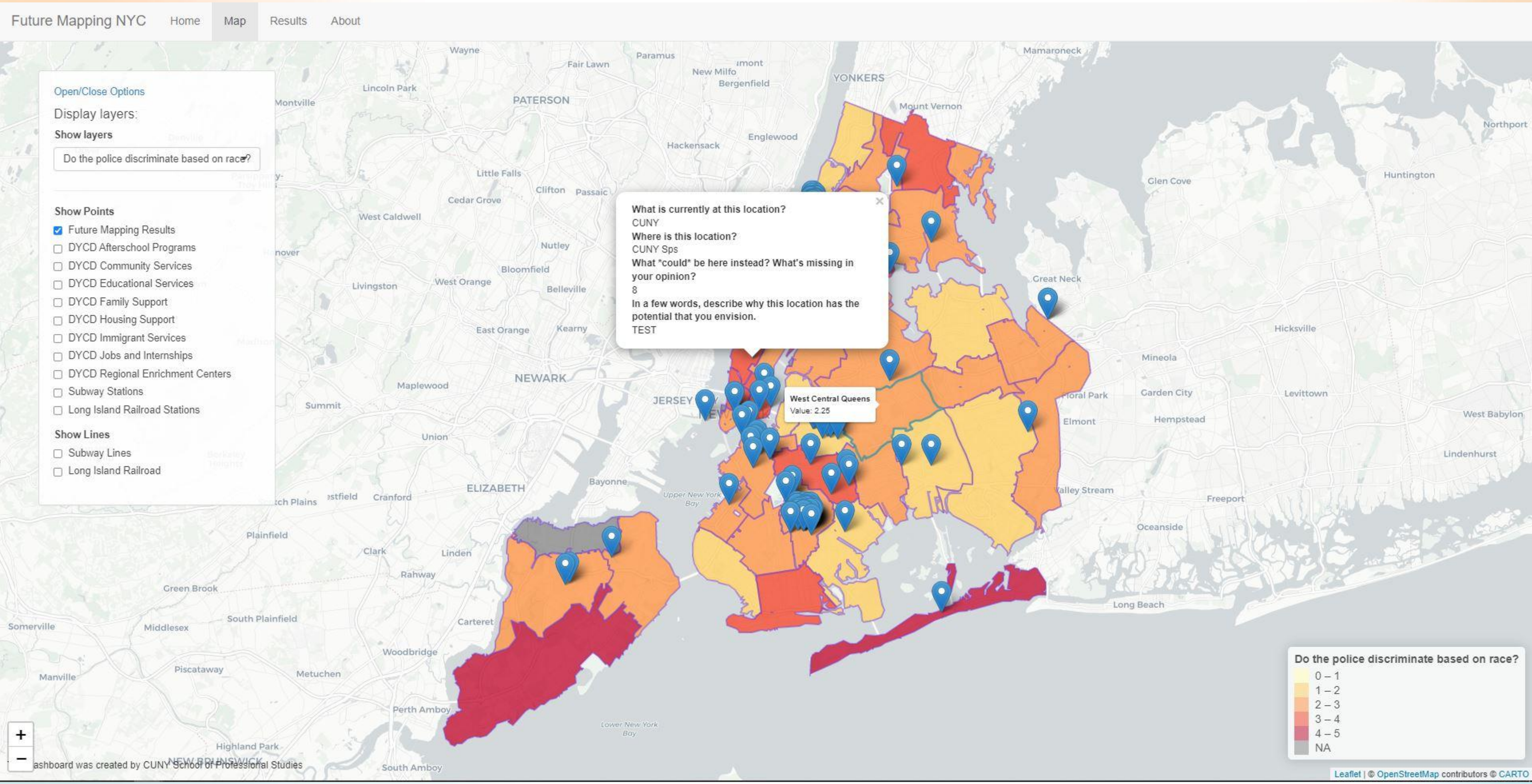
AtlasMaker

Easily make multiple Leaflet maps in a Shiny app

Rachel Greenlee & Zach Palmore

Quick overview

1. What problem are we trying to solve?
2. Process of development
3. See AtlasMaker
4. Review code for AtlasMaker



Future Mapping NYC Home Map Results About

Open/Close Options

Display layers:

Show layers

Do the police discriminate based on race?

Show Points

- [x] Future Mapping Results
- [ ] DYCD Afterschool Programs
- [ ] DYCD Community Services
- [ ] DYCD Educational Services
- [ ] DYCD Family Support
- [ ] DYCD Housing Support
- [ ] DYCD Immigrant Services
- [ ] DYCD Jobs and Internships
- [ ] DYCD Regional Enrichment Centers
- [ ] Subway Stations
- [ ] Long Island Railroad Stations

Show Lines

- [ ] Subway Lines
- [ ] Long Island Railroad

What is currently at this location?
CUNY
Where is this location?
CUNY Sps
What *could* be here instead? What's missing in your opinion?
8
In a few words, describe why this location has the potential that you envision.
TEST

West Central Queens
Value: 2.25

Do the police discriminate based on race?

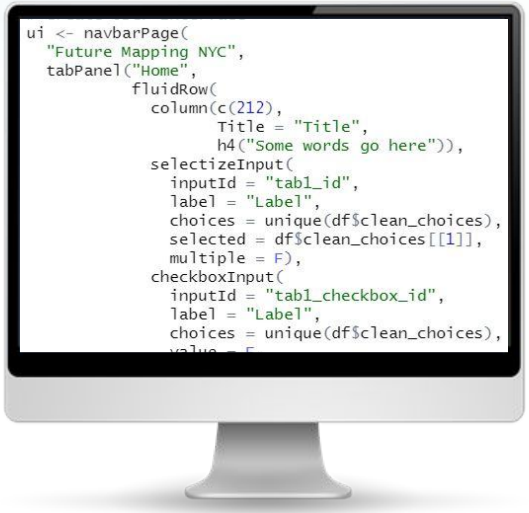
🟨	0 – 1
🟨	1 – 2
🟧	2 – 3
🟥	3 – 4
🟥	4 – 5
⬜	NA

ashboard was created by CUNY School of Professional Studies

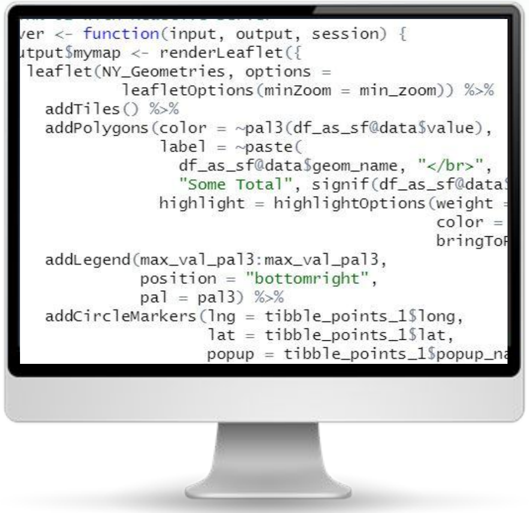
Leaflet | © OpenStreetMap contributors © CARTO

The Challenge

How would you recreate this?



```
ui <- navbarPage(
  "Future Mapping NYC",
  tabPanel("Home",
           fluidRow(
             column(c(212),
                    Title = "Title",
                    h4("Some words go here")),
             selectizeInput(
               inputId = "tab1_id",
               label = "Label",
               choices = unique(df$clean_choices),
               selected = df$clean_choices[[1]],
               multiple = F),
             checkboxInput(
               inputId = "tab1_checkbox_id",
               label = "Label",
               choices = unique(df$clean_choices),
               value = F
```



```
ver <- function(input, output, session) {
utput$mymap <- renderLeaflet({
 leaflet(NY_Geometries, options =
           leafletOptions(minZoom = min_zoom)) %>%
   addTiles() %>%
   addPolygons(color = ~pal3(df_as_sf@data$value),
               label = ~paste(
                 df_as_sf@data$geom_name, "</br>",
                 "Some Total", signif(df_as_sf@data
               highlight = highlightOptions(weight =
                                            color =
                                            bringToF
   addLegend(max_val_pal3:max_val_pal3,
             position = "bottomright",
             pal = pal3) %>%
   addCircleMarkers(lng = tibble_points_1$long,
                    lat = tibble_points_1$lat,
                    popup = tibble_points_1$popup_na
```

The Challenge

How would you recreate this?

Input cleaned polygon choices specific to each dataset (or merge all data into one df)

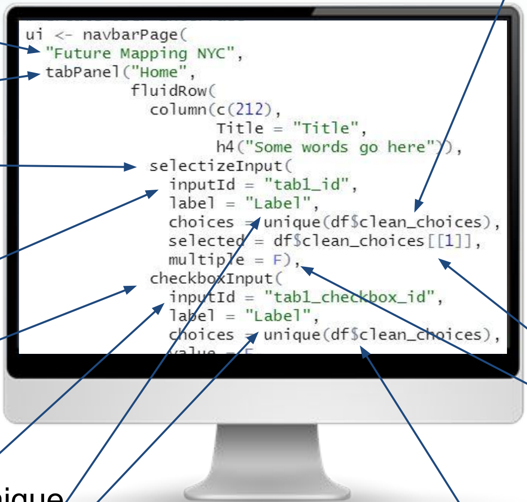
Add Title

Map within a tab

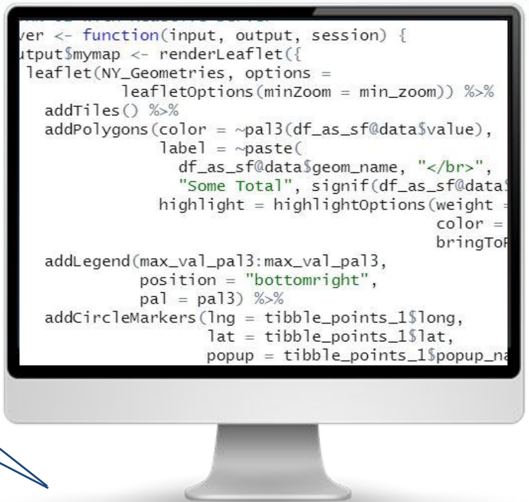
Create drop-down list of filter options for polygons

Specify unique tab ID for the polygons

Create checkboxes for points (lat/long)



```r
ui <- navbarPage(
  "Future Mapping NYC",
  tabPanel("Home",
           fluidRow(
             column(c(212),
                    Title = "Title",
                    h4("Some words go here")),
             selectizeInput(
               inputId = "tab1_id",
               label = "Label",
               choices = unique(df$clean_choices),
               selected = df$clean_choices[[1]],
               multiple = F),
             checkboxInput(
               inputId = "tab1_checkbox_id",
               label = "Label",
               choices = unique(df$clean_choices),
```



```r
ver <- function(input, output, session) {
utput$mymap <- renderLeaflet({
 leaflet(NY_Geometries, options =
              leafletOptions(minZoom = min_zoom)) %>%
   addTiles() %>%
   addPolygons(color = ~pal3(df_as_sf@data$value),
               label = ~paste(
                 df_as_sf@data$geom_name, "</br>",
                 "Some Total", signif(df_as_sf@data$
               highlight = highlightOptions(weight =
                                            color =
                                            bringToF
   addLegend(max_val_pal3:max_val_pal3,
             position = "bottomright",
             pal = pal3) %>%
   addCircleMarkers(lng = tibble_points_1$long,
                    lat = tibble_points_1$lat,
                    popup = tibble_points_1$popup_na
```

Specify another unique ID for the points

Input clean point options using specific dataset or merged df

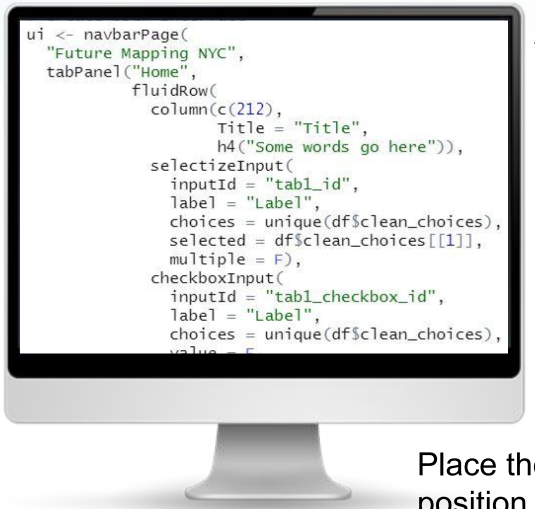
Choose one and only one from df filter choices

Provide labels to make sense of the data once visualized

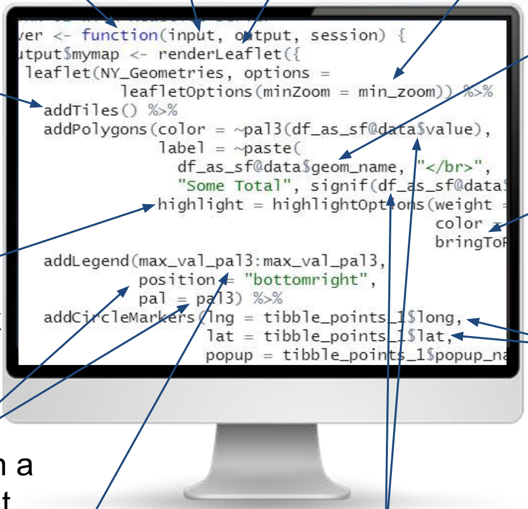
Important: This can be reused easier than server

The Challenge

How would you recreate this?



```r
ui <- navbarPage(
  "Future Mapping NYC",
  tabPanel("Home",
           fluidRow(
             column(c(212),
                    Title = "Title",
                    h4("Some words go here")),
             selectizeInput(
               inputId = "tab1_id",
               label = "Label",
               choices = unique(df$clean_choices),
               selected = df$clean_choices[[1]],
               multiple = F),
             checkboxInput(
               inputId = "tab1_checkbox_id",
               label = "Label",
               choices = unique(df$clean_choices),
```



```r
ver <- function(input, output, session) {
utput$mymap <- renderLeaflet({
 leaflet(NY_Geometries, options =
           leafletOptions(minZoom = min_zoom)) %>%
   addTiles() %>%
   addPolygons(color = ~pal3(df_as_sf@data$value),
               label = ~paste(
                 df_as_sf@data$geom_name, "</br>",
                 "Some Total", signif(df_as_sf@data$
               highlight = highlightOptions(weight =
                                            color =
                                            bringToF
   addLegend(max_val_pal3:max_val_pal3,
             position = "bottomright",
             pal = pal3) %>%
   addCircleMarkers(lng = tibble_points_1$long,
                    lat = tibble_points_1$lat,
                    popup = tibble_points_1$popup_na
```

Setup function with input, output for user session

Filter df input prior to mapping or lose reactivity

Use function to render leaflet map in Shiny using input from setup

Set limits on the user view to prohibit super zoom

Add OSM Grid Tiles (excluding provider Tiles)

Create a popup for each point with a polygon name (borough, county…)

Label the polygon's popup box and set opacity

Select ordering of layers (polygons, polylines, points)

Specify df latitude and longitude values for points

Place the legend in a position and color it using the df's color palette function

Set minimum and maximum values for legend

Access Polygons from Spatial Data Frame and map each value to a color with your pal()

(Pre-server) Create a color palette function finds the bounds of your data's values and assigns colors to them on a scale

(Pre-server) Find the minimum and maximum values to create the legend for each set of df values

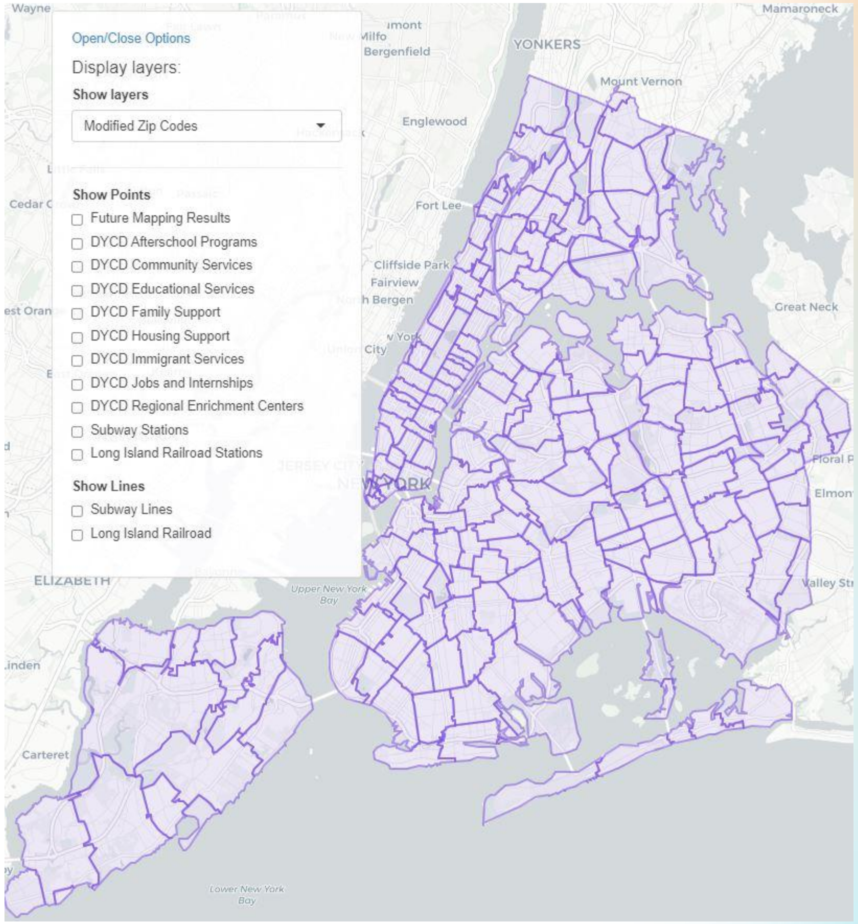
The Challenge

What if you had to build more and customize each?

At its core:
Repetitive, isolated,
manually exhausting

Would you change your approach?

1. **Repetitive** - copy and paste same UI and Server chunks for each layer which consists of 25+ polygons, polylines, and points
2. **Isolated** - continuously edit and test each time new data is added to ensure the application functions as intended and still works with existing data
3. **Manual** - color palette selection requires knowing your legend bounds and tailoring each within that dataset's map


Open/Close Options

Display layers:

Show layers

Modified Zip Codes

Show Points

- [ ] Future Mapping Results
- [ ] DYCD Afterschool Programs
- [ ] DYCD Community Services
- [ ] DYCD Educational Services
- [ ] DYCD Family Support
- [ ] DYCD Housing Support
- [ ] DYCD Immigrant Services
- [ ] DYCD Jobs and Internships
- [ ] DYCD Regional Enrichment Centers
- [ ] Subway Stations
- [ ] Long Island Railroad Stations

Show Lines

- [ ] Subway Lines
- [ ] Long Island Railroad

Core Focus Areas

How should we come up with the best solution for this challenge?

Automatic

Eliminate the need to specify a multitude of components and automatically adapt to new data and inputs

Modular

Leveraging the use of modules that can stack with one another for increased functionality and reactivity

Design

Integrate with existing frameworks to create functional yet visually appealing themed maps using Leaflet and Shiny

Time

Reduce the amount of spent on repetitive tasks like cleaning, copying, and minor editing during development

Our Plan

Development of AtlasMaker v0.9

1/28

Exploration

Explore Future Mapping App, identify areas for improvement, create plan to make improvements

2/14

Map Development

Develop a basic functional one-page shiny app with skeleton mapping feature

2/28

Module Conversion

Convert this shiny app into a module to allow multiple themed tabs that leverage the use of modules

3/14

Functionality Testing

Run tests on modules with different datasets and troubleshoot to improve functionality

5/1

Customization & Design

Build in options for the user to customize the map design and themes

5/16

Package Conversion

Convert modularized shiny app with leaflet map into package for open use

5/20

AtlasMaker Demo Map

version 0.9

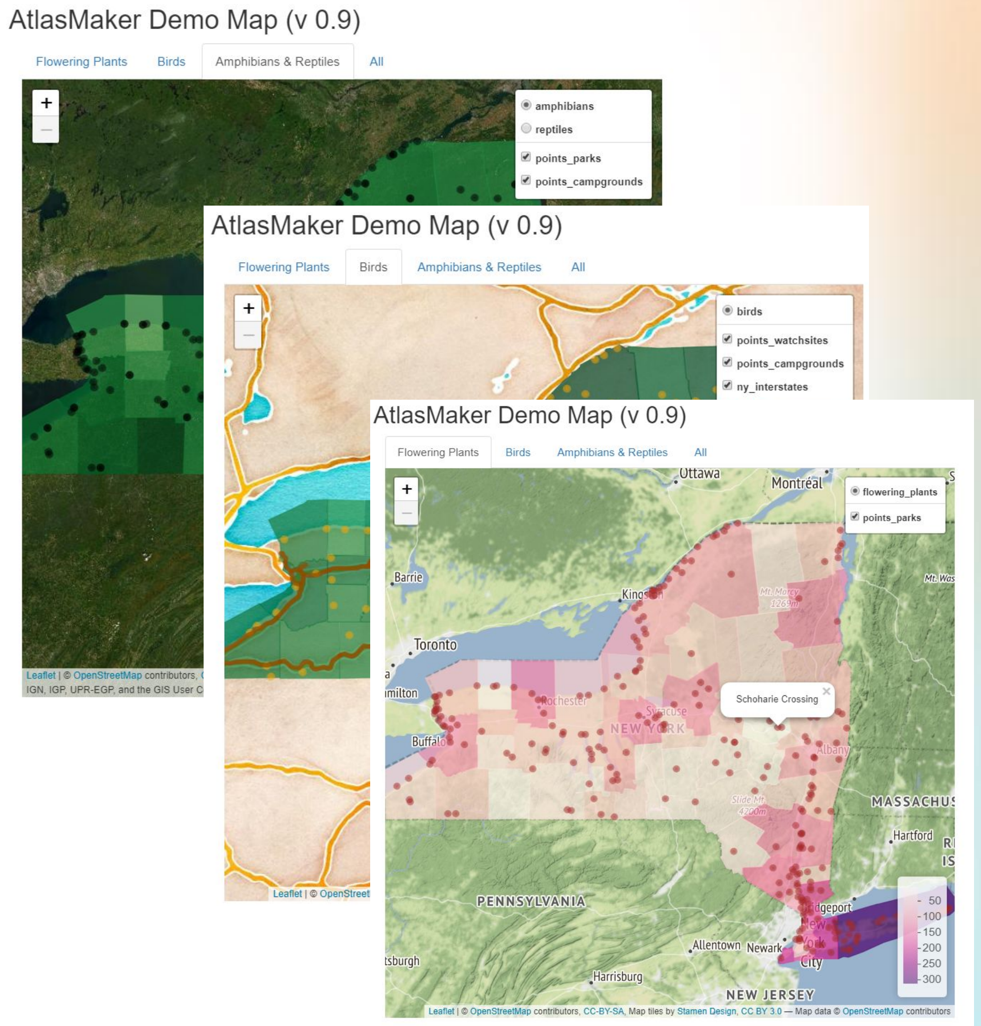
Demo Map

Biodiversity data by New York counties

atlas:
***noun* a book of maps**
or charts

Make multiple, related maps quickly

1. **Quick** - eliminate need to copy/paste/edit code blocks to recreate maps that share much of the same structure
2. **Design** - good dataviz requires less clutter, split your data across multiple maps for ease of interpretation
3. **Story** - data is most meaningful in context, group together appropriate layers by themed tab
4. **Ease** - if you know how to make one Leaflet map in Shiny, you can use AtlasMaker to make many



How to use AtlasMaker

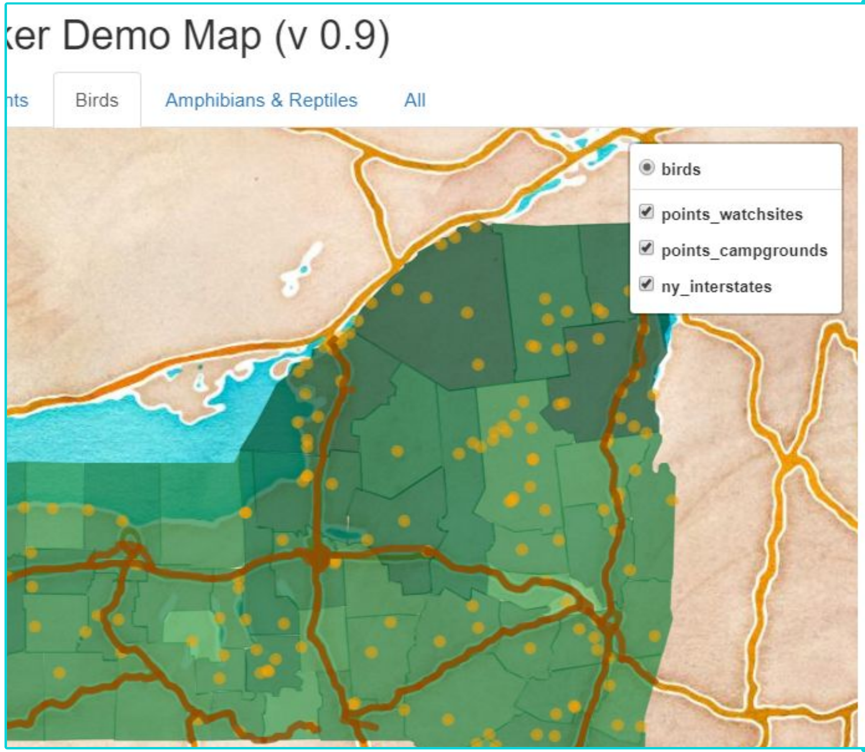
easy, non-repetitive code

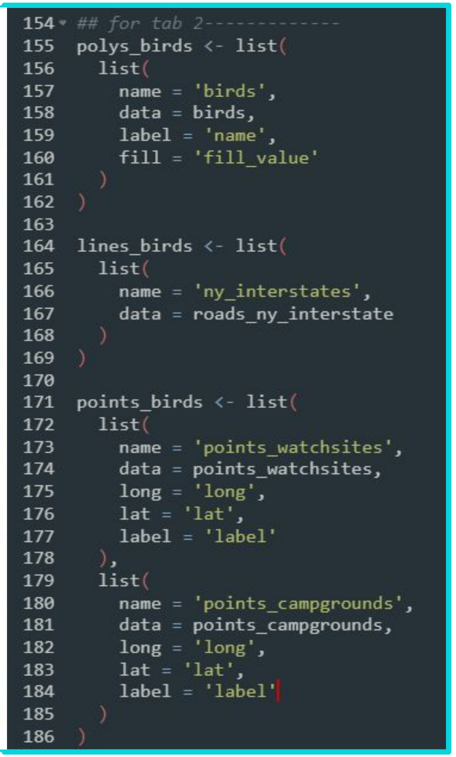
Steps to AtlasMaker

1. Clean your data, use appropriate geospatial format (sp/sf packages are useful).
2. Build lists of any polygons, points, and polylines needed for any map tab.
3. Build a map_server per map tab, with as few/many arguments as you want
4. Enjoy your atlas!

Prep lists per map

These feed into the module-based AtlasMaker library.

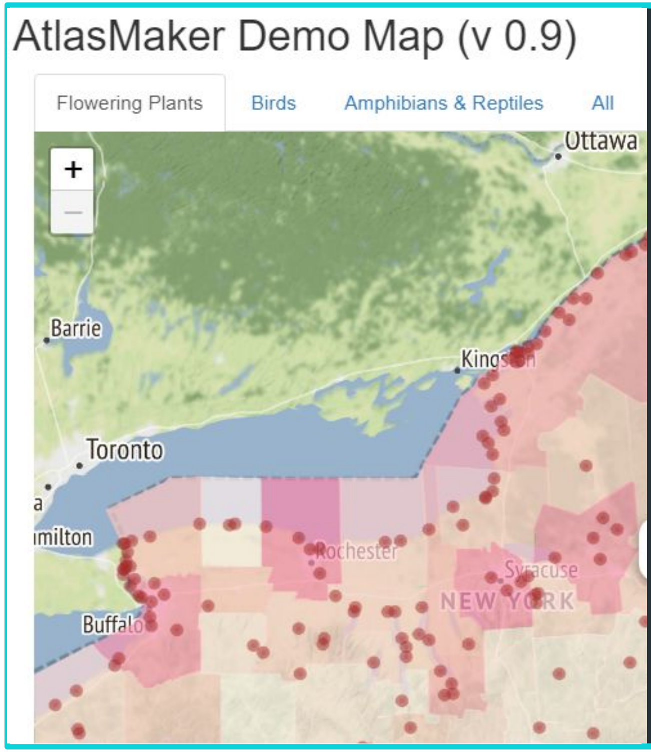


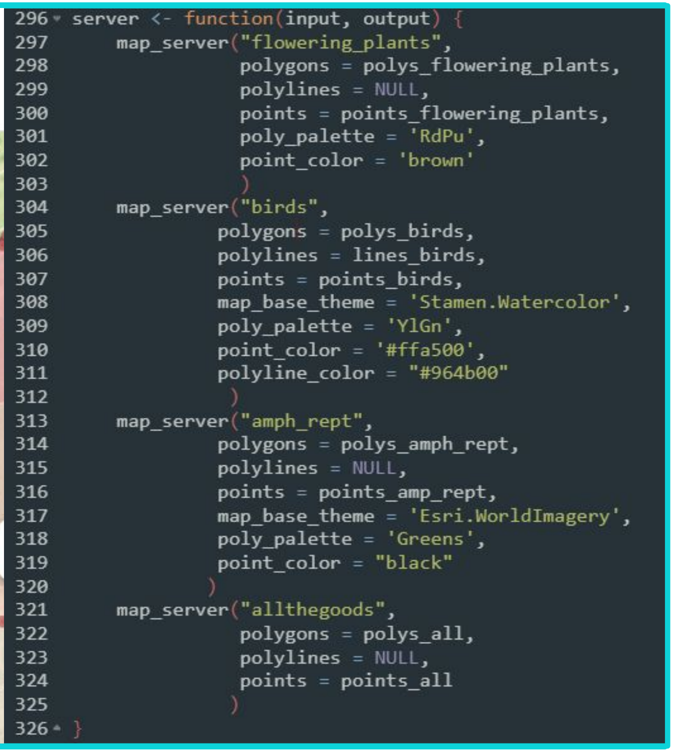


```
## for tab 2-------------
polys_birds <- list(
  list(
    name = 'birds',
    data = birds,
    label = 'name',
    fill = 'fill_value'
  )
)

lines_birds <- list(
  list(
    name = 'ny_interstates',
    data = roads_ny_interstate
  )
)

points_birds <- list(
  list(
    name = 'points_watchsites',
    data = points_watchsites,
    long = 'long',
    lat = 'lat',
    label = 'label'
  ),
  list(
    name = 'points_campgrounds',
    data = points_campgrounds,
    long = 'long',
    lat = 'lat',
    label = 'label'
  )
)
```

Create a map_server per map

These feed into the module-based AtlasMaker library.





```
server <- function(input, output) {
    map_server("flowering_plants",
               polygons = polys_flowering_plants,
               polylines = NULL,
               points = points_flowering_plants,
               poly_palette = 'RdPu',
               point_color = 'brown'
               )
    map_server("birds",
             polygons = polys_birds,
             polylines = lines_birds,
             points = points_birds,
             map_base_theme = 'Stamen.Watercolor',
             poly_palette = 'YlGn',
             point_color = '#ffa500',
             polyline_color = "#964b00"
              )
    map_server("amph_rept",
             polygons = polys_amph_rept,
             polylines = NULL,
             points = points_amp_rept,
             map_base_theme = 'Esri.WorldImagery',
             poly_palette = 'Greens',
             point_color = "black"
            )
    map_server("allthegoods",
               polygons = polys_all,
               polylines = NULL,
               points = points_all
              )
}
```

Thank you

See https://github.com/rachel-greenlee/AtlasMaker for more information.