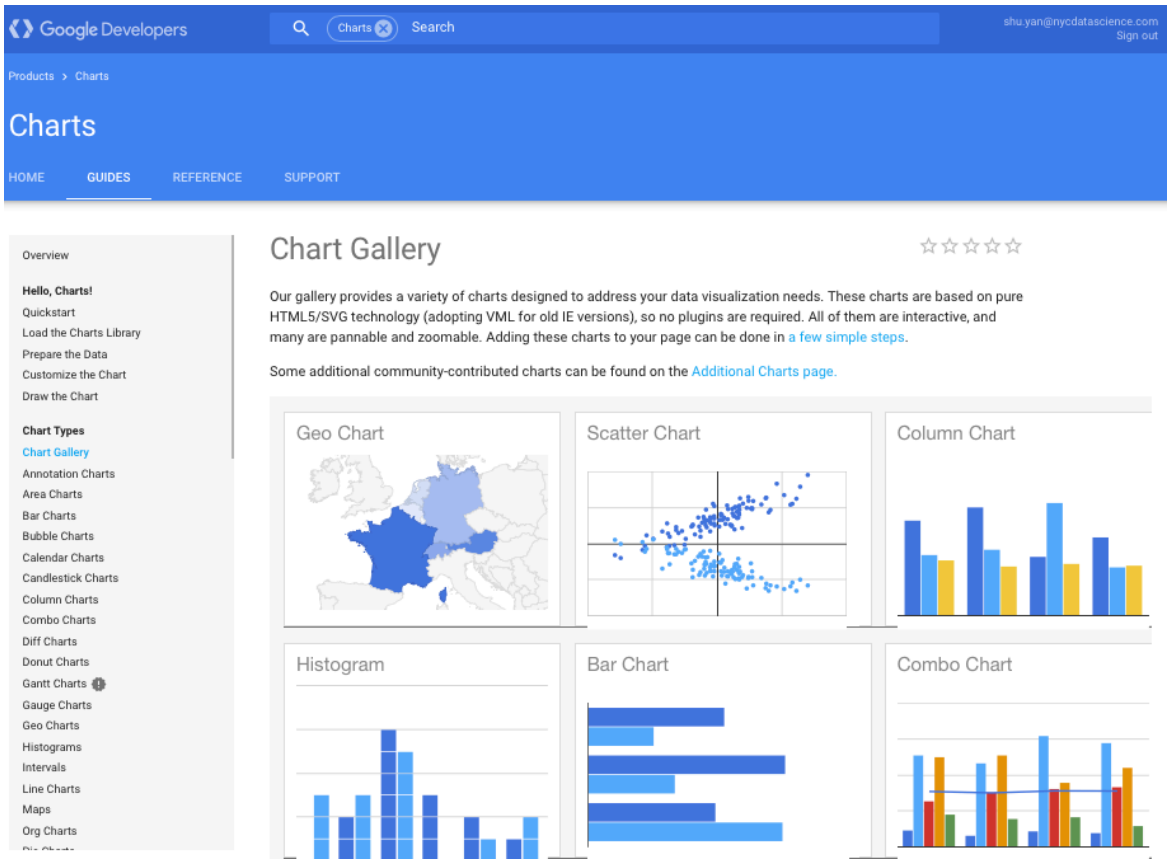
NYC DATA SCIENCE ACADEMY

googleVis / Leaflet

NYC DataScience Academy

GoogleVis API

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



2/36

Example

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

3/36

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

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

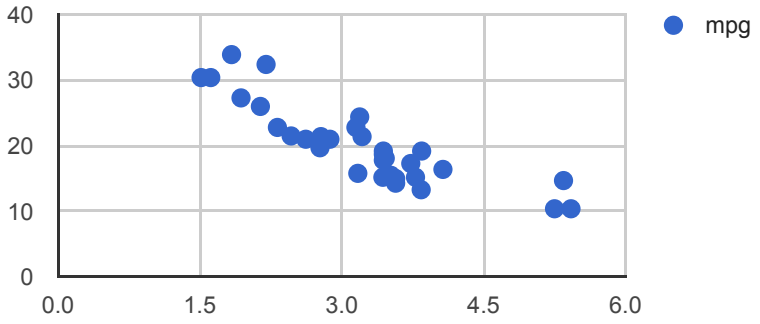
A Simple Example

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

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

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>ScatterChartID2b1c3dad9b70</title>
## <meta http-equiv="content-type" content="text/html;charse
## <style type="text/css">
## body {
##   color: #444444;
##   font-family: Arial,Helvetica,sans-serif;
##   font-size: 75%;
##   }
##   a {
##   color: #4D87C7;
##   text-decoration: none;
## }
## </style>
## </head>
## <body>
##  <!-- ScatterChart generated in R 3.3.2 by googleVis 0.6.
## <!-- Fri Apr 14 02:08:44 2017 -->
##
```

Data Format

https://developers.google.com/chart/interactive/docs/g
format

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

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

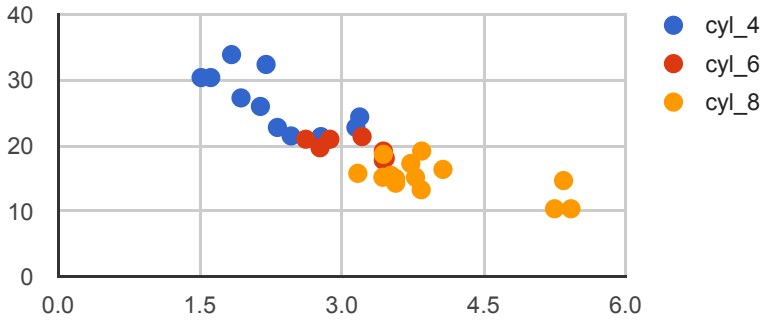
Data Format

```
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)
```

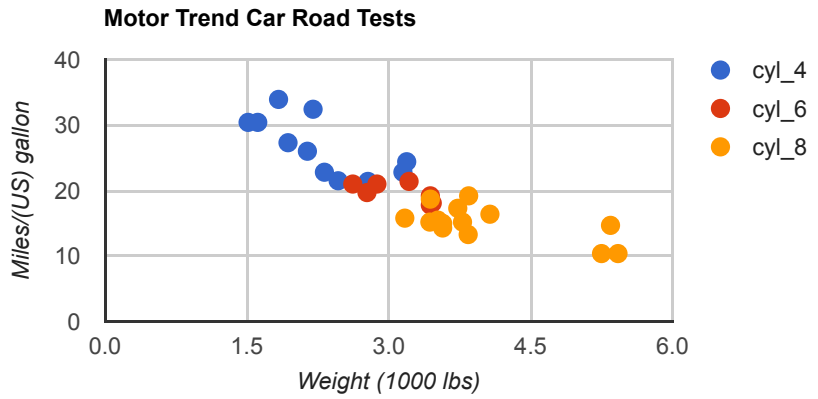


12/36

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:

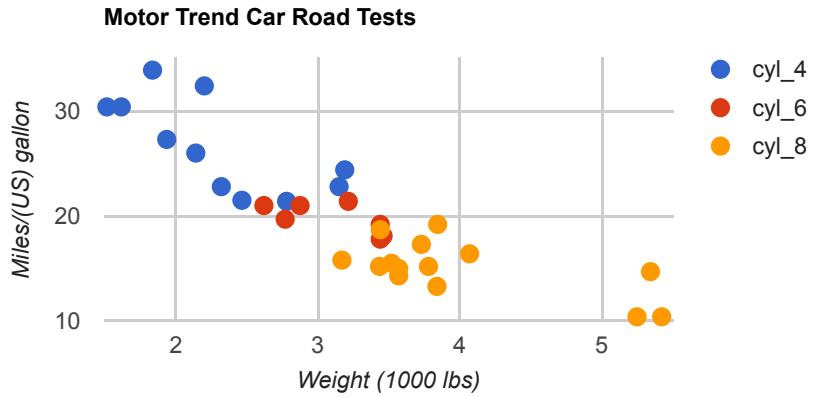
explorer.actions	The Google Charts explorer supports three actions: • `dragToPan`: Drag to pan around the chart horizontally and vertically. To pan only along the horizontal axis, use `explorer: { axis: 'horizontal' }`. Similarly for the vertical axis. • `dragToZoom`: The explorer's default behavior 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', 'rightClickT
plot(gvisScatterChart(dt,options=my_options)")
```



15/36

Optional Column Roles

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

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

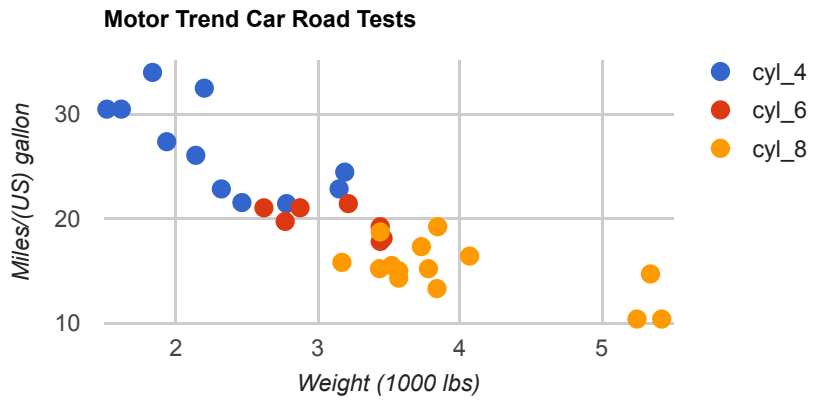
Setting Tooltips

```
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 cy
## Mazda RX4         2.620    NA          Mazda RX4  21.0
## Mazda RX4 Wag     2.875    NA      Mazda RX4 Wag  21.0
## Datsun 710        2.320  22.8         Datsun 710    NA
## Hornet 4 Drive    3.215    NA     Hornet 4 Drive  21.4
## Hornet Sportabout 3.440    NA  Hornet Sportabout    NA  H
## Valiant           3.460    NA            Valiant  18.1
##                   cyl_8 cyl_8.html.tooltip
## Mazda RX4            NA          Mazda RX4
## Mazda RX4 Wag        NA      Mazda RX4 Wag
## Datsun 710           NA         Datsun 710
## Hornet 4 Drive       NA     Hornet 4 Drive
## Hornet Sportabout  18.7  Hornet Sportabout
## Valiant              NA            Valiant
```

Setting Tooltips

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



18/36

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)
```

19/36

A Quick Example

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

20/36

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

23/36

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

```
colStates <- map("state", fill = TRUE,
                 plot = FALSE,
                 region = c("florida", "louisiana", "mississ
                            "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

29/36

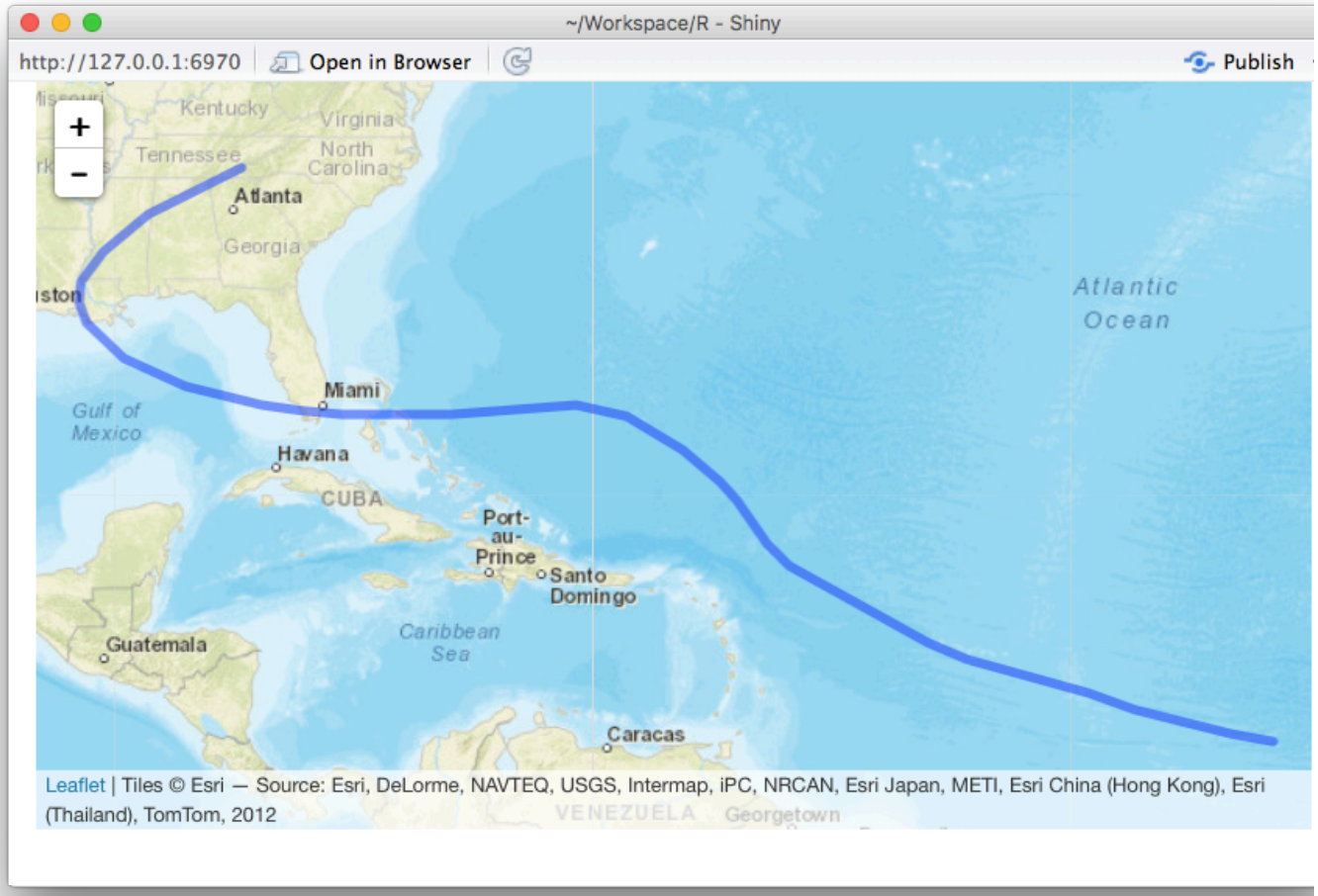
Using Leaflet with Shiny

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



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

```
colStates <- map("state", fill = TRUE, plot = FALSE,
                 region = c("florida", "louisiana", "mississ
                            "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
                            layerId = LETTERS[1:6])
    } else {
      proxy %>% removeShape(layerId = LETTERS[1:6])
    }
  })
}
```

35/36

Modifying Existing Maps