundaries.

```r
ggplot() +
  geom_sf(data = campus_sidewalks$osm_lines,
          col = color_roads,
          size = 0.4,
          alpha = 0.65) +
  geom_sf(data = campus_roads$osm_lines,
          col = color_roads,
          size = 0.6,
          alpha = 0.8) +
  coord_sf(xlim = c(min_lon, max_lon),
           ylim = c(min_lat, max_lat),
           expand = FALSE) + #expand = FALSE will cut off any roads that extend outside your border
  theme(axis.line=element_blank(),
        axis.text.x=element_blank(),
        axis.text.y=element_blank(),
        axis.ticks=element_blank(),
        axis.title.x=element_blank(),
        axis.title.y=element_blank(),
        legend.position="none",
        panel.background=element_blank(),
        panel.border=element_blank(),
        panel.grid.major=element_blank(),
        panel.grid.minor=element_blank(),
        plot.background=element_rect(fill = NULL))
```



Adding water with tigris

MSU's campus features the Red Cedar River, so let's see if we can get that on the map.

This code gets us county lines for the state of Michigan and crops to our boundary box with `st_crop()`. In such a small area this pretty much equates to no lines, but it does give us a background and a location to add water to later.

```
counties_MI <- counties(state="MI",cb=T,class="sf",)
```

```
##   |                                                                              |
```

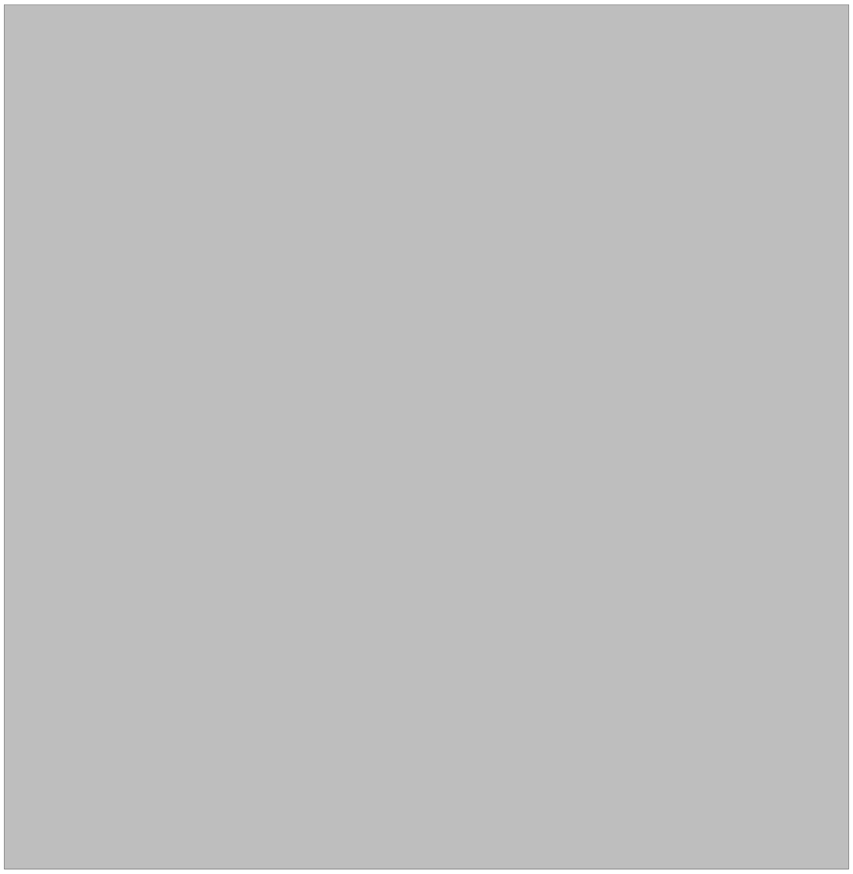
```
counties_MI <- st_crop(counties_MI,
                       xmin=min_lon,xmax=max_lon,
                       ymin=min_lat,ymax=max_lat)
```

```
## although coordinates are longitude/latitude, st_intersection assumes that they are planar
```

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

A quick look at our non-existent county lines on the MSU campus.

```
ggplot() +
  geom_sf(data=counties_MI,fill="gray",lwd=0) +
  coord_sf(xlim = c(min(campus_bbx[1,]), max(campus_bbx[1,])),
           ylim = c(min(campus_bbx[2,]), max(campus_bbx[2,])),
           expand = FALSE) +
  theme(legend.position = F) +
  theme_void()
```



To get the water in our location, we need some custom code, which I thank Dr. Esteban Moro for including in his blog post here.

```
get_water <- function(county_GEOID){
  area_water("MI", county_GEOID, class = "sf")
}
water <- do.call(rbind,
                 lapply(counties_MI$COUNTYFP,get_water))
```

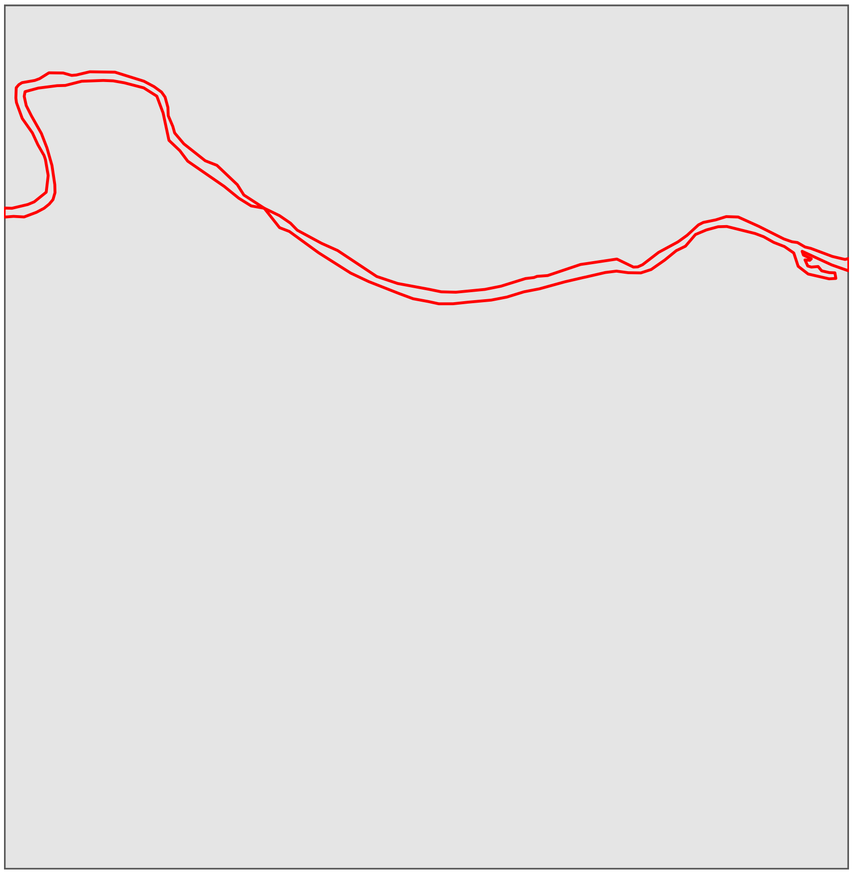
```
##   |                                                                              |
```

```
water <- st_crop(water,
                 xmin=min_lon,xmax=max_lon,
                 ymin=min_lat,ymax=max_lat)
```

```
## although coordinates are longitude/latitude, st_intersection assumes that they are planar
## Warning: attribute variables are assumed to be spatially constant throughout all
## geometries
```

If we take a look at the only water feature in our campus location, we can see the Red Cedar River.

```
ggplot() +
  geom_sf(data=counties_MI)+
  geom_sf(data=water,
          inherit.aes = F,
          col="red")+
  coord_sf(xlim = c(min(campus_bbx[1,]), max(campus_bbx[1,])),
         ylim = c(min(campus_bbx[2,]), max(campus_bbx[2,])),
         expand = FALSE)+
  theme(legend.position = F) + theme_void()
```



We can now use the water polygon we created to carve area out of our counties polygon, again with a custom function I took from Esteban Moro's blog post.

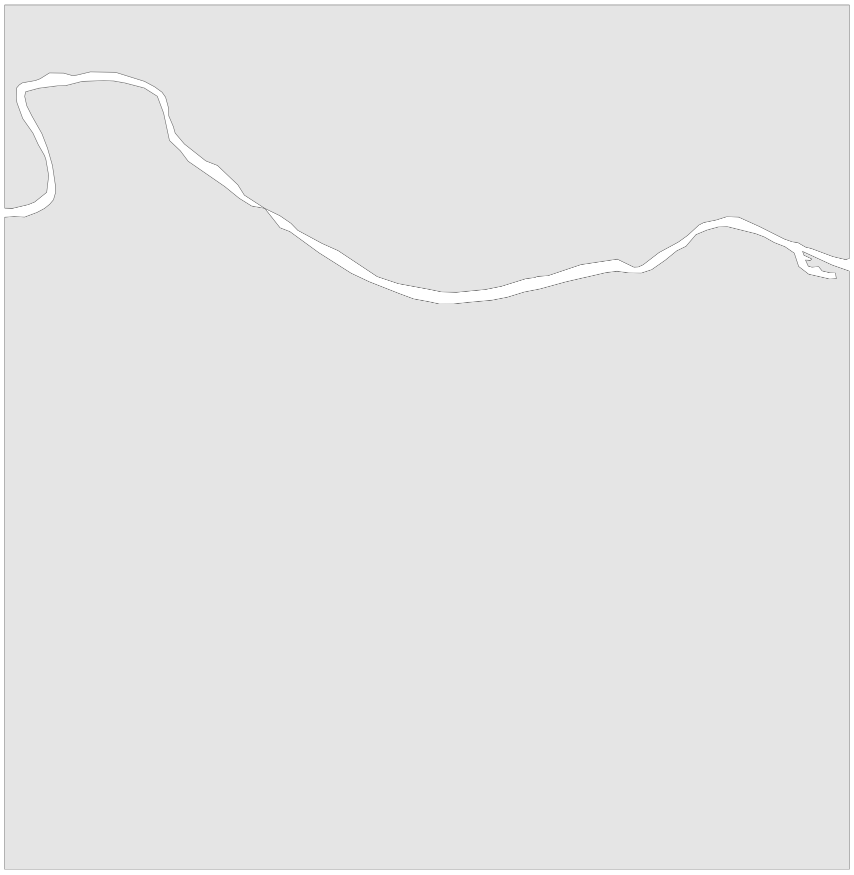
```
st_erase <- function(x, y) {
  st_difference(x, st_union(y))
}
counties_MI <- st_erase(counties_MI,water)
```

```
## although coordinates are longitude/latitude, st_union assumes that they are planar
```

```
## although coordinates are longitude/latitude, st_difference assumes that they are planar
## Warning: attribute variables are assumed to be spatially constant throughout all
## geometries
```

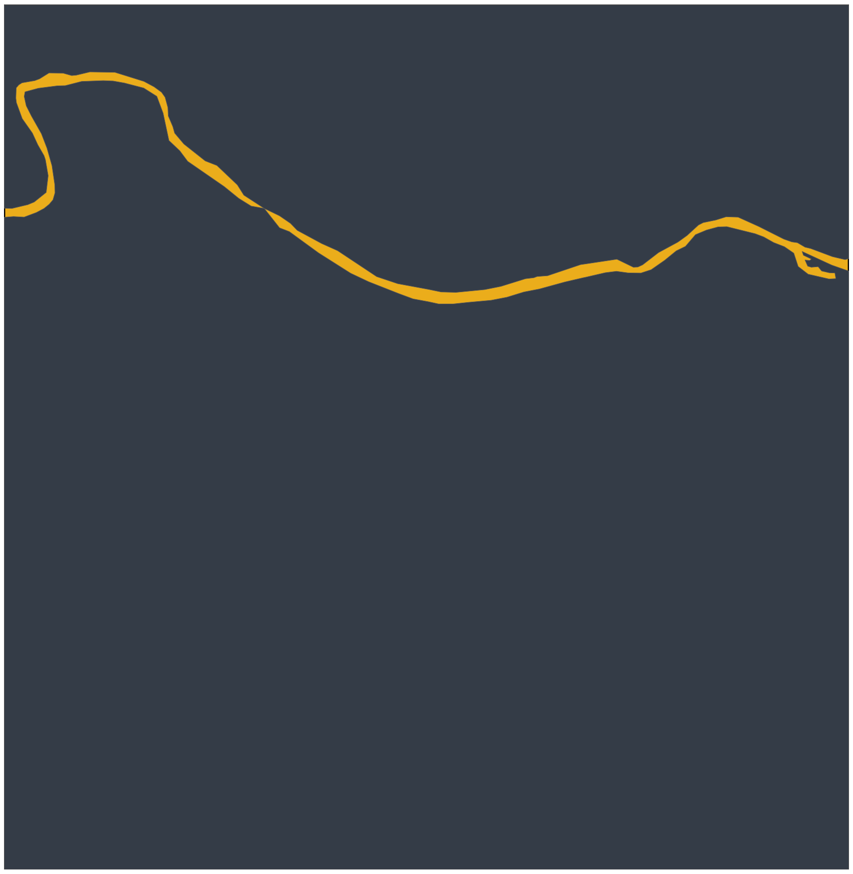
We can now see the water feature carved out of our geographical background.

```
ggplot() +
  geom_sf(data=counties_MI,
          lwd=0)+
  coord_sf(xlim = c(min(campus_bbx[1,]), max(campus_bbx[1,])),
         ylim = c(min(campus_bbx[2,]), max(campus_bbx[2,])),
         expand = FALSE)+
  theme(legend.position = F) + theme_void()
```



As with all ggplots, colors are easily customizable.

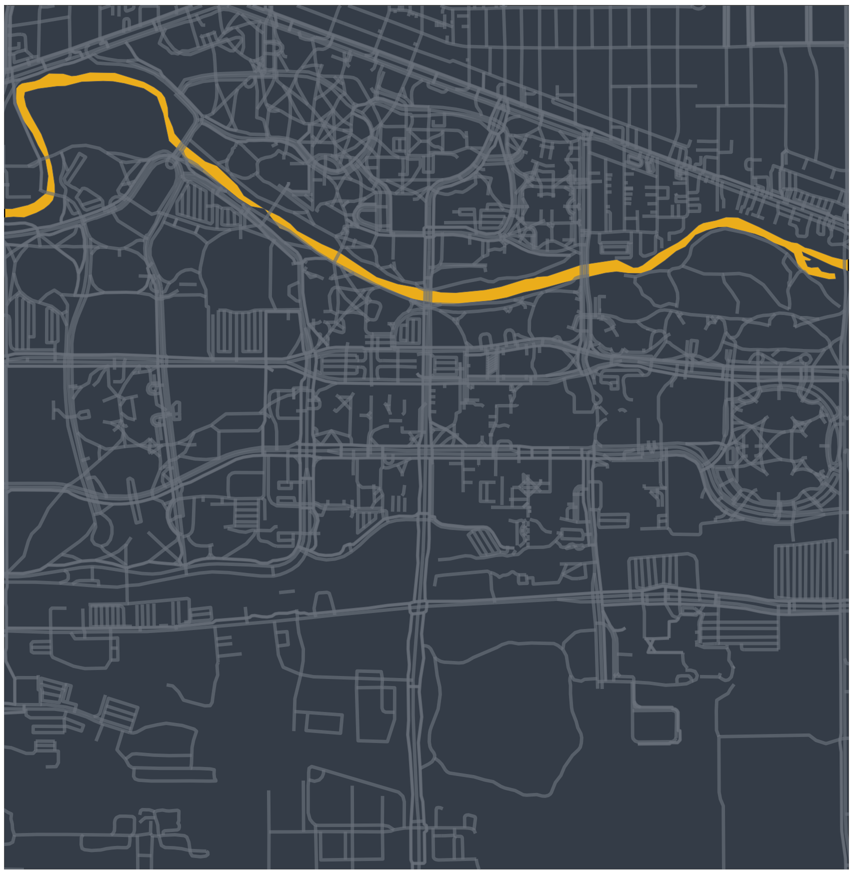
```
ggplot() +
  geom_sf(data=counties_MI,
          inherit.aes= FALSE,
          lwd=0.0,fill=rgb(0.203,0.234,0.277)) + #this is the grayish color
  coord_sf(xlim = c(min(campus_bbx[1,]), max(campus_bbx[1,])),
         ylim = c(min(campus_bbx[2,]), max(campus_bbx[2,])),
         expand = FALSE) +
  theme(legend.position = F) + theme_void() +
  theme(panel.background =
          element_rect(fill = rgb(0.92,0.679,0.105))) #this is the yellow color
```



Putting it all together

We have all of our parts, let's put them together.

```
ggplot() +
  geom_sf(data=counties_MI,
          inherit.aes= FALSE,
          lwd=0.0,fill=rgb(0.203,0.234,0.277))+
  geom_sf(data = campus_sidewalks$osm_lines,
          inherit.aes = FALSE,
          color=color_roads,
          size = .4,
          alpha = .65) +
  geom_sf(data = campus_roads$osm_lines,
          inherit.aes = FALSE,
          color=color_roads,
          size = .6,
          alpha = .65) +
  coord_sf(xlim = c(min(campus_bbx[1,]), max(campus_bbx[1,])),
           ylim = c(min(campus_bbx[2,]), max(campus_bbx[2,])),
           expand = FALSE) +
  theme(legend.position = F) + theme_void()+
  theme(panel.background =
          element_rect(fill = rgb(0.92,0.679,0.105)))
```



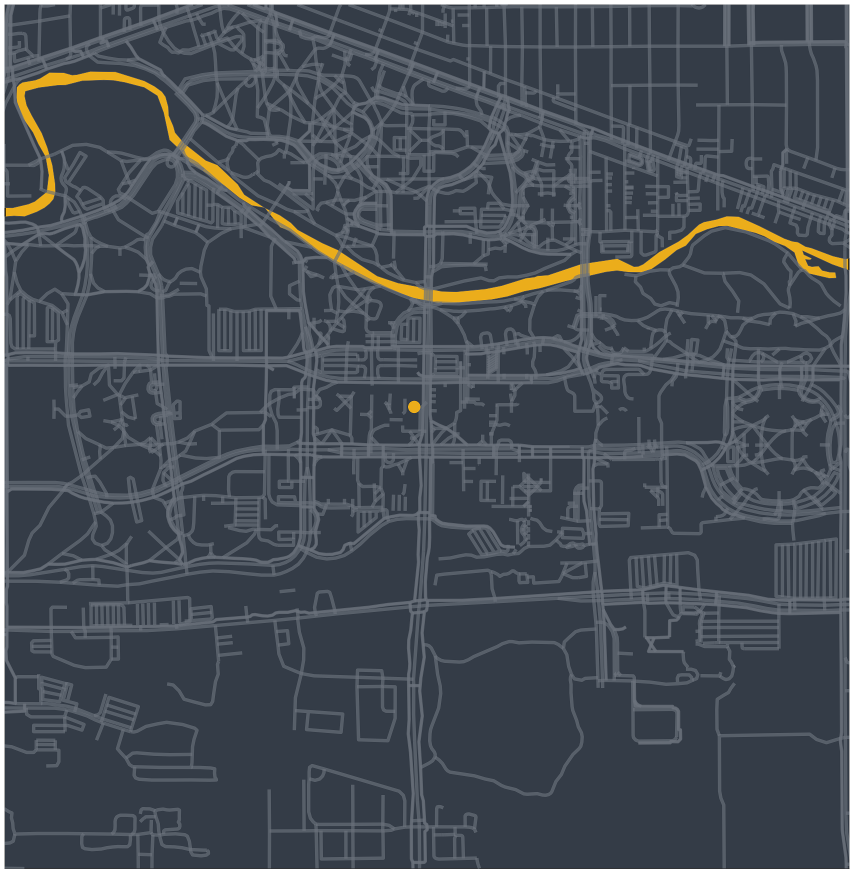
One last thing, to mark that special place. For my lab group, that is our alternate meeting place, The MSU Dairy Store.

First get the coordinates from Google Maps.

```r
dairy_store <- data.frame(x = -84.478404, y = 42.724459)
```

Use `geom_point()` to mark it on the map!

```r
ggplot() +
  geom_sf(data=counties_MI,
          inherit.aes= FALSE,
          lwd=0.0,fill=rgb(0.203,0.234,0.277)) +
  geom_sf(data = campus_sidewalks$osm_lines,
          inherit.aes = FALSE,
          color=color_roads,
          size = .4,
          alpha = .65) +
  geom_sf(data = campus_roads$osm_lines,
          inherit.aes = FALSE,
          color=color_roads,
          size = .6,
          alpha = .65) +
  geom_point(data = dairy_store,
             inherit.aes = FALSE,
             aes(x = x, y = y),
              color = rgb(0.92,0.679,0.105)) +
  coord_sf(xlim = c(min(campus_bbx[1,]), max(campus_bbx[1,])),
           ylim = c(min(campus_bbx[2,]), max(campus_bbx[2,])),
           expand = FALSE) +
  theme(legend.position = F) + theme_void()+
  theme(panel.background =
          element_rect(fill = rgb(0.92,0.679,0.105)))
```



Saving your map

We may want to print our map, add text, or overlap images in other programs such as Google Slides, ImageMagick, or InkScape. To do so, we need to save in high resolution. We will use the `ggsave()` function for this purpose. First, your final map must be saved into an R object (here, it is `final_map`). The width and height will be measured in terms of whatever the `units` argument is set to. In this case I use inches. Also very important is the `dpi` argument. Here it is set to 500, which means the image will be saved with high resolution at 500 dots per inch. This prevents us from printing large, blurry maps! I have not used it here, but `limitsize` may become an argument you need to use if you are trying to save a file larger than 50 inches by 50 inches.

```
ggsave(final_map,
       filename = "~/personal_campus_map.png",
       width = 24,
       height = 24,
       units = "in",
       dpi = 500)
```

Thanks! If this has interested you, I suggest checking out these other short blog posts: - http://estebanmoro.org/post/2020-10-19-personal-art-map-with-r/ - https://ggplot2tutor.com/tutorials/streetmaps - https://dominicroye.github.io/en/2018/accessing-openstreetmap-data-with-r/