Creating the streetmap

The first thing will be to get a beautiful map of the streets and boundaries of a given city. In this case, Boston, because is the place were I live. To download the streets we use the `osmdata` package to query OSM. First we define our bounding box, i.e. the area where we are going to query OSM:

```
library(osmdata)
bbx <- getbb("Boston, MA")
```

But sometimes you want a more specific area and you can create your own bounding box:

```
min_lon <- -71.28735; max_lon <- -70.900578
min_lat <- 42.245838; max_lat <- 42.453673
bbx <- rbind(x=c(min_lon,max_lon),y=c(min_lat,max_lat))
colnames(bbx) <- c("min","max")
```

OSM data has many layers (or "features"). You can find them here. They go from railways, vegetation to streets or points of interest. Let's see how many of them are related to `highway`

```
options(width = 60)
##  [1] "bridleway"              "bus_guideway"
##  [3] "bus_stop"               "construction"
##  [5] "corridor"               "crossing"
##  [7] "cycleway"               "elevator"
##  [9] "emergency_access_point" "emergency_bay"
## [11] "escape"                 "footway"
## [13] "give_way"               "living_street"
## [15] "milestone"              "mini_roundabout"
## [17] "motorway"               "motorway_junction"
## [19] "motorway_link"          "passing_place"
## [21] "path"                   "pedestrian"
## [23] "platform"               "primary"
## [25] "primary_link"           "proposed"
## [27] "raceway"                "residential"
## [29] "rest_area"              "road"
## [31] "secondary"              "secondary_link"
## [33] "service"                "services"
## [35] "speed_camera"           "steps"
## [37] "stop"                   "street_lamp"
## [39] "tertiary"               "tertiary_link"
## [41] "toll_gantry"            "track"
## [43] "traffic_mirror"         "traffic_signals"
## [45] "trailhead"              "trunk"
## [47] "trunk_link"             "turning_circle"
## [49] "turning_loop"           "unclassified"
```

They are a lot of them, but the main ones are `motorway`, `trunk`, `primary`, `secondary`, `tertiary` that distinguish between highways and other roads conecting cities or small towns. Let's collect them:

```
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")) %>%
  osmdata_sf()
```

We have 182 lines. Let's have a look to them. We will color them according to the label `highway`.

```
require(sf)
ggplot() +
  geom_sf(data = highways$osm_lines,
          aes(color=highway),
          size = .4,
          alpha = .65)+
  theme_void()
```

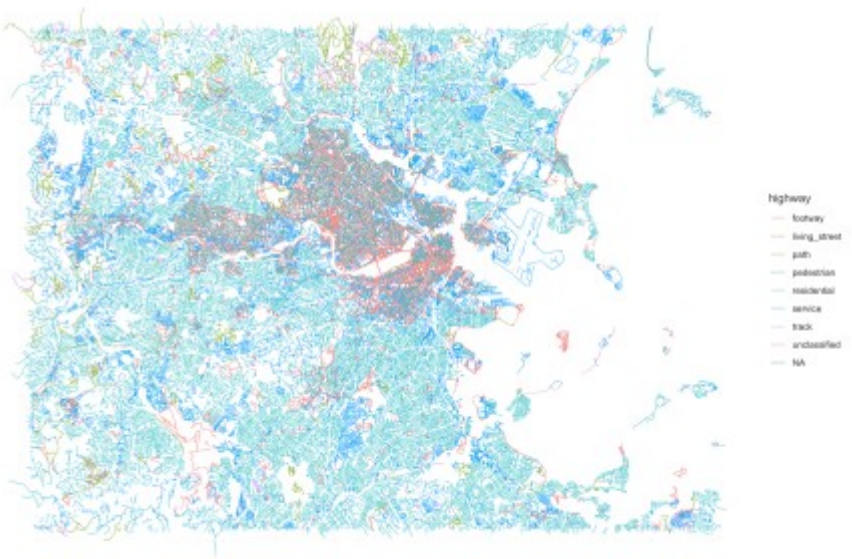
But we are also going to use the small streets, pedestrian paths or living streets that connect our neighborhoods (be advised, this is a large query so you might probably want to save it for later)

```
streets <- bbx %>%
  opq()%>%
  add_osm_feature(key = "highway",
                  value = c("residential", "living_street",
                            "service","unclassified",
                            "pedestrian", "footway",
                            "track","path")) %>%
  osmdata_sf()
```

This time, we have 286 lines. Let's have a look to them.

```
ggplot() +
  geom_sf(data = streets$osm_lines,
```

```
          aes(color=highway),
          size = .4,
          alpha = .65)+
  theme_void()
```



Those maps are already beautiful, right? You probably can see a darker area downtown Boston and Cambridge because of the larger density of streets there.

We are going to combine them and have different width for the highways and streets to highlight the importance of highways. At the same time, let's crop the streets to show only those within the bounding box.

```
color_roads <- rgb(0.42,0.449,0.488)
ggplot() +
  geom_sf(data = streets$osm_lines,
          col = color_roads,
          size = .4,
          alpha = .65) +
  geom_sf(data = highways$osm_lines,
          col = color_roads,
          size = .6,
          alpha = .8)+
  coord_sf(xlim = c(min_lon,max_lon),
           ylim = c(min_lat,max_lat),
         expand = FALSE)+
  theme(legend.position = F) + theme_void()
```

Adding the geography

Although those maps are very informative, they lack a key component of the geography: the land, rivers, coast lines that define our cities. In the maps of Boston above you are probably missing the shape of Charles River or those of the beautiful Boston Harbor Islands. In most maps around the internet, you will find those geographies are including using tiles which are basically a raster layer of images with different characteristics (names of the places, POI, etc). However, since we are going to print these maps in a large format, we need better resolution that typically those image tiles provide. Ideally we would like vector graphics like the lines we downloaded from OSM.

Instead of using raster images, we are going to use the polygons for the different counties around the Boston Area. The US Census has an amazing service called Tiger to retrieve those boundaries in ESRI's shapefile format. In `R` we can access those shapefiles using the `tigris` package. For example, these are the counties in the state of Massachusetts for the same bounding box as before:

We can see the coastline and some of the Boston Harbor Islands, but the Charles River or the ponds around Boston are missing. Also note that the resolution is very low. The `tigris` library

calls the TIGER API using the default (maximum) resolution `500k` (1:500k) which is not enough for our high resolution map.

To get a better geography layer we are going to carve out from these polygons the shapefile of the area of water, which can be also obtained from the TIGER services. Fortunately, those water lines have a much better resolution than the counties shapefiles.

```
get_water <- function(county_GEOID){
  area_water("MA", county_GEOID, class = "sf")
}
water <- do.call(rbind,
                 lapply(counties_MA$COUNTYFP,get_water))
water <- st_crop(water,
                 xmin=min_lon,xmax=max_lon,
                 ymin=min_lat,ymax=max_lat)
```

Let's have a look to them:

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

Finally, let's carve out the water polygons from the counties polygons.

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

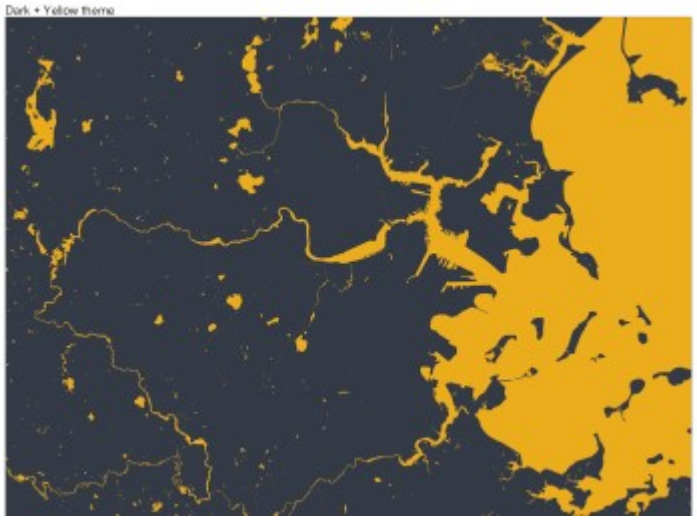
which looks like this (not showing the lines between counties)

```
ggplot() +
```

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

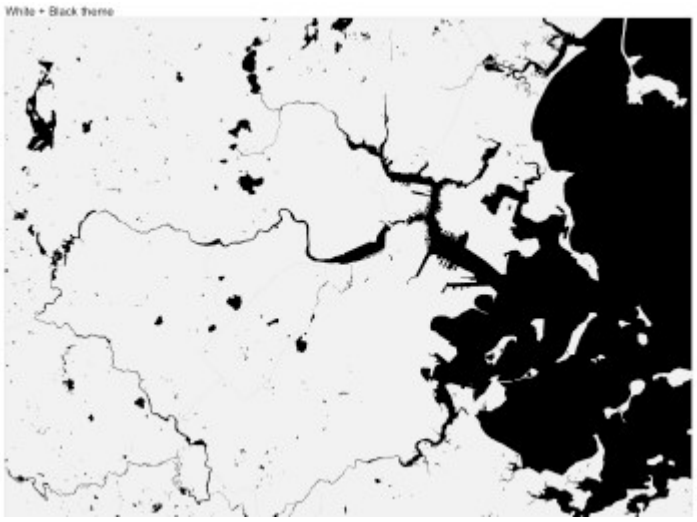

We can play now with the different colors for the water or land. Apart from the one above, I like these combinations (you might notice still the lines between counties. Don't worry, they disappear when we print them in large format):

```
ggplot() +
  geom_sf(data=counties_MA,
          inherit.aes= FALSE,
          lwd=0.0,fill=rgb(0.203,0.234,0.277))+
  coord_sf(xlim = c(min(bbx[1,]), max(bbx[1,])),
         ylim = c(min(bbx[2,]), max(bbx[2,])),
         expand = FALSE)+
  theme(legend.position = F) + theme_void()+
  theme(panel.background=
          element_rect(fill = rgb(0.92,0.679,0.105)))+
  ggtitle("Dark + Yellow theme")
```



```
ggplot() +
  geom_sf(data=counties_MA,
          inherit.aes= FALSE,
          lwd=0.0,fill="white")+
  coord_sf(xlim = c(min(bbx[1,]), max(bbx[1,])),
         ylim = c(min(bbx[2,]), max(bbx[2,])),
         expand = FALSE)+
  theme(legend.position = F) + theme_void()+
  theme(panel.background=
          element_rect(fill = rgb(0.9,0.9,0.9)))+
  ggtitle("Black + White theme")
```

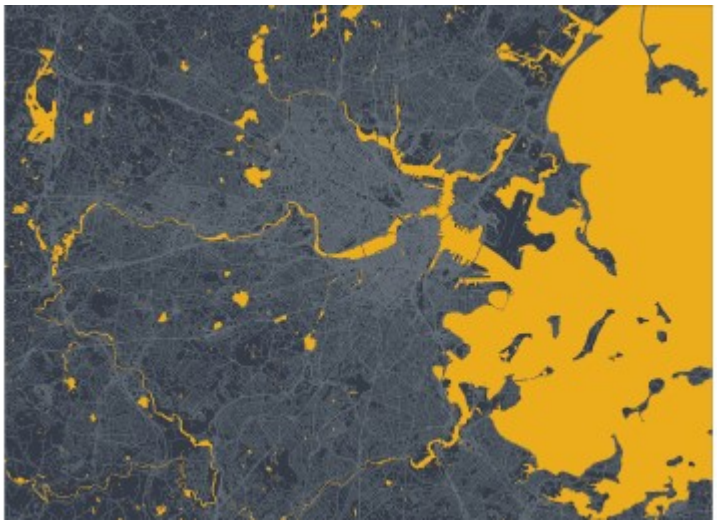

```
ggplot() +
  geom_sf(data=counties_MA,
          inherit.aes= FALSE,
          lwd=0.0,fill=rgb(0.95,0.95,0.95))+
  coord_sf(xlim = c(min(bbx[1,]), max(bbx[1,])),
         ylim = c(min(bbx[2,]), max(bbx[2,])),
         expand = FALSE)+
  theme(legend.position = F) + theme_void()+
  theme(panel.background=element_rect(fill = "black"))+
  ggtitle("White + Black theme")
```



Putting geography and streets together

Once we have the geography we can add the streets for our final streetmap:

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



Adding the personal mobility

As I said before, those maps are beautiful already. But I wanted to give them a more personal touch. As part of our research, we have been studying how people move in urban areas. Using high precision and anonymized data we have been able to understand for example how segregation happens on our streets, neighborhoods, working places. One of the things we found is that even if we happen to be on the same area, we tend to segregate ourselves and visit very different places. Even across the street!.

As part of that research, I've been collecting the locations that I visited in the last years in the Boston area. I basically used Google Location History which needs to be activated to work. Once it is activated, it starts saving the places you go. You can even visualize your movements using the Google Maps Timeline. Or you can download it in using the Google Takeout tool. Of course, if you don't access to your location history or it is deactivated, you can always list the most relavant locations in the area your live and create a table with the right format (see below).

Once you download your location, you will get two sets of files. The first one is called `Location History.json` that contains a lot of location events (including latitute, longitud, timestamp) and a directory called `Semantic Location History` that has all the places visited by year and month. Make sure when you download your data from Google Takeout that this semantic history is enabled. We will use the semantic one, since it contains the list of places visted (inferred by Google).

```
require(jsonlite)
require(data.table)
path <- "./Semantic Location History/"
files <- list.files(path,pattern="*.json",recursive=T)
get_locations <- function(f){
  data <- jsonlite::fromJSON(paste0(path,f))
  tlObj <- data$timelineObjects$placeVisit
  loc <- cbind(tlObj$location,tlObj$duration)
  tt <- as.numeric(loc$startTimestampMs)/1000
  loc$time<-as.POSIXct(tt,origin = "1970-01-01")
  #conver longitude & latitude from E7 to GPS
  loc$lat = loc$latitudeE7 / 1e7
  loc$lon = loc$longitudeE7 / 1e7
  loc <- data.table(loc)
  loc <- loc[,c("placeId","time","lat","lon")]
  loc <- loc[!is.na(lon)]
}
locs.df<-rbindlist(lapply(files,get_locations))
```

The data includes the Google Id for the place, the timestamp of the visit its longitud/latitude position. Of course, I'm not showing you my personal data. This table you see below was generated sampling from a bunch of random locations in the Boston area around my working place (MIT Media Lab).

```
head(locs.df) %>% kbl() %>% kable_styling()
```

placeId	time	lat	lon
ChIJ780101a191a	2020-02-20 16:55:59	42.36126	-71.08695
ChIJ140101a191a	2020-03-16 23:42:52	42.37321	-71.09359
ChIJ740101a191a	2020-05-31 15:38:12	42.37020	-71.06178
ChIJ320101a191a	2020-06-03 21:20:21	42.37311	-71.12237
ChIJ420101a191a	2020-02-10 12:15:43	42.37220	-71.06315
ChIJ840101a191a	2020-05-25 21:53:40	42.38635	-71.22257

One way to add this personal information on the map would be to add the location points to the previous map. But for most of us that would be just a bunch (a hundred in my case) of points which are very close to each other. Since we already have the streets in the map we are going to do something different: to plot the routes we took to go from one place to the next one. Two comments about this:

- The semantic locations do not include the routes taken to visit those places. Those can be calculated using the network of streets already downloaded.
- However, most of the days we will only have one significant place visited. That means we cannot calculate any route that day. What we are going to do is to assume that each day starts at your home location and ends up in the same place so at least you have two

routes for those days with at least one location in the Google dataset. In my case, I'm going to choose the MIT Media Lab as my home location (I spent a lot of time there last year, but it's definitely not my home 😄)

We first create a function that return the routes between all the locations within a given day. To get the routes between two locations we used the function `osrmRoute` in the `osrm` package which give use the shortest path between two points in the network of streets.

```
require(osrm)
daily_routes <- function(date){
  ll<-locs.df[as.Date(time)==date,c("lon","lat")]
  #add home early in the morning
  ll<-rbind(data.table(lon=-71.087658,lat=42.36),ll)
  #add home late in the day
  ll<-rbind(ll,data.table(lon=-71.087658,lat=42.36))
  route <- NULL
  for(j in 2:nrow(ll)){
    p1 <- c(ll$lon[j-1],ll$lat[j-1])
    p2 <- c(ll$lon[j],ll$lat[j])
    oo <- osrmRoute(src=p1,dst=p2,returnclass = "sf",
                    overview="full")
    route <- rbind(route,oo)
  }
  route
}
```

Note that those routes might not be the *actual route* taken to go between those points. The `osrmRoute` algorithm calculate the shortest path only.

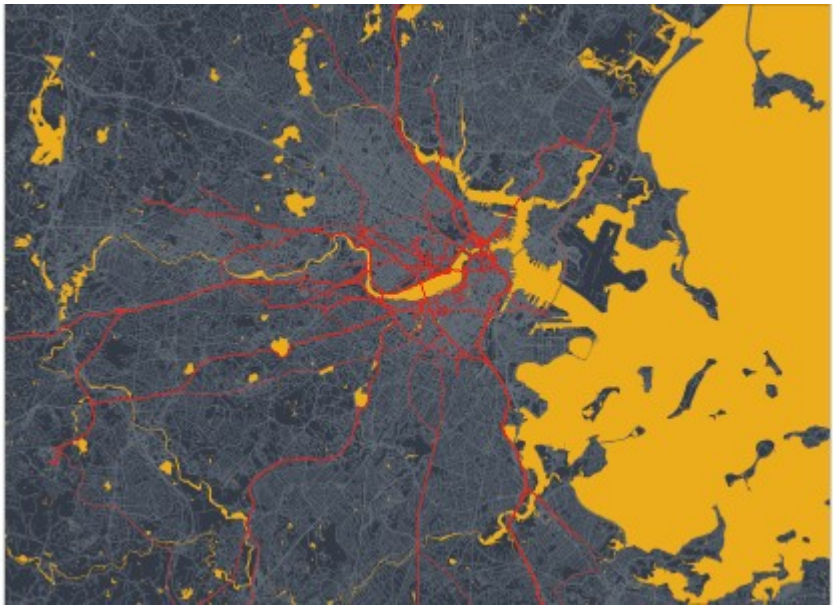
And then we get all the routes for the whole period (this will take some time)

```
dates <- unique(as.Date(locs.df$time))
routes <- do.call(rbind,
                  lapply(dates),daily_routes))
```

We are now ready to plot our geography + streets + personal routes.

```
final_map <- ggplot() +
  geom_sf(data=counties_MA,
          inherit.aes= FALSE,
          lwd=0.0,fill=rgb(0.203,0.234,0.277))+
  geom_sf(data = streets$osm_lines,
          inherit.aes = FALSE,
          color=color_roads,
          size = .4,
          alpha = .65) +
  geom_sf(data = highways$osm_lines,
          inherit.aes = FALSE,
          color=color_roads,
          size = .6,
          alpha = .65) +
  geom_sf(data=st_geometry(routes),
          inherit.aes = FALSE,col="red",alpha=0.5)+
  coord_sf(xlim = c(min(bbx[1,]), max(bbx[1,])),
```

```
            ylim = c(min(bbx[2,]), max(bbx[2,])),
            expand = FALSE) +
  theme(legend.position = F) + theme_void()+
  theme(panel.background=
          element_rect(fill=rgb(0.92,0.679,0.105)))
final_map
```



Printing them out

Finally let's print it out in high resolution. In my case, I used 24 inches by 36 inches, but you can chose other size:

```
ggsave(final_map,
       filename = "my_personal_map_art.png",
       scale = 1,
       width = 36,
       height = 24,
       units = "in",
       dpi = 500)
```

Note the `dpi = 500` which forces to save the image with high resolution (500 dots per inch). This is specially important if you want to print your poster in large dimensions. The file generated is large (around 40Mb).

The last step is to send to print or frame. You can use many services online to do that, but you need one that allows you to send such a big file. In my case I ended up using the Mounted Canvas Print - 36" x 24" at vistaprint.

And here is the final result:

I hope you enjoyed my small project to produce personal map art with R.