NYC DATA SCIENCE ACADEMY

Shiny Topics

NYC DataScience Academy

Outline

1 **GoogleVis**

2 Leaflet

3 ShinyDashBoard

GoogleVis API

https://developers.google.com/chart/interactive/docs/gallery

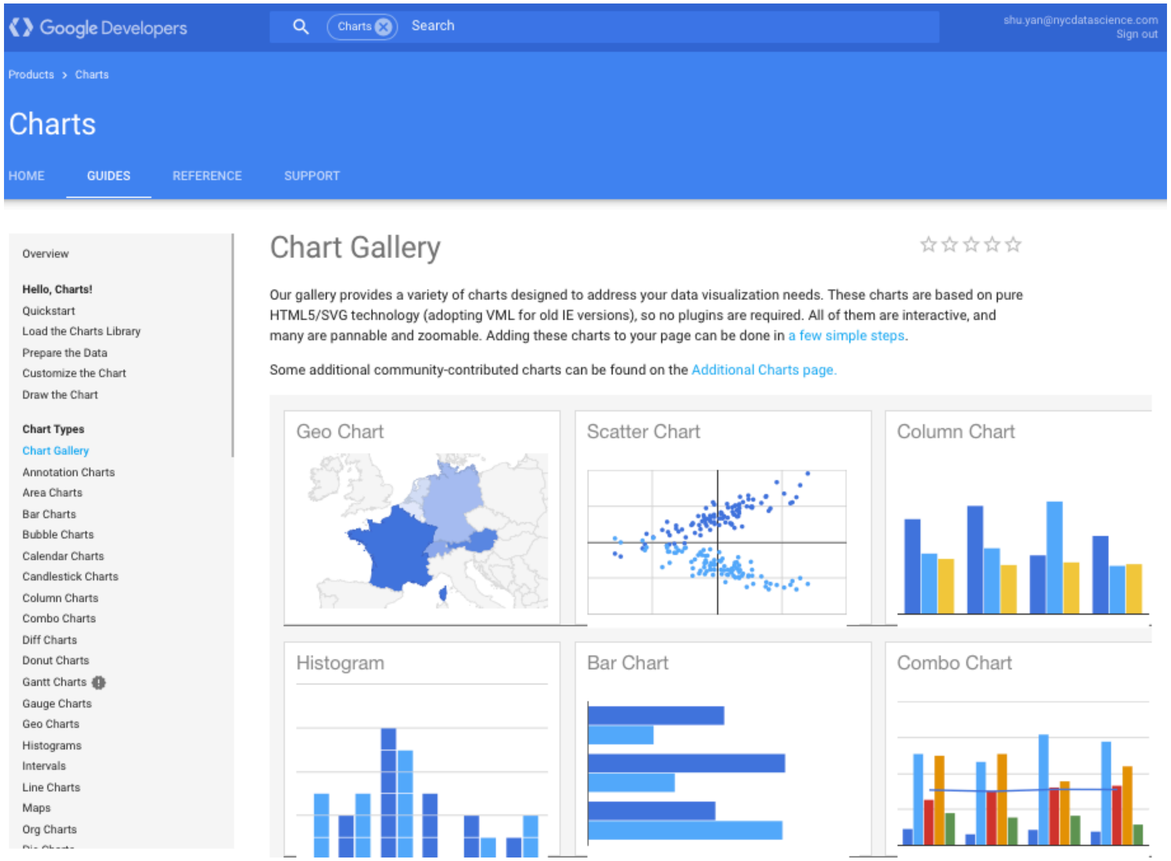


Chart Gallery

Our gallery provides a variety of charts designed to address your data visualization needs. These charts are based on pure HTML5/SVG technology (adopting VML for old IE versions), so no plugins are required. All of them are interactive, and many are pannable and zoomable. Adding these charts to your page can be done in a few simple steps.

Some additional community-contributed charts can be found on the Additional Charts page.

Example

```
M <- gvisMotionChart(Fruits, "Fruit", "Year",
                     options=list(width=600, height=400))
plot(M)
```

Click to use Flash

Charts in googleVis

https://cran.r-project.org/web/packages/googleVis/googleVis.pdf

- Line chart: `gvisLineChart`
- Column chart: `gvisColumnChart`
- Combo chart: `gvisComboChart`
- Scatter chart: `gvisScatterChart`
- Bubble chart: `gvisBubbleChart`
- Geo Chart: `givsGeoChart`
- Table: `gvisTable`

and more…

Library and Demo

```r
## Install the package if you haven't
# install.packages("googleVis")
library(googleVis)
demo(googleVis)
```

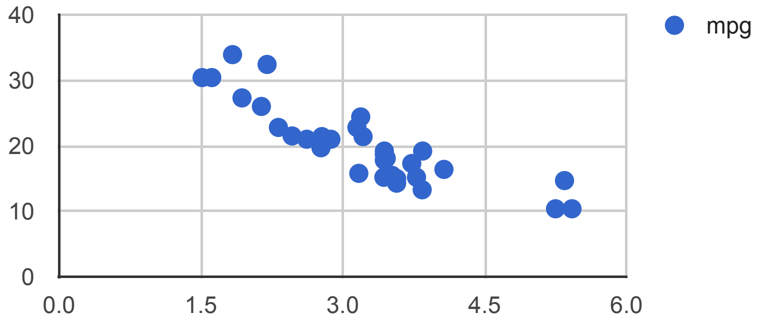
A Simple Example

```r
head(mtcars, n = 10)
```

```
##                    mpg cyl  disp  hp drat    wt  qsec vs am gear carb
## Mazda RX4         21.0   6 160.0 110 3.90 2.620 16.46  0  1    4    4
## Mazda RX4 Wag     21.0   6 160.0 110 3.90 2.875 17.02  0  1    4    4
## Datsun 710        22.8   4 108.0  93 3.85 2.320 18.61  1  1    4    1
## Hornet 4 Drive    21.4   6 258.0 110 3.08 3.215 19.44  1  0    3    1
## Hornet Sportabout 18.7   8 360.0 175 3.15 3.440 17.02  0  0    3    2
## Valiant           18.1   6 225.0 105 2.76 3.460 20.22  1  0    3    1
## Duster 360        14.3   8 360.0 245 3.21 3.570 15.84  0  0    3    4
## Merc 240D         24.4   4 146.7  62 3.69 3.190 20.00  1  0    4    2
## Merc 230          22.8   4 140.8  95 3.92 3.150 22.90  1  0    4    2
## Merc 280          19.2   6 167.6 123 3.92 3.440 18.30  1  0    4    4
```

A Simple Example

```
scatter <- gvisScatterChart(mtcars[,c("wt", "mpg")])
plot(scatter)
```



How it Works

- The R function creates an HTML page
- The HTML page calls Google Charts
- The result is an interactive HTML graphic

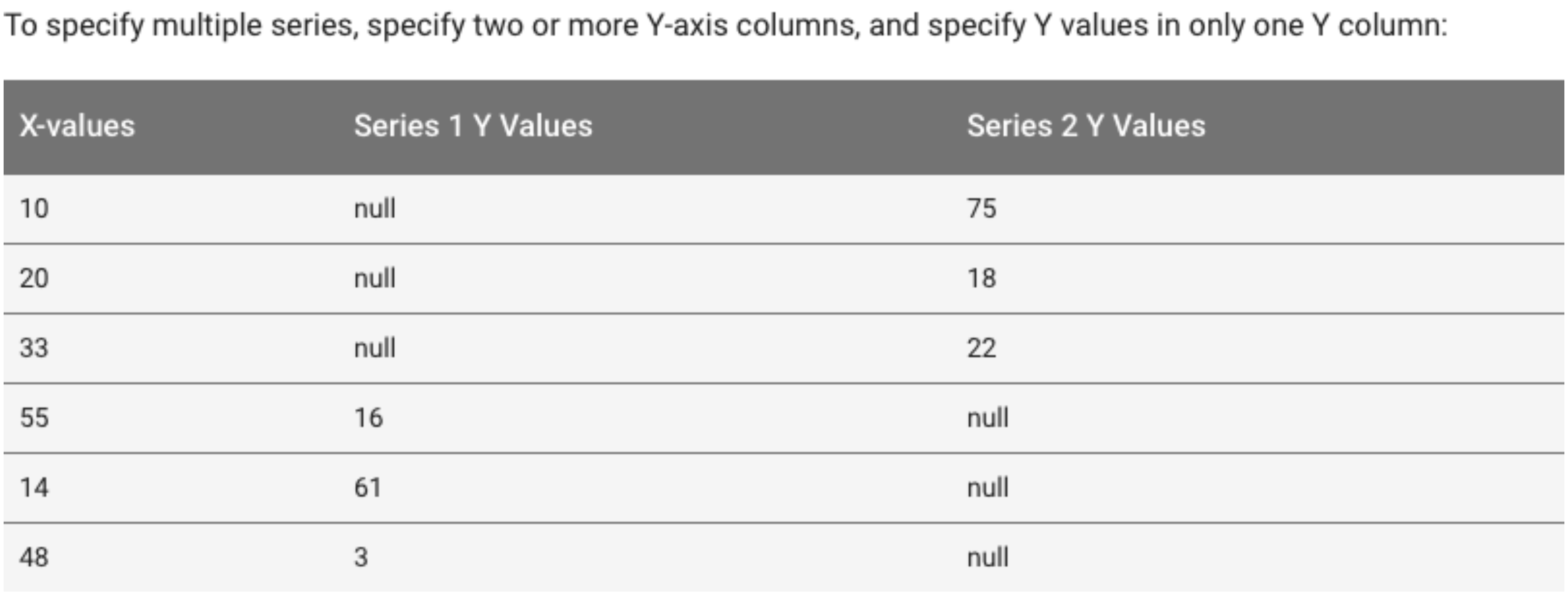
HTML Output

```
print(scatter)
```

```
## <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
##   "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
## <html xmlns="http://www.w3.org/1999/xhtml">
## <head>
## <title>ScatterChartID8efc2c501e</title>
## <meta http-equiv="content-type" content="text/html;charset=utf-8" />
## <style type="text/css">
## body {
##   color: #444444;
##   font-family: Arial,Helvetica,sans-serif;
##   font-size: 75%;
##   }
##   a {
##   color: #4D87C7;
##   text-decoration: none;
## }
```

Data Format

https://developers.google.com/chart/interactive/docs/gallery/scatterchart
format



To specify multiple series, specify two or more Y-axis columns, and specify Y values in only one Y column:

X-values	Series 1 Y Values	Series 2 Y Values
10	null	75
20	null	18
33	null	22
55	16	null
14	61	null
48	3	null

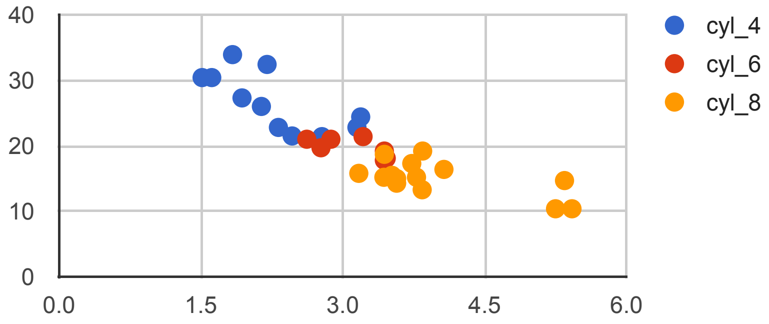
Data Format

```
dt <- mtcars[,c("wt", "mpg")]
dt$cyl_4 <- ifelse(mtcars$cyl==4, dt$mpg, NA)
dt$cyl_6 <- ifelse(mtcars$cyl==6, dt$mpg, NA)
dt$cyl_8 <- ifelse(mtcars$cyl==8, dt$mpg, NA)
dt$mpg <- NULL
head(dt)
```

```
##                      wt cyl_4 cyl_6 cyl_8
## Mazda RX4         2.620    NA  21.0    NA
## Mazda RX4 Wag     2.875    NA  21.0    NA
## Datsun 710        2.320  22.8    NA    NA
## Hornet 4 Drive    3.215    NA  21.4    NA
## Hornet Sportabout 3.440    NA    NA  18.7
## Valiant           3.460    NA  18.1    NA
```

Data Format

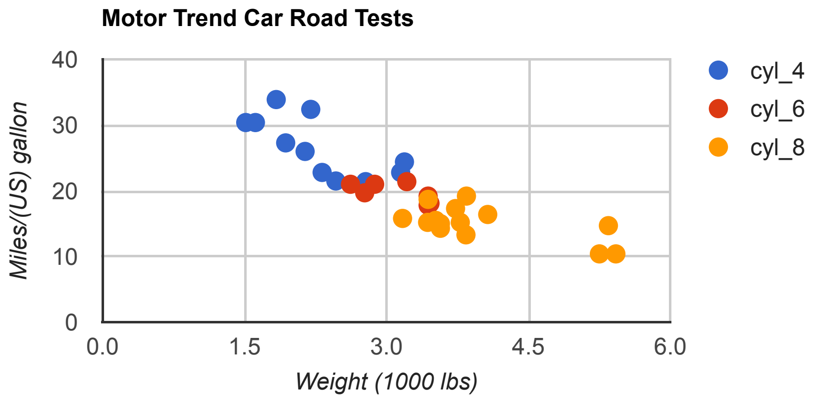
```
scatter <- gvisScatterChart(dt)
plot(scatter)
```



Setting Options

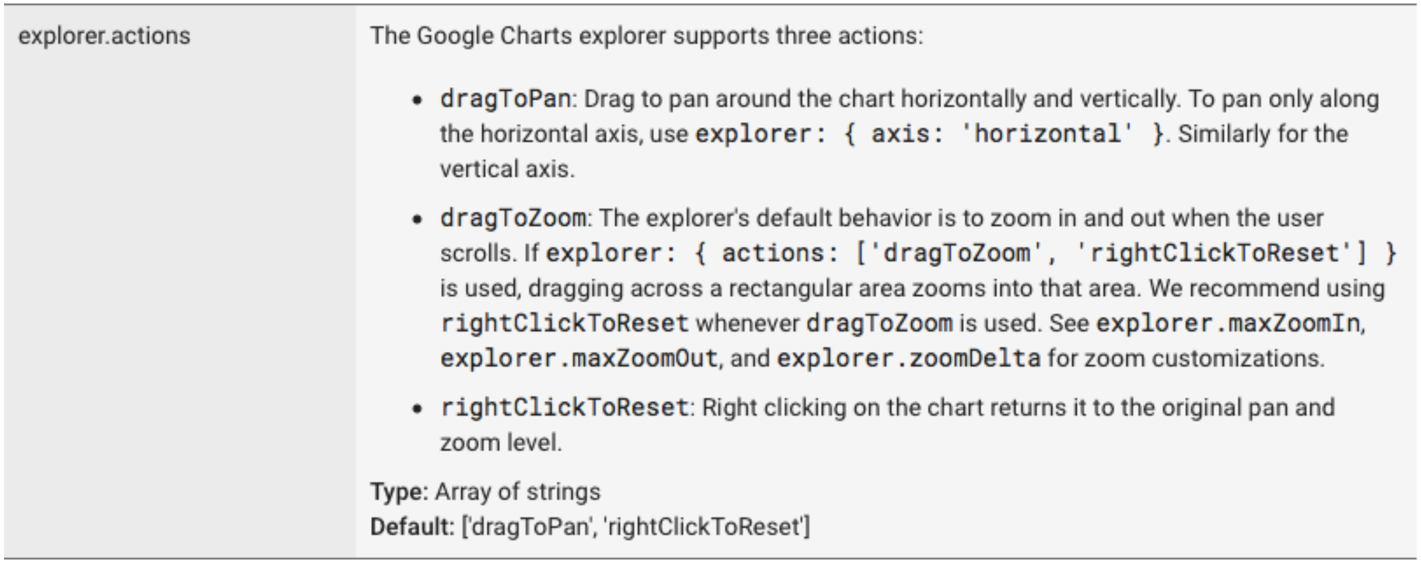
The parameters can be set via a named list.

```
my_options <- list(width="600px", height="300px",
                   title="Motor Trend Car Road Tests",
                   hAxis="{title:'Weight (1000 lbs)'}",
                   vAxis="{title:'Miles/(US) gallon'}")
plot(gvisScatterChart(dt,options=my_options))
```



Setting Options

The parameters have to map those of the Google documentation. For example:



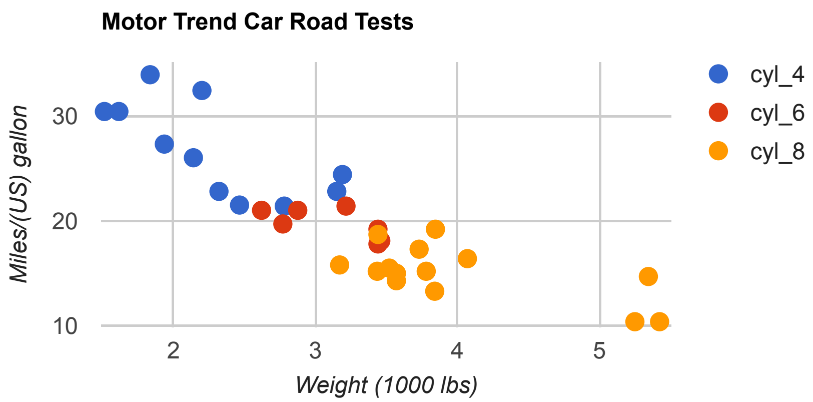
explorer.actions	The Google Charts explorer supports three actions: • `dragToPan`: Drag to pan around the chart horizontally and vertically. To pan only along the horizontal axis, use `explorer: { axis: 'horizontal' }`. Similarly for the vertical axis. • `dragToZoom`: The explorer's default behavior is to zoom in and out when the user scrolls. If `explorer: { actions: ['dragToZoom', 'rightClickToReset'] }` is used, dragging across a rectangular area zooms into that area. We recommend using `rightClickToReset` whenever `dragToZoom` is used. See `explorer.maxZoomIn`, `explorer.maxZoomOut`, and `explorer.zoomDelta` for zoom customizations. • `rightClickToReset`: Right clicking on the chart returns it to the original pan and zoom level. **Type:** Array of strings **Default:** ['dragToPan', 'rightClickToReset']

explorer:{actions:['dragToZoom', 'rightClickToReset']}:

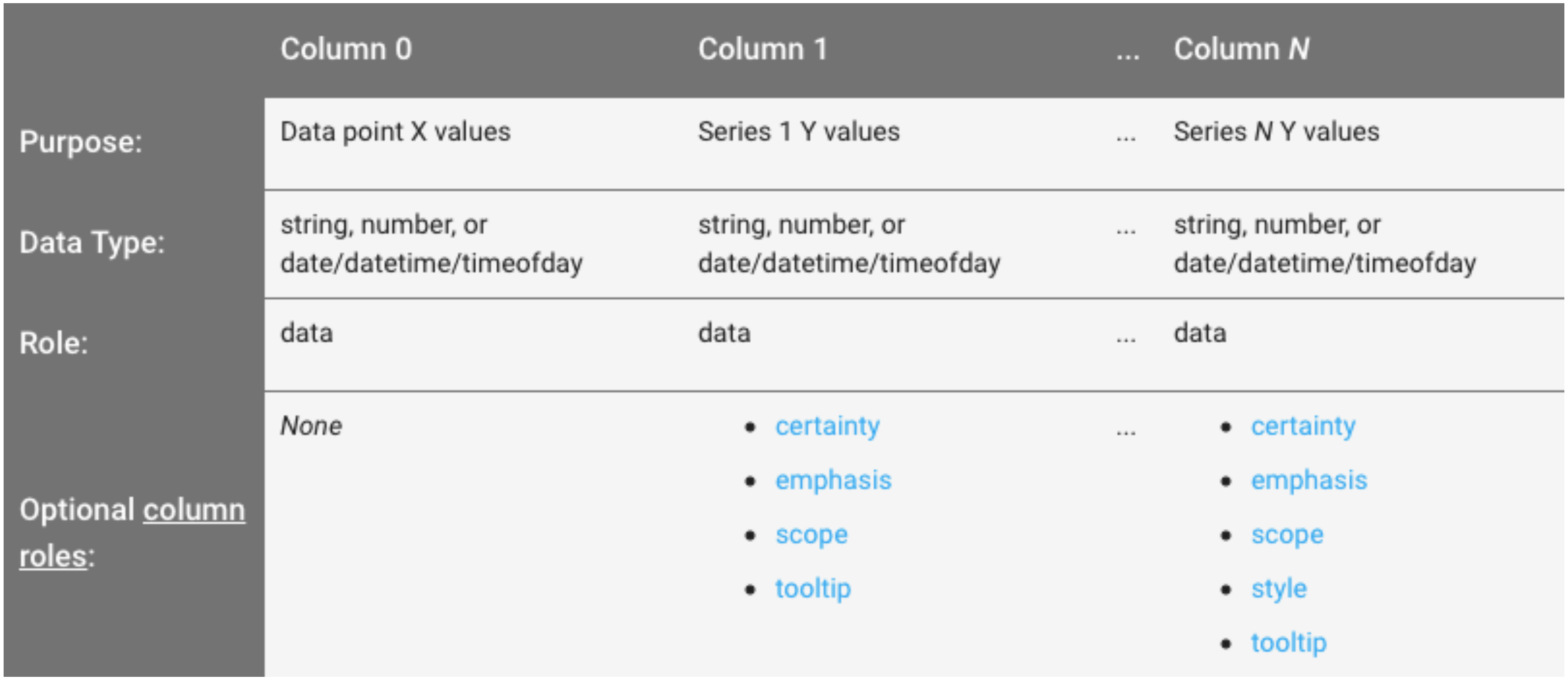
```
explorer="{actions:['dragToZoom', 'rightClickToReset']}"
```

Setting Options

```
my_options$explorer <- "{actions:['dragToZoom', 'rightClickToReset']}"
plot(gvisScatterChart(dt,options=my_options)")
```



Optional Column Roles



	Column 0	Column 1	...	Column *N*
Purpose:	Data point X values	Series 1 Y values	...	Series *N* Y values
Data Type:	string, number, or date/datetime/timeofday	string, number, or date/datetime/timeofday	...	string, number, or date/datetime/timeofday
Role:	data	data	...	data
Optional column roles:	*None*	• certainty • emphasis • scope • tooltip	...	• certainty • emphasis • scope • style • tooltip

https://developers.google.com/chart/interactive/docs/roles#tooltiprole

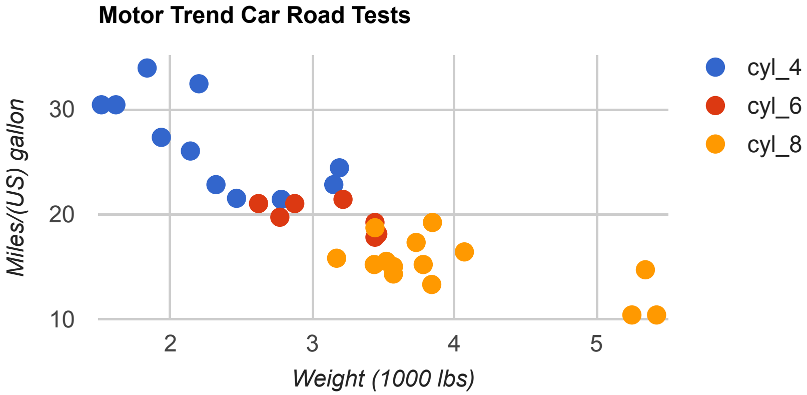
Setting Tooltips

```r
dt <- mtcars[,c("wt", "mpg")]
dt$cyl_4 <- ifelse(mtcars$cyl==4, dt$mpg, NA)
dt$cyl_4.html.tooltip <- rownames(dt)
dt$cyl_6 <- ifelse(mtcars$cyl==6, dt$mpg, NA)
dt$cyl_6.html.tooltip <- rownames(dt)
dt$cyl_8 <- ifelse(mtcars$cyl==8, dt$mpg, NA)
dt$cyl_8.html.tooltip <- rownames(dt)
dt$mpg <- NULL
head(dt)
```

```
##                      wt cyl_4 cyl_4.html.tooltip cyl_6 cyl_6.html.tooltip
## Mazda RX4         2.620    NA          Mazda RX4  21.0          Mazda RX4
## Mazda RX4 Wag     2.875    NA      Mazda RX4 Wag  21.0      Mazda RX4 Wag
## Datsun 710        2.320  22.8         Datsun 710    NA         Datsun 710
## Hornet 4 Drive    3.215    NA     Hornet 4 Drive  21.4     Hornet 4 Drive
## Hornet Sportabout 3.440    NA  Hornet Sportabout    NA  Hornet Sportabout
## Valiant           3.460    NA            Valiant  18.1            Valiant
##                   cyl_8 cyl_8.html.tooltip
```

Setting Tooltips

```
plot(gvisScatterChart(dt,options=my_options))
```



Outline

1 GoogleVis

2 **Leaflet**

3 ShinyDashBoard

Introduction to Leaflet

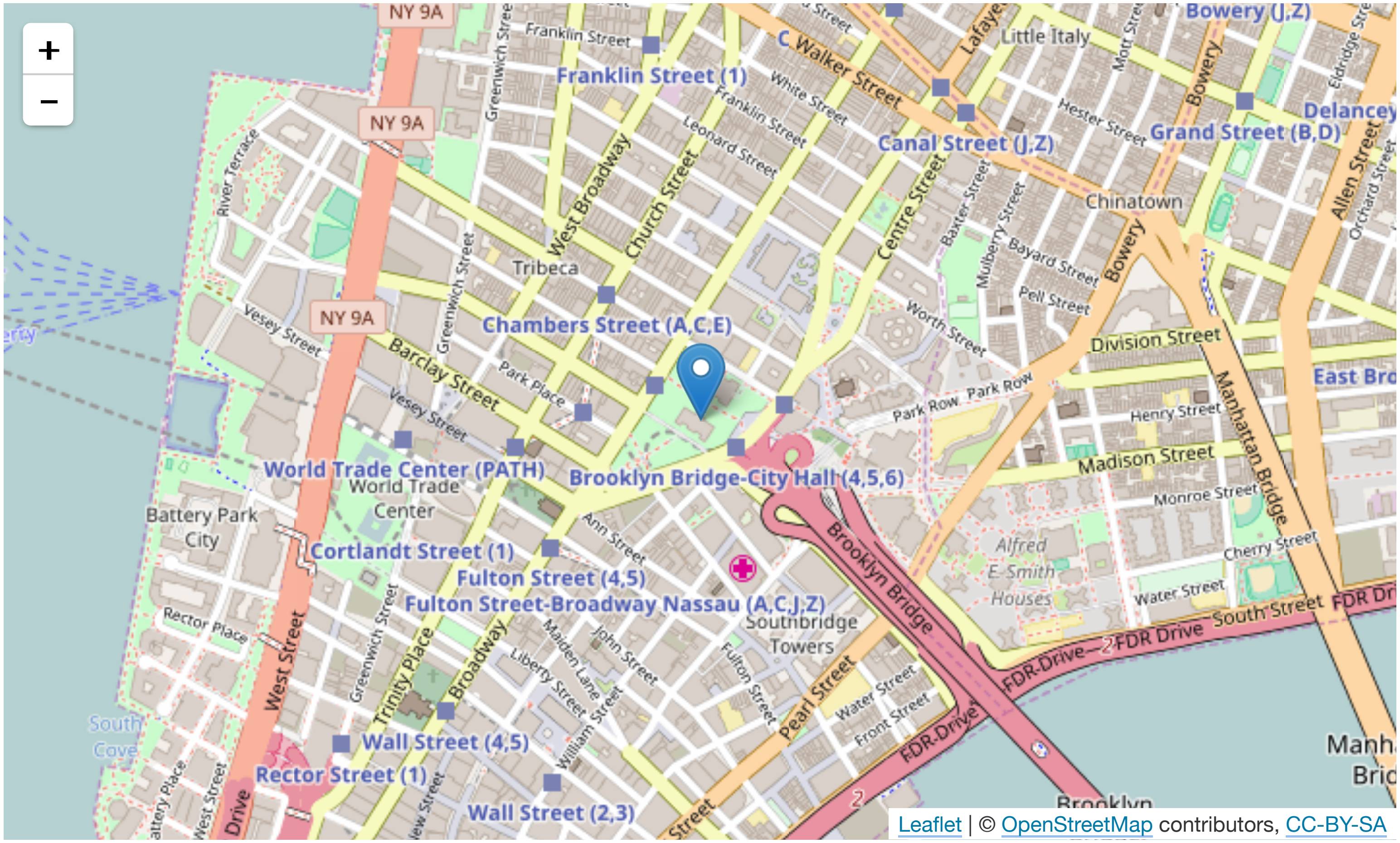
- Leaflet is one of the most popular open-source JavaScript libraries for interactive maps.
- Leaflet R package makes it easy to integrate and control Leaflet maps in R.

To use leaflet is as simple as to use many other R packages

```
# You need to use the development version for some
# of the advanced features in leaflet.
# To install the development version from Github, run
devtools::install_github("rstudio/leaflet")
library(leaflet)
```

A Quick Example

```r
leaflet() %>% addTiles() %>%  # Add default OpenStreetMap map tiles
  addMarkers(lng=-74.0059, lat=40.7128, popup="New York City")
```



Adding Data

There're several ways to visualize data with Leaflet maps:

- addMarkers()
- addCircleMarkers()
- addPopups()
- addPolylines()
- addPolygons()
- addCircles()
- addRectangles()
- addTopoJSON()
- addGeoJSON()

Visualizing Hurricane Andrew Path

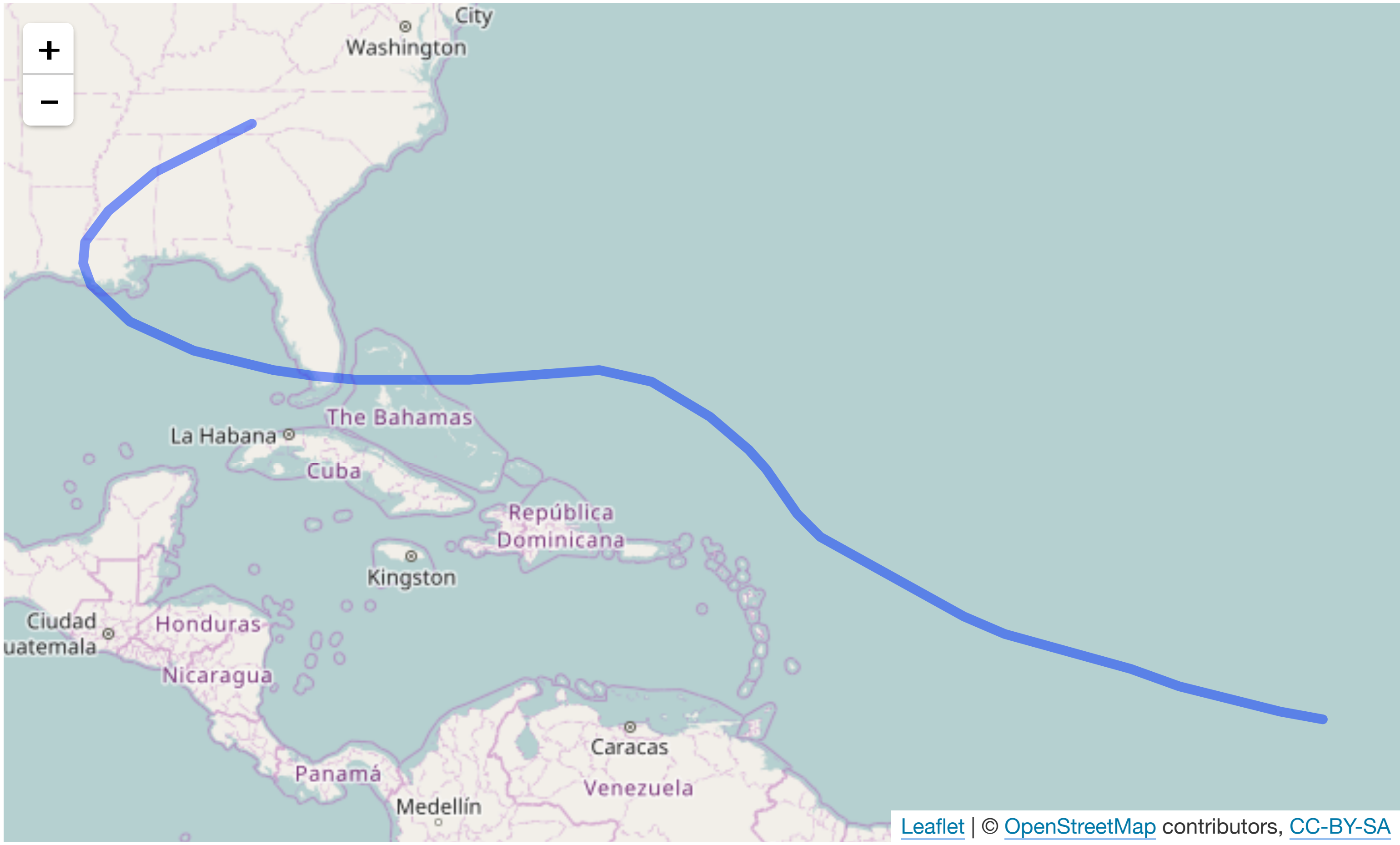
The Andrew dataset (built in dataset in `googleVis` library) includes hurricane Andrew storm path from 16 August to 28 August 1992.

Let's visualize the path using `addPolylines()`

- Pass `Long`/`Lat` columns of Andrew dataset as the first two variables

```
leaflet_andrew <- leaflet(Andrew) %>%
  addTiles() %>%
  addPolylines(~Long, ~Lat)
leaflet_andrew
```

Visualizing Hurricane Andrew Path



Adding Polygons

There were 6 states that were affected by the hurricane along the path:

Florida, *Louisiana*, *Mississippi*, *Alabama*, *Georgia*, and *Tennesse*.

Now let's color them using polygons.

Adding Polygons

We first create a `map` object that contains the geoshapes of the 6 states

Let's create such an object using the `map()` function from the `maps` package

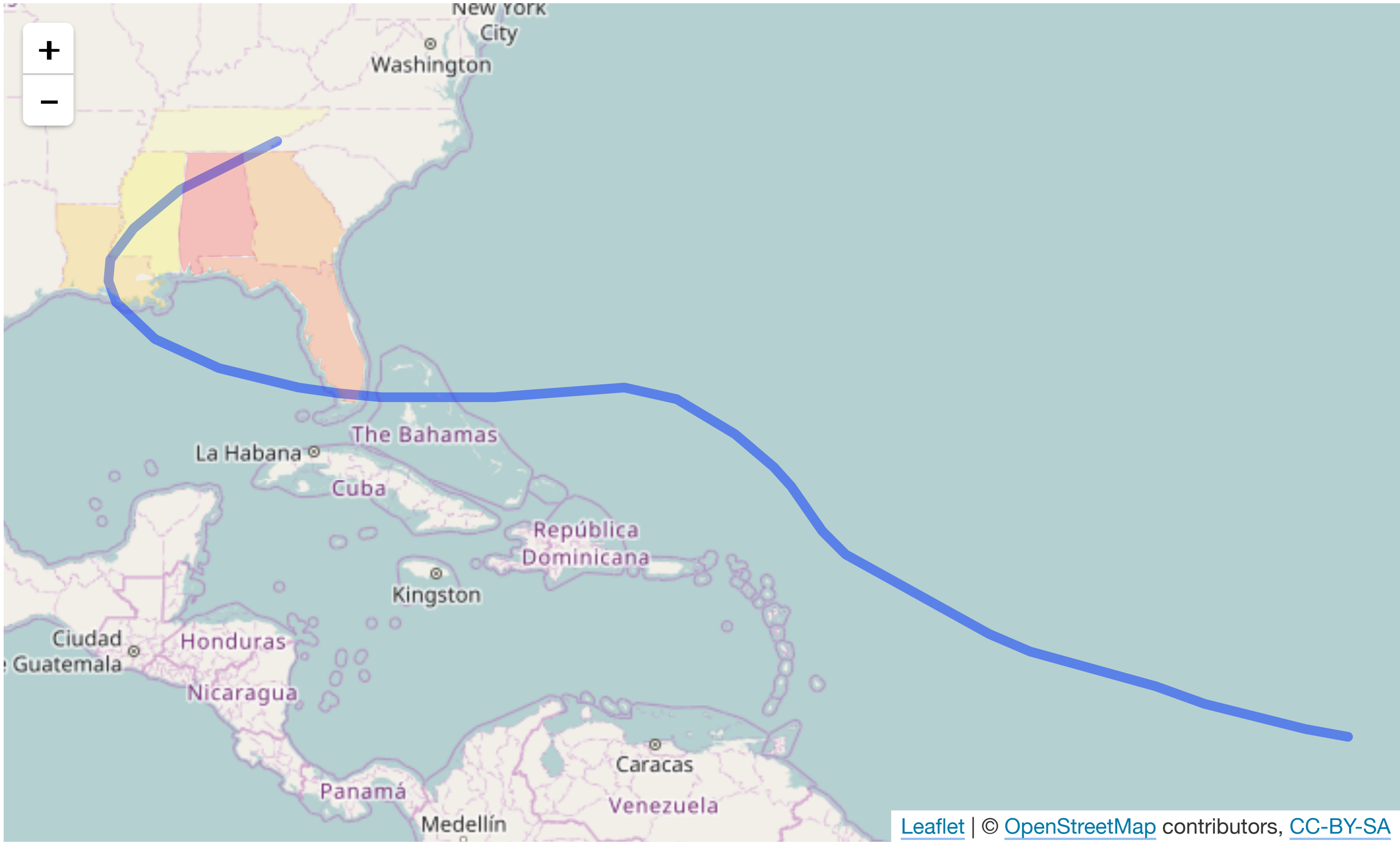
```r
colStates <- map("state", fill = TRUE,
                 plot = FALSE,
                 region = c("florida", "louisiana", "mississippi",
                            "alabama", "georgia", "tennesse"))
```

Adding Polygons

Next we create another layer on top of the leaflet map by adding polygons using the `map` object we just created.

```
leaflet_andrew <- leaflet_andrew %>%
  addPolygons(data=colStates,
              fillColor = heat.colors(6, alpha = 1),
              stroke = FALSE)
leaflet_andrew
```

Adding Polygons



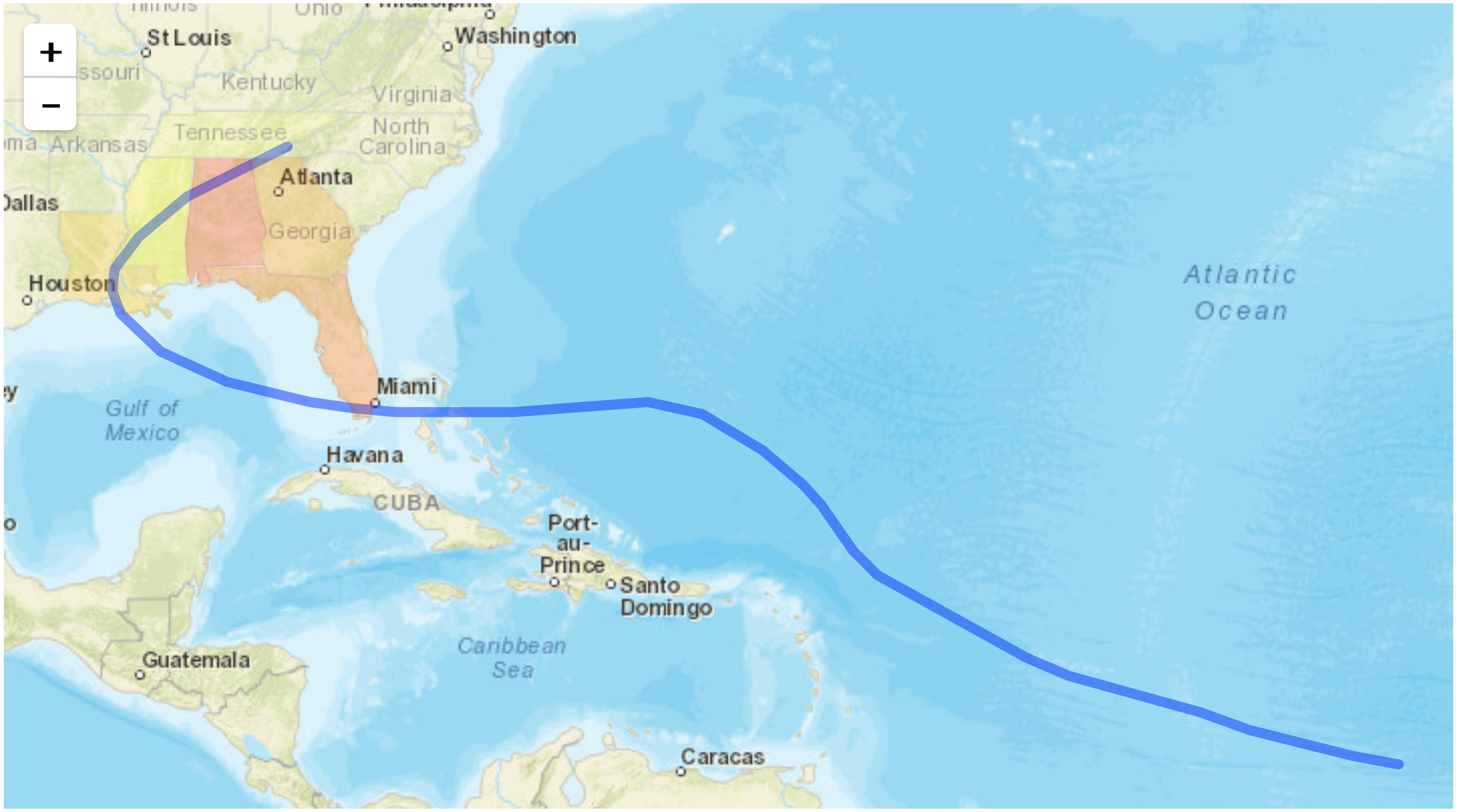
Changing Tiles

One of the fascinating things about the leaflet package is the variety of available tiles, which can be added using the `addProviderTiles()` function.

Let's change the tile to `Esri.WorldStreetMap`

```
leaflet_andrew <- leaflet_andrew %>%
  addProviderTiles("Esri.WorldStreetMap")
leaflet_andrew
```

Changing Tiles



Leaflet | © OpenStreetMap contributors, CC-BY-SA, Tiles © Esri — Source: Esri, DeLorme, NAVTEQ, USGS, Intermap, iPC, NRCAN, Esri Japan, METI, Esri China (Hong Kong), Esri (Thailand), TomTom, 2012

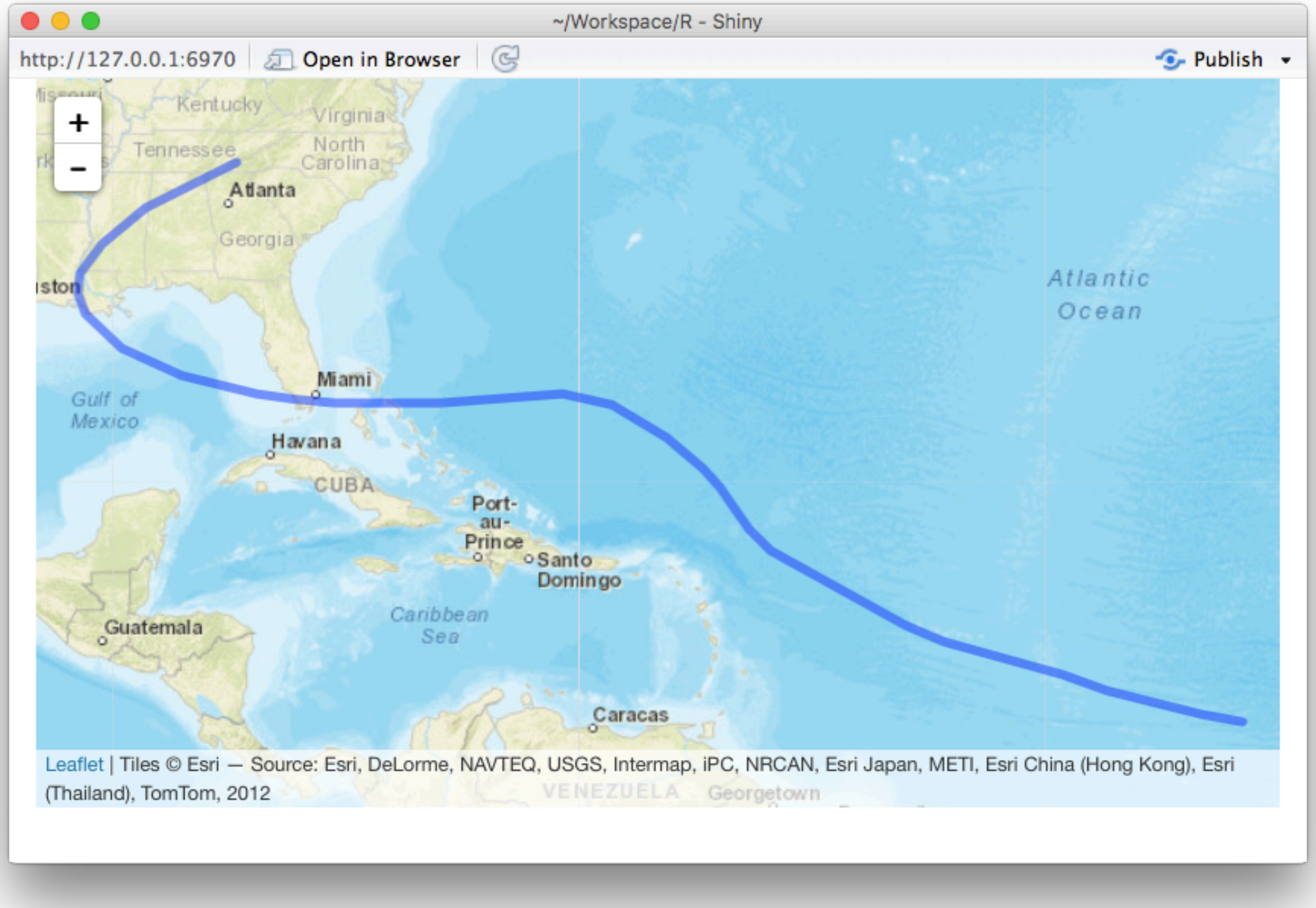
Using Leaflet with Shiny

Making Leaflet maps in Shiny is similiar to other output widgets:

- UI -> `leafletOutput`
- server -> `renderLeaflet`

```
ui <- fluidPage(
  leafletOutput("mymap")
)
server <- function(input, output, session) {
  output$mymap <- renderLeaflet({
    leaflet(Andrew) %>%
      addProviderTiles("Esri.WorldStreetMap") %>%
      addPolylines(~Long, ~Lat)
    })
}
shinyApp(ui, server)
```

Using Leaflet with Shiny



Leaflet | Tiles © Esri — Source: Esri, DeLorme, NAVTEQ, USGS, Intermap, iPC, NRCAN, Esri Japan, METI, Esri China (Hong Kong), Esri (Thailand), TomTom, 2012

Modifying Maps with `leafletProxy`

- Reactive inputs and expressions that affect the renderLeaflet expression will cause the entire map to be redrawn from scratch.
    - All of the settings will be reset
    - Every single layer will be recomputed
- To modify a map that's already running in the page, use the `leafletProxy()` function in place of the `leaflet()` call

Modifying Maps with `leafletProxy`

Assume we want to provide an option to draw state polygons on our shiny app:

- use `addPolygons` when the checkbox is checked,
- use `removeShape` when the checkbox is unchecked.

Modifying Existing Maps - UI

Let's add a `checkboxInput` to UI first:

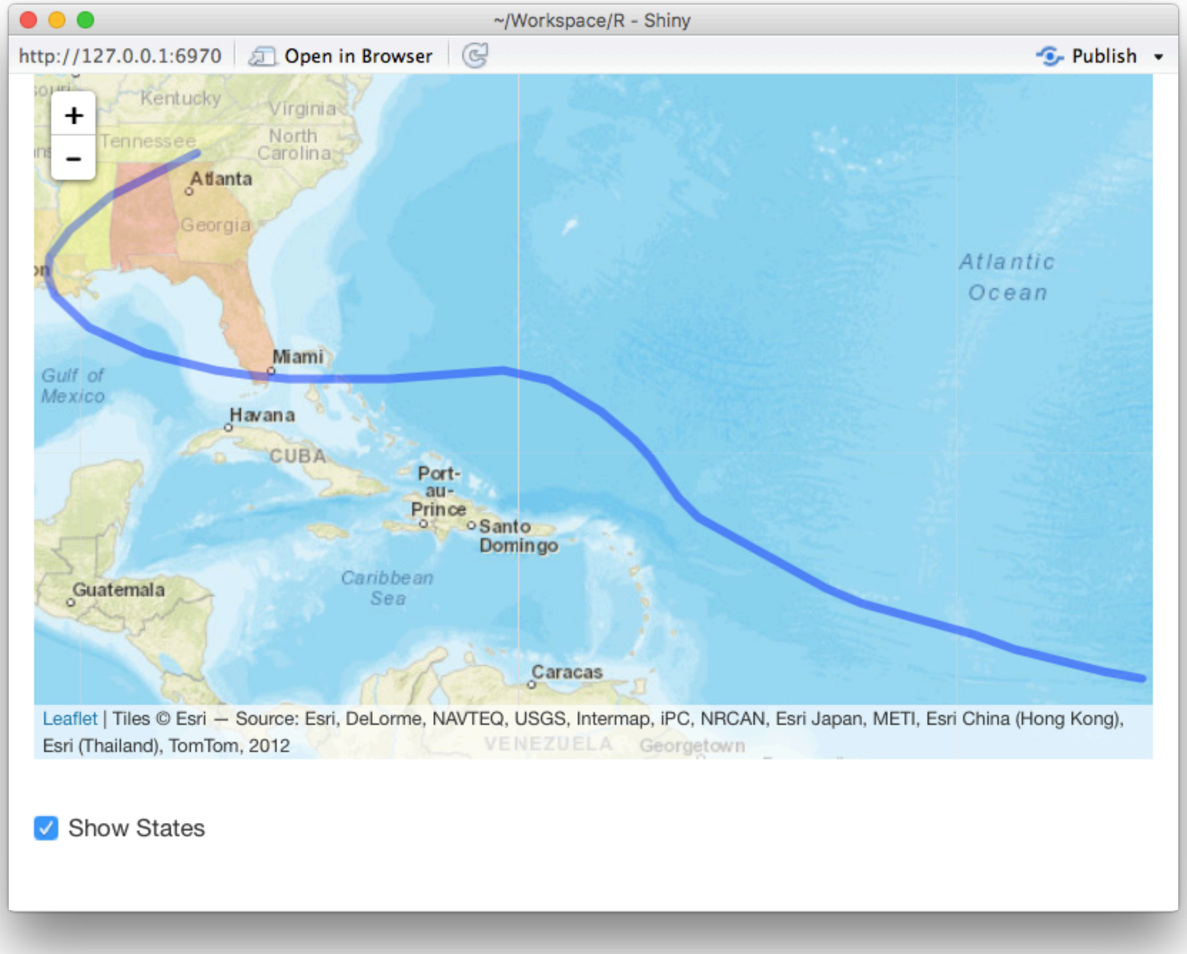
```
ui <- fluidPage(
  leafletOutput("mymap"),
  br(),
  checkboxInput("show", "Show States", value = FALSE)
)
```

The server side is a little complicated - we need to add another fucntion called `observeEvent` to make response.

Modifying Existing Maps - server

```r
colStates <- map("state", fill = TRUE, plot = FALSE,
                 region = c("florida", "louisiana", "mississippi",
                            "alabama", "georgia", "tennesse"))
server <- function(input, output, session) {
  ...
  observeEvent(input$show, {
    proxy <- leafletProxy("mymap")
    if(input$show) {
      proxy %>% addPolygons(data=colStates, stroke = FALSE,
                            fillColor = heat.colors(6, alpha = 1),
                            layerId = LETTERS[1:6])
    } else {
      proxy %>% removeShape(layerId = LETTERS[1:6])
    }
  })
}
```

Modifying Existing Maps



Leaflet | Tiles © Esri — Source: Esri, DeLorme, NAVTEQ, USGS, Intermap, iPC, NRCAN, Esri Japan, METI, Esri China (Hong Kong), Esri (Thailand), TomTom, 2012

☑ Show States

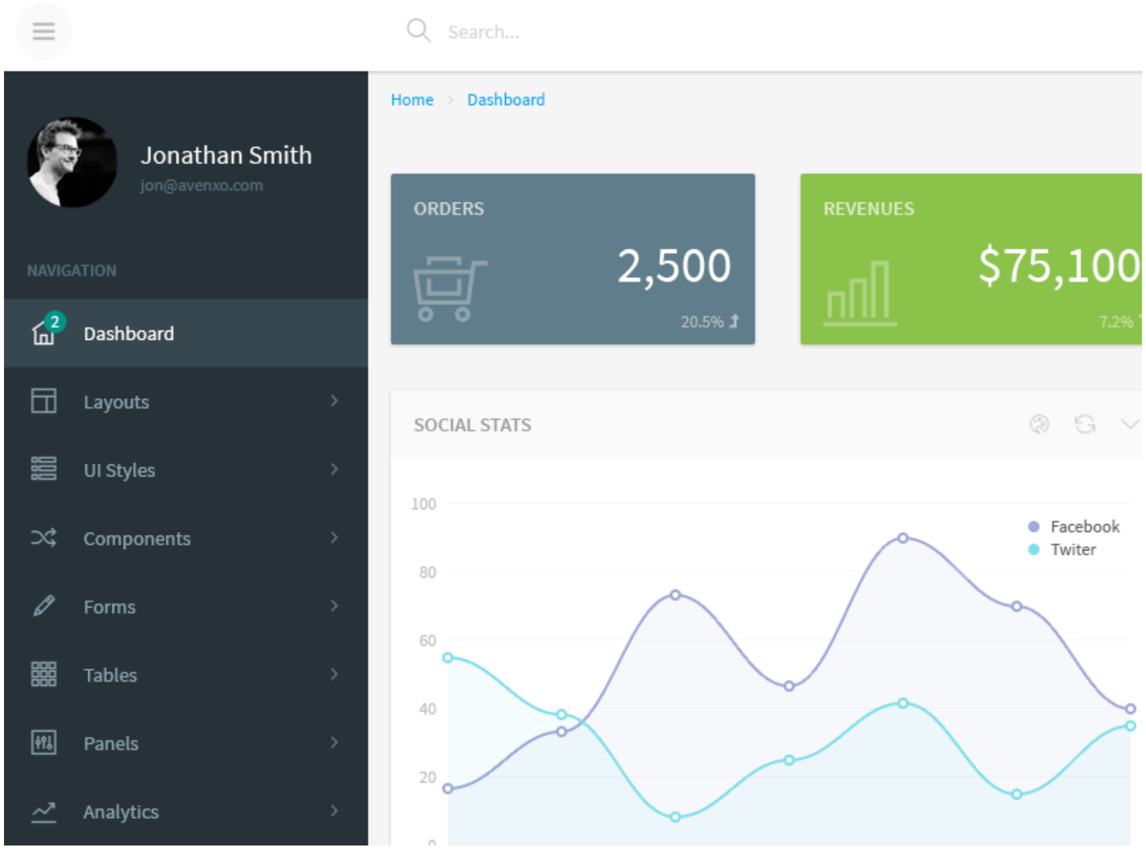
Outline

1 GoogleVis

2 Leaflet

3 **ShinyDashBoard**

Dashboard UI Design



shinydashboard Installation

```
install.packages("shinydashboard")
```

See the documentation at http://rstudio.github.io/shinydashboard/ for more information

shinydashboard Layout

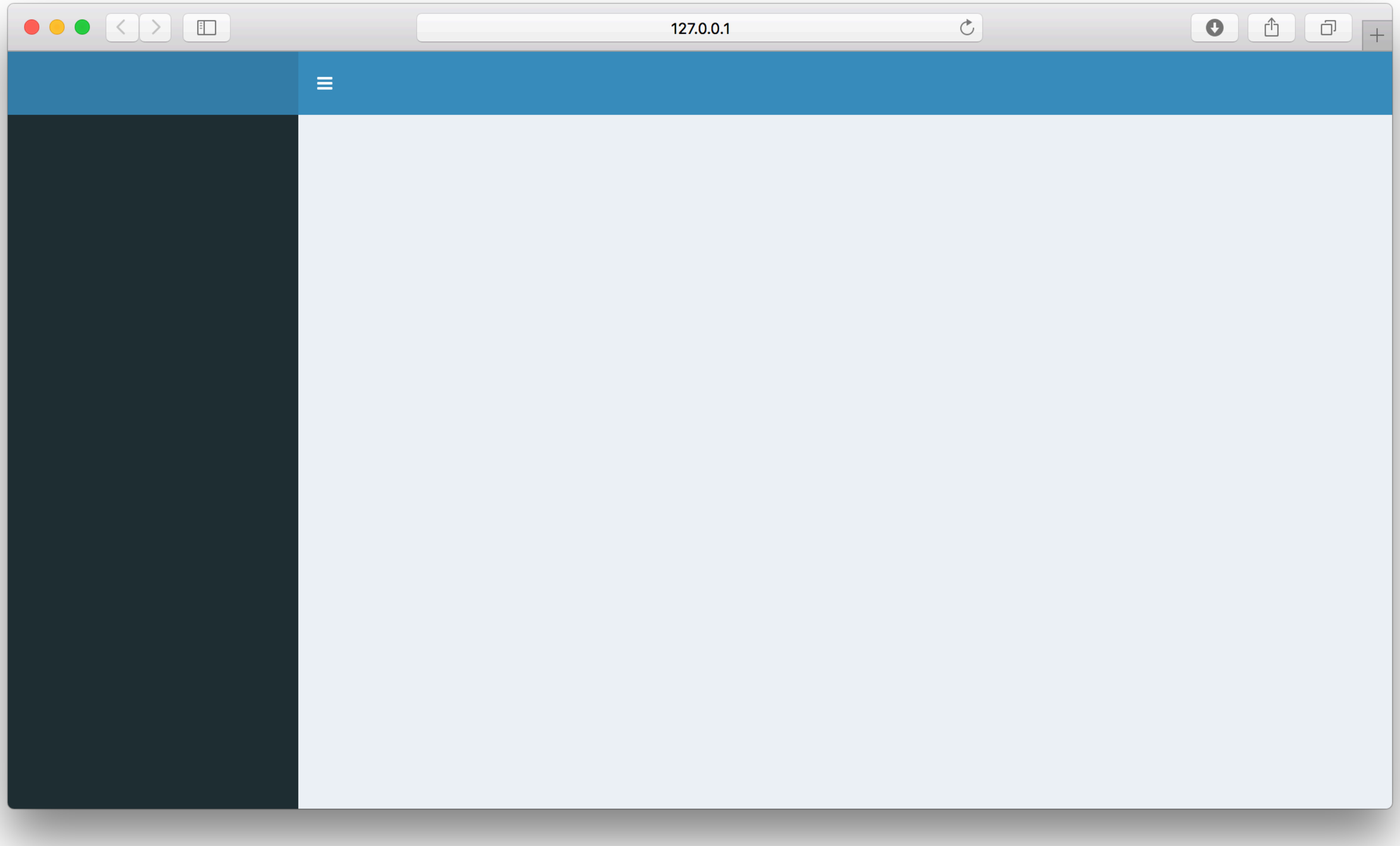
A dashboard has three parts: a header, a sidebar, and a body.

```r
## ui.R ##
library(shinydashboard)

shinyUI(dashboardPage(
  dashboardHeader(),
  dashboardSidebar(),
  dashboardBody()
))


## server.R ##
shinyServer(function(input, output){
})
```

shinydashboard Layout



Header

Setting the title is simple, just use the `title` argument in `dashboardHeader`:

```
## ui.R ##
dashboardHeader(title = "My Dashboard")
```

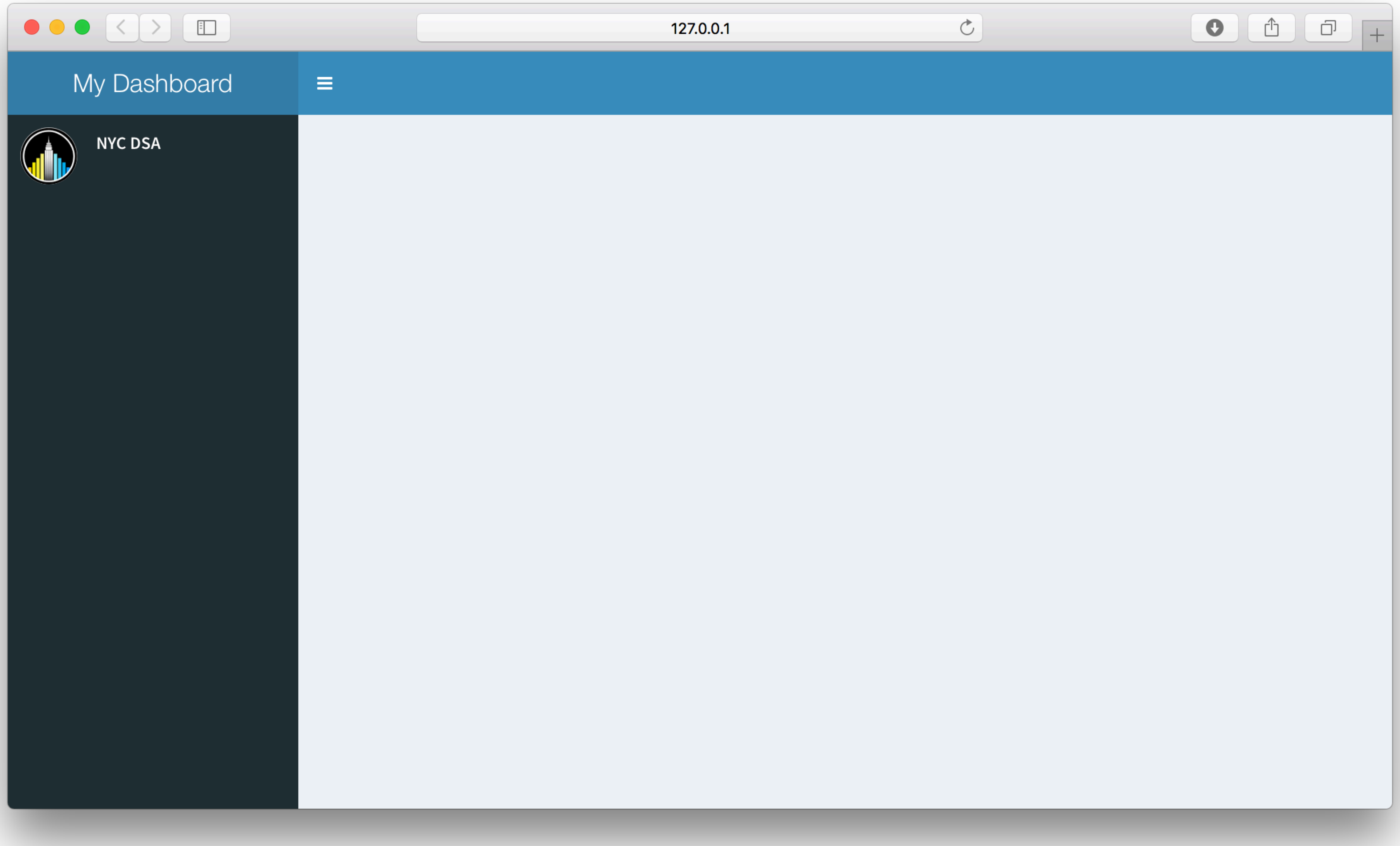
(Note: Besides header, we can also set three types of menus – messages, notifications, and tasks - in header)

Adding Personal Information in Sidebar

We can also add personal info easily with `siderbarUserPanel` inside `dashboardSidebar`:

```
## ui.R ##
dashboardSidebar(
  sidebarUserPanel("Your Name",
                   image = <link to Your Photo>)
)
```

Adding Header and Personal Info



Dataset - state.x77

Matrix with 50 rows and 8 columns giving the following statistics in the respective columns.

- **Population**: population estimate as of July 1, 1975
- **Income**: per capita income (1974)
- **Illiteracy**: illiteracy (1970, percent of population)
- **Life Exp**: life expectancy in years (1969–71)
- **Murder**: murder and non-negligent manslaughter rate per 100,000 population (1976)
- **HS Grad**: percent high-school graduates (1970)
- **Frost**: mean number of days with minimum temperature below freezing (1931–1960) in capital or large city
- **Area**: land area in square miles

Dataset

```
##            Population Income Illiteracy Life Exp Murder HS Grad Frost
## Alabama          3615   3624        2.1    69.05   15.1    41.3    20
## Alaska            365   6315        1.5    69.31   11.3    66.7   152
## Arizona          2212   4530        1.8    70.55    7.8    58.1    15
## Arkansas         2110   3378        1.9    70.66   10.1    39.9    65
## California      21198   5114        1.1    71.71   10.3    62.6    20
## Colorado         2541   4884        0.7    72.06    6.8    63.9   166
##              Area
## Alabama     50708
## Alaska     566432
## Arizona    113417
## Arkansas    51945
## California 156361
## Colorado   103766
```

Using global.R (Data preparation)

Objects defined in **global.R** are loaded into the global environment and are available to both **server.R** and **ui.R**.

```
## global.R ##

# convert matrix to dataframe
state_stat <- data.frame(state.name = rownames(state.x77), state.x77)
# remove row names
rownames(state_stat) <- NULL
# create variable with colnames as choice
choice <- colnames(state_stat)[-1]
```

Goals

We want to:

- allow user to select differet columns using `selectizeInput()`;
- visualize the column using `gvisGeoChart()` and `gvisHistogram()`;
- display full dataset using `DT` library and highlight the selected column.

Sidebar menu items and tabs

Hmmm… too much for one page, let's split them into two tabs.

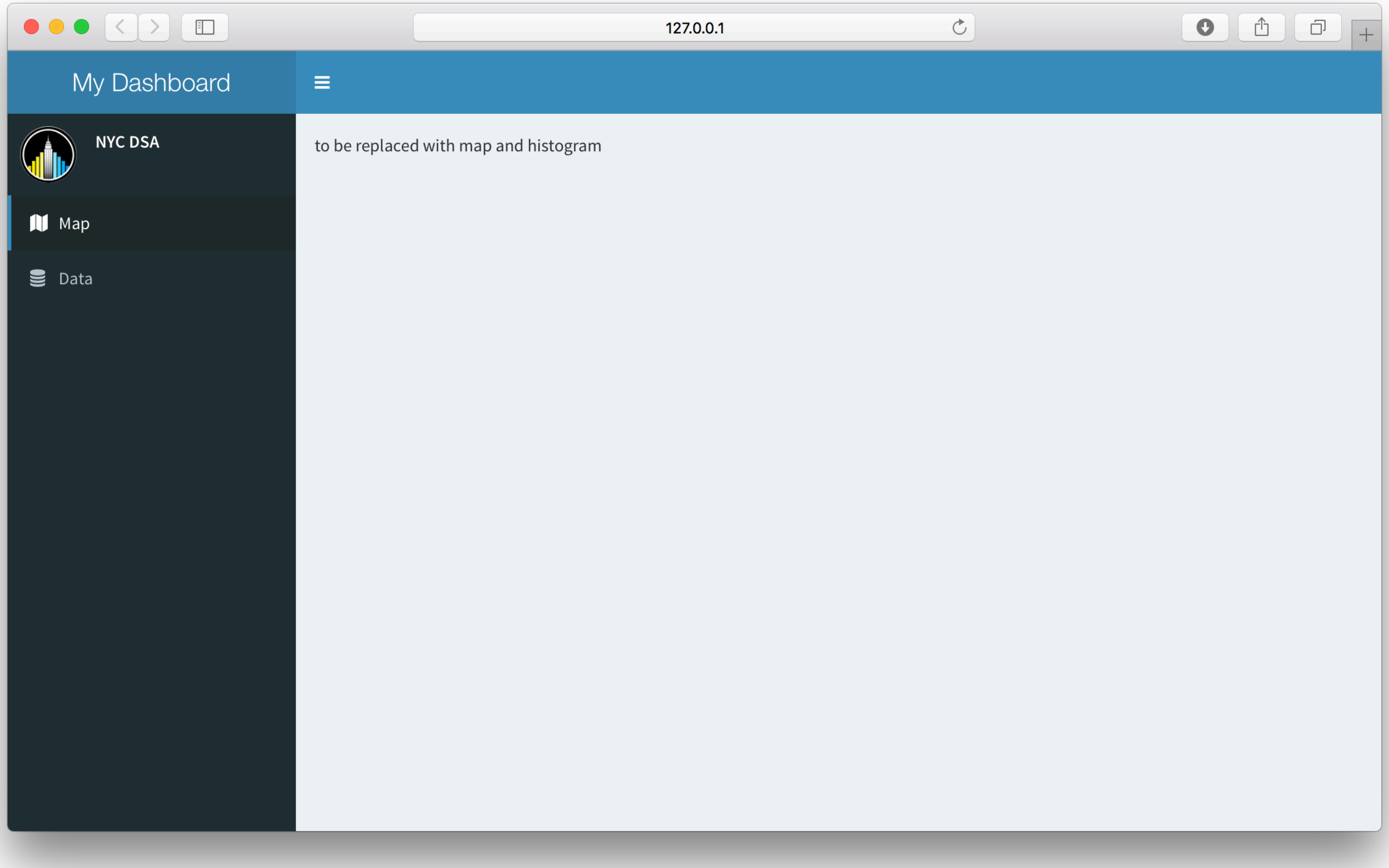
- Similar to `tabPanels` from Shiny, in `shinydashboard` we can create `menuItem` and `tabItem`.
- To match up a `menuItem` with a `tabItem`, make sure that they have matching values for `tabName`.
- When users click on one of the `menuItems` in the sideBar, it will display different content (`tabItem`) in the body of the dashboard.

Sidebar menu items and tabs

The menu items are put in `sidebarMenu()` as follows:

```r
## ui.R ##
dashboardSidebar(
    sidebarUserPanel("Your Name", image = <link to Your Photo>),
    sidebarMenu(
        menuItem("Map", tabName = "map", icon = icon("map")),
        menuItem("Data", tabName = "data", icon = icon("database")))
)
dashboardBody(
    tabItems(
        tabItem(tabName = "map",
                "to be replaced with map and histogram"),
        tabItem(tabName = "data",
                "to be replaced with datatable"))
)
```

Sidebar menu items and tabs



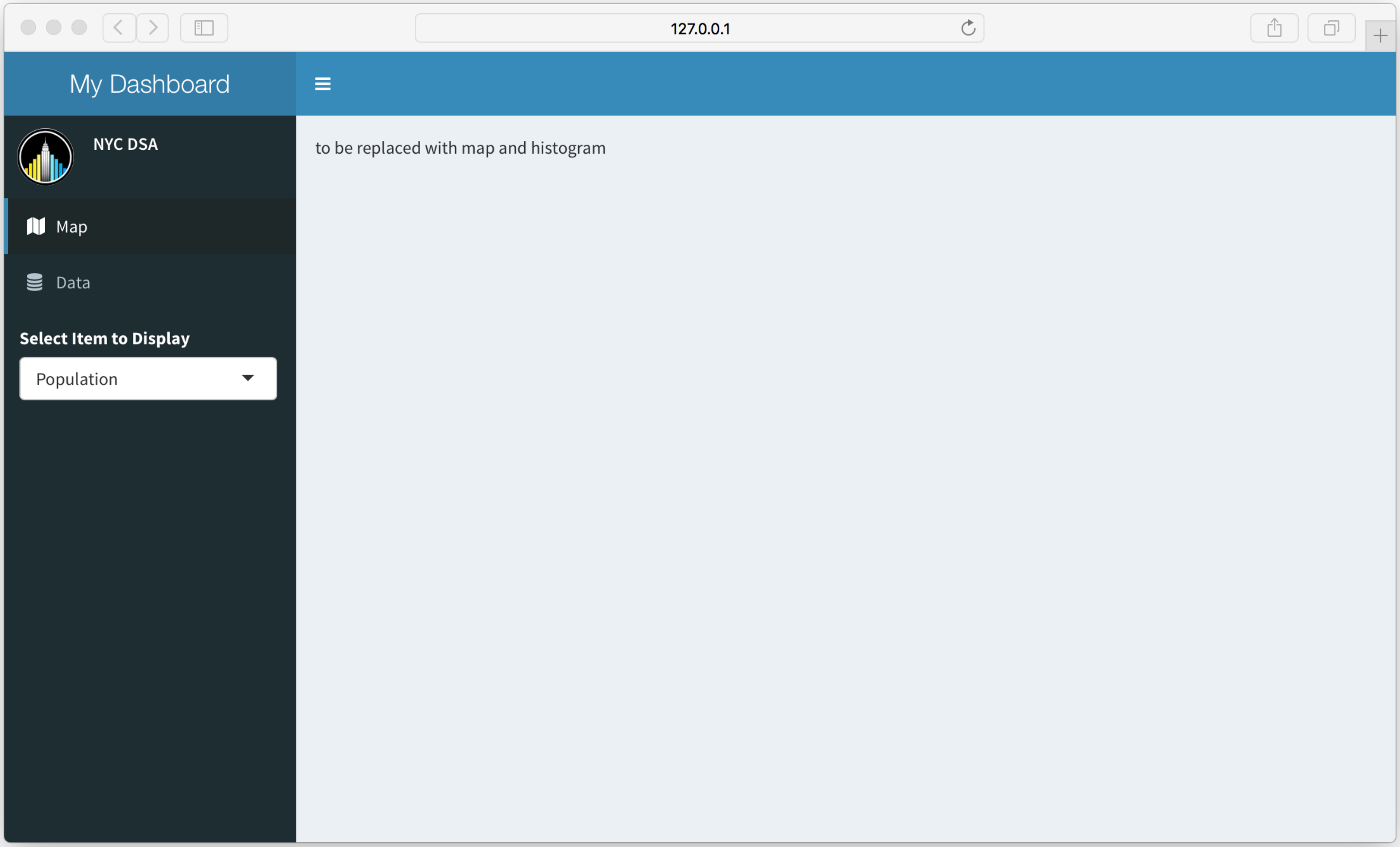
to be replaced with map and histogram

Adding Input Widget

Since we want user to be able to choose which column to visualize, we can insert a `selectizeInput()` inside `dashboardSidebar()`.

```
## ui.R ##
dashboardSidebar(
    ... ...
    selectizeInput("selected",
                   "Select Item to Display",
                   choice)
)
```

Adding Input Widget



Select Item to Display

Population

to be replaced with map and histogram

Building Reactive Outputs

Time to give your Shiny app a “live” quality!

The object will be reactive if the code calls `input$selected`.

Remeber we want to:

- build a map using `gvisGeoChart()`
  - `renderGvis()` -> `htmlOutput()`
- build a histogram using `gvisHistogram()`
  - `renderGvis()` -> `htmlOutput()`
- display full dataset using `datatable()`
  - `DT::renderDataTable()` -> `DT::dataTableOutput()`

Installing DT package

The R package DT provides an R interface to the JavaScript library DataTables.

R data objects (matrices or data frames) can be displayed as tables on HTML pages, and DataTables provides filtering, pagination, sorting, and many other features in the tables.

```r
# You need to use the development version for some
# of the advanced features in DT
# To install the development version from Github, run
devtools::install_github('rstudio/DT')
```

Building Reactive Outputs in `server.R`

Now let's build the render part in `server.R`.

- map and histogram

```
## server.R ##
# show map using googleVis
output$map <- renderGvis({
    gvisGeoChart(state_stat, "state.name", input$selected,
                 options=list(region="US", displayMode="regions",
                              resolution="provinces",
                              width="auto", height="auto"))
                # using width="auto" and height="auto" to
                # automatically adjust the map size
})
# show histogram using googleVis
output$hist <- renderGvis(
    gvisHistogram(state_stat[,input$selected, drop=FALSE]))
```

Building Reactive Outputs in **`server.R`**

- datatable

```r
## server.R ##
# show data using DataTable
output$table <- DT::renderDataTable({
    datatable(state_stat, rownames=FALSE) %>%
        formatStyle(input$selected,
                    background="skyblue", fontWeight='bold')
        # Highlight selected column using formatStyle
})
```

The datatable documentation can be found via: https://rstudio.github.io/DT/

Adding Reactive Outputs to Boxes in `ui.R`

Boxes are the main building blocks of dashboard pages.

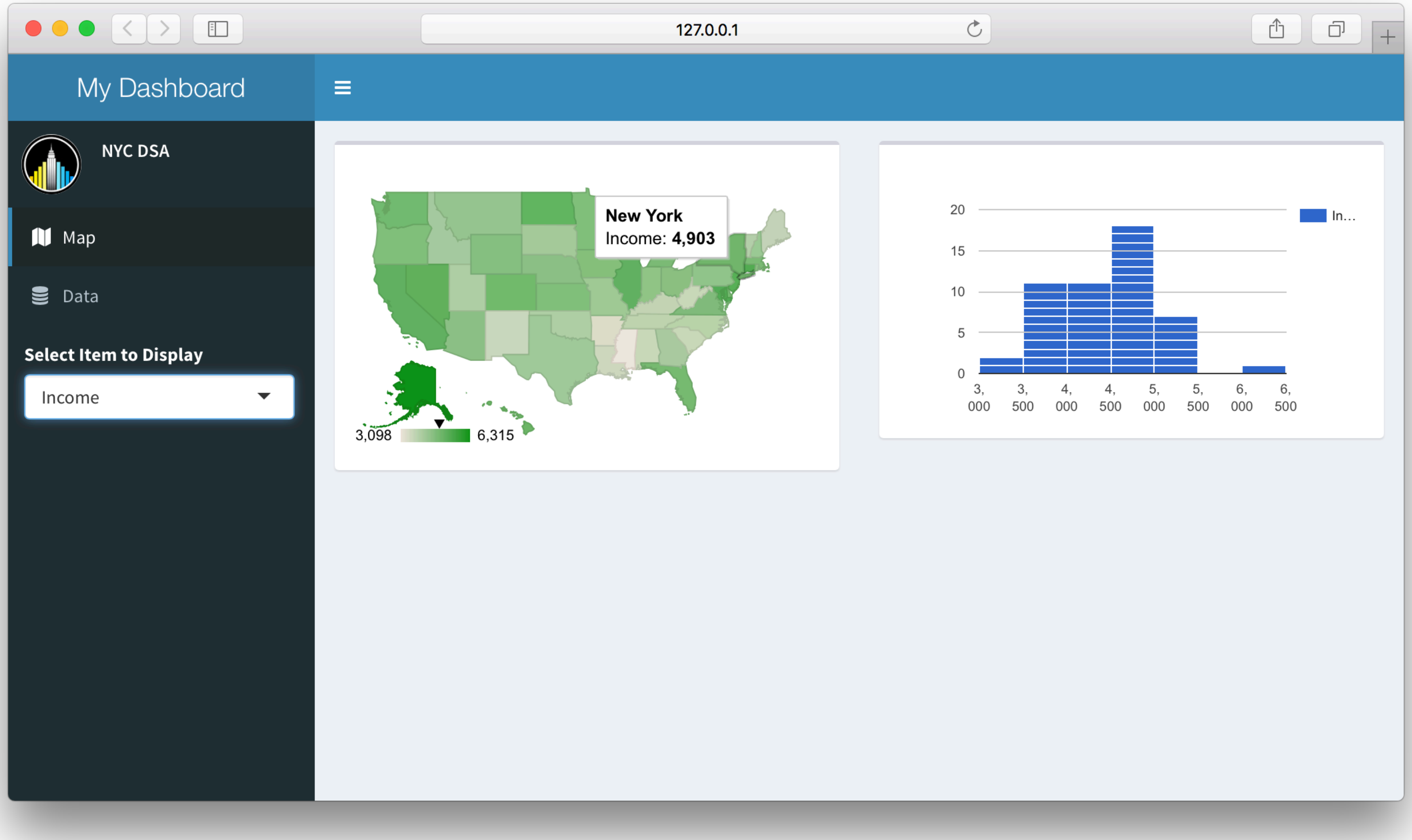
A basic box can be created with the `box()` function, and the contents of the box can be (most) any Shiny UI content.

In a typical dashboard, these boxes would be placed inside a `fluidRow()`.

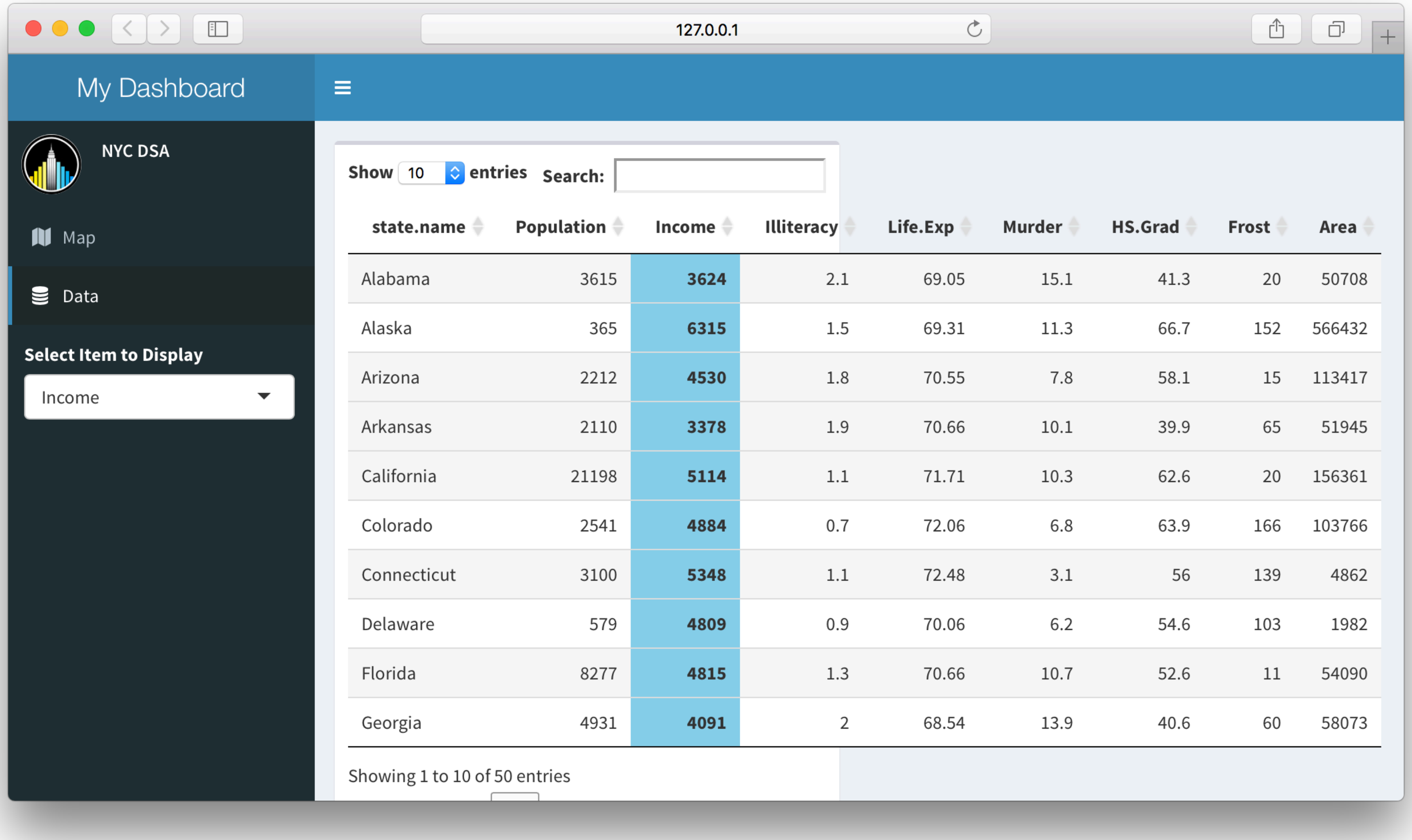
Adding Reactive Outputs to Boxes in `ui.R`

```r
## ui.R ##
dashboardBody(
    tabItems(
        tabItem(tabName = "map",
                         # gvisGeoChart
                fluidRow(box(htmlOutput("map")),
                         # gvisHistoGram
                         box(htmlOutput("hist")))),
        tabItem(tabName = "data",
                         # datatable
                fluidRow(box(DT::dataTableOutput("table"))))
))
```

Reactive Output Map Tab



Reactive Output Data Tab



Select Item to Display

Income

Show 10 entries Search:

state.name	Population	Income	Illiteracy	Life.Exp	Murder	HS.Grad	Frost	Area
Alabama	3615	**3624**	2.1	69.05	15.1	41.3	20	50708
Alaska	365	**6315**	1.5	69.31	11.3	66.7	152	566432
Arizona	2212	**4530**	1.8	70.55	7.8	58.1	15	113417
Arkansas	2110	**3378**	1.9	70.66	10.1	39.9	65	51945
California	21198	**5114**	1.1	71.71	10.3	62.6	20	156361
Colorado	2541	**4884**	0.7	72.06	6.8	63.9	166	103766
Connecticut	3100	**5348**	1.1	72.48	3.1	56	139	4862
Delaware	579	**4809**	0.9	70.06	6.2	54.6	103	1982
Florida	8277	**4815**	1.3	70.66	10.7	52.6	11	54090
Georgia	4931	**4091**	2	68.54	13.9	40.6	60	58073

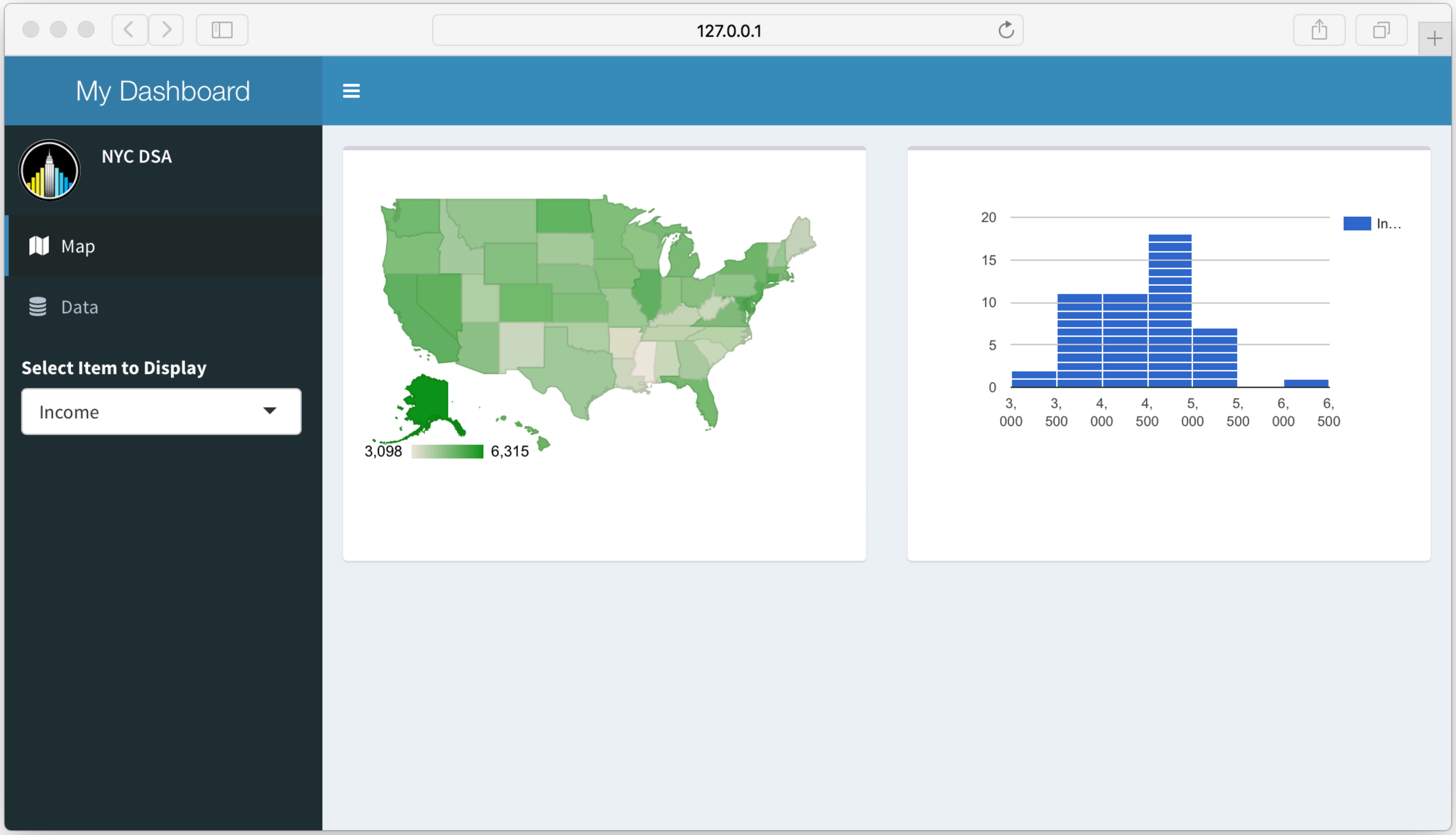
Showing 1 to 10 of 50 entries

Modifying the sizes of the boxes

- In the Map tab, we may want the two boxes to have the same height.
- In the Data tab, we may want the box that contains datatable to cover entire body width.

```
## ui.R ##
dashboardBody(
    tabItems(
        tabItem(tabName = "map",
                fluidRow(box(htmlOutput("map"),
                             height = 300),
                         box(htmlOutput("hist"),
                             height = 300))),
        tabItem(tabName = "data",
                fluidRow(box(DT::dataTableOutput("table"),
                             width = 12)))))
```

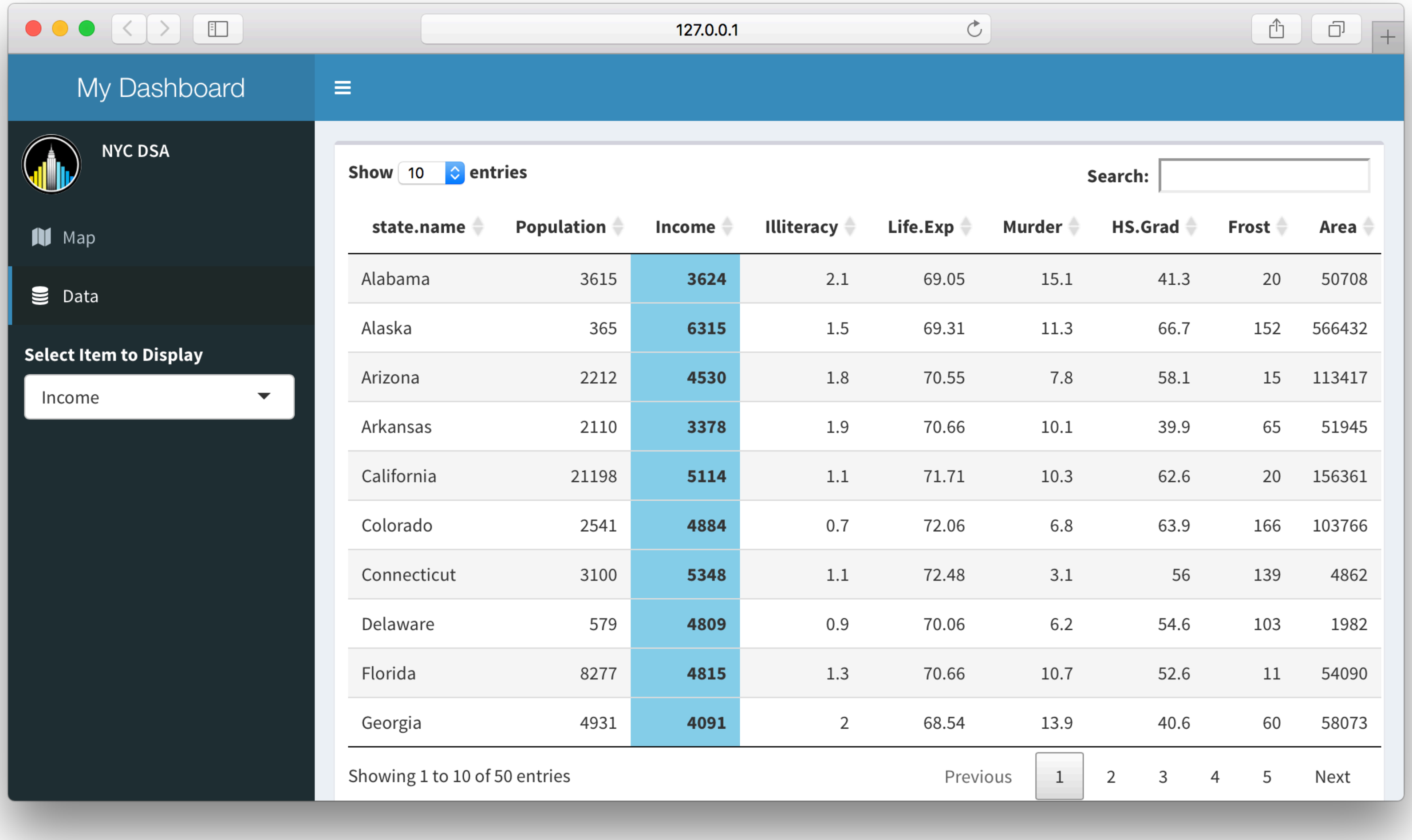
Reactive Output Map Tab



Select Item to Display

Income

Reactive Output Data Tab



Select Item to Display

Income

Show 10 entries

Search:

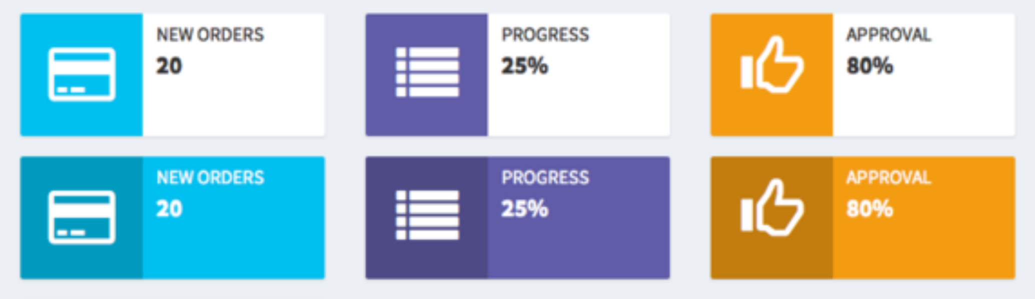
state.name	Population	Income	Illiteracy	Life.Exp	Murder	HS.Grad	Frost	Area
Alabama	3615	**3624**	2.1	69.05	15.1	41.3	20	50708
Alaska	365	**6315**	1.5	69.31	11.3	66.7	152	566432
Arizona	2212	**4530**	1.8	70.55	7.8	58.1	15	113417
Arkansas	2110	**3378**	1.9	70.66	10.1	39.9	65	51945
California	21198	**5114**	1.1	71.71	10.3	62.6	20	156361
Colorado	2541	**4884**	0.7	72.06	6.8	63.9	166	103766
Connecticut	3100	**5348**	1.1	72.48	3.1	56	139	4862
Delaware	579	**4809**	0.9	70.06	6.2	54.6	103	1982
Florida	8277	**4815**	1.3	70.66	10.7	52.6	11	54090
Georgia	4931	**4091**	2	68.54	13.9	40.6	60	58073

Showing 1 to 10 of 50 entries

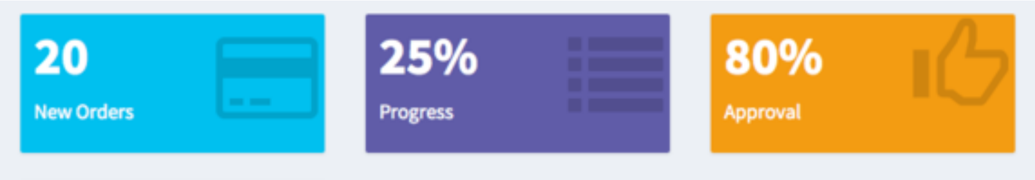
infoBox and valueBox

infoBox and **valueBox** can be used for displaying simple numeric or text values, with an icon. Here are some examples:

- infoBox



- valueBox



Building infoBox

To build an reactive infoBox object is similar to other reactive object:

- use `infoBox()` and `renderInfoBox()` in `server.R`
- use `infoBoxOutput()` in `ui.R`

Now let's build three `infoBox`es to display the following descriptive statistics of the selected column:

- state that has the highest value
- state that has the lowest value
- average value

Building infoBox in `server.R`

```r
# show statistics using infoBox
output$maxBox <- renderInfoBox({
    max_value <- max(state_stat[,input$selected])
    max_state <-
        state_stat$state.name[state_stat[,input$selected]==max_value]
    infoBox(max_state, max_value, icon = icon("hand-o-up"))
})
output$minBox <- renderInfoBox({
    min_value <- min(state_stat[,input$selected])
    min_state <-
        state_stat$state.name[state_stat[,input$selected]==min_value]
    infoBox(min_state, min_value, icon = icon("hand-o-down"))
})
output$avgBox <- renderInfoBox(
    infoBox(paste("AVG.", input$selected),
            mean(state_stat[,input$selected]),
            icon = icon("calculator"), fill = TRUE))
```

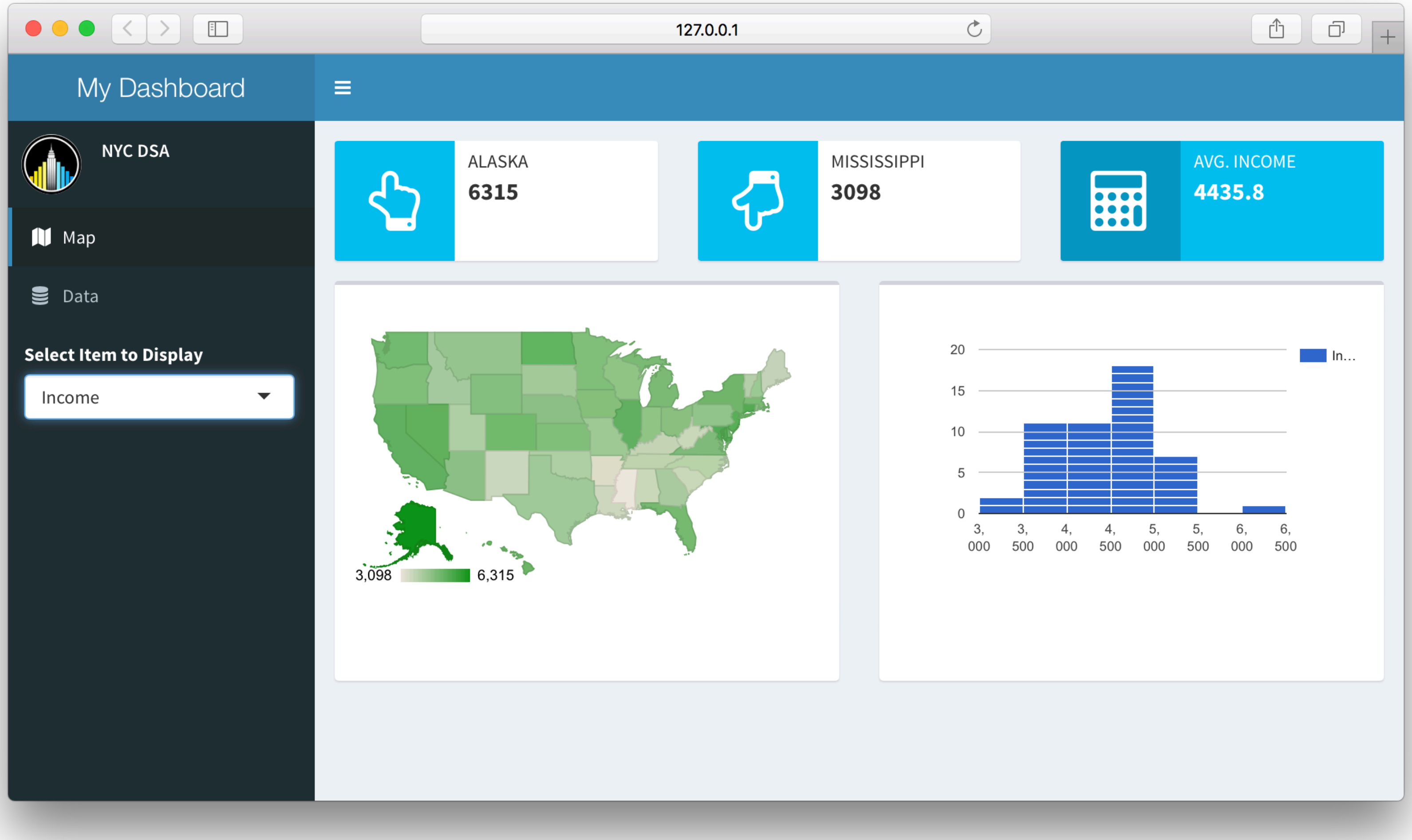
Building infoBox in `ui.R`

Now let's place the three infoBox above the map and histogram.

The first tabItem then becomes:

```r
## ui.R ##
tabItem(tabName = "map",
        fluidRow(infoBoxOutput("maxBox"),
                 infoBoxOutput("minBox"),
                 infoBoxOutput("avgBox")),
        fluidRow(box(htmlOutput("map"), height = 300),
                 box(htmlOutput("hist"), height = 300)))
```

Building infoBox



127.0.0.1

My Dashboard

NYC DSA

Map

Data

Select Item to Display

Income

ALASKA
6315

MISSISSIPPI
3098

AVG. INCOME
4435.8