Blog Post 2: leaflet and dygraphs Packages

Nishali

November 23, 2017

Introduction:

After coding interactive elements within ShinyApp I wanted to know if there were other packages that can be used to create interactive elements within R Markdown. After doing some research on html widgets I decided to focus on both `leaflet` and `dygraphs`, because both of these packages can be used in a less data intensive matter. I wanted to find a simplified way for companies to analyze time data to look at customer retention and peak times of service, but I also wanted to learn how to create quick location visuals for customers to readily access these companies. As a graphic designer I wanted to find an easier way to create graphic elements that can easily be encoded instead of handrawn and `leaflet` would allow me to create interactve location maps that would be more exciting to use. However, as a statistics major I was really interested in analyzing time series data, but in a way that is interactive and easy to use. This post will look at two aspects of html widgets that fulfill these requirements:

Content

1. Basics of **leaflet** : using data that has latitudinal and longitudinal coordinates to create interactive maps.
2. Basics of **dygraphs** : using data involving time to create interactive line graphs.

[NOTE: This post is targeted towards people fairly familiar with R, but not experts.]

Importing Required Packages and Importing data

[You must first install the package in the r console, before you can call the packages within your r script.The packages to install appear in the form below]

Data Download

`install.packages("readr")`

`install.packages("htmlwidgets")`

Maps

`install.packages("leaflet")`

`install.packages("dplyr")`

Time Series Data

`install.packages("dygraphs")`

Downloading the Data and Packages

In order to download the time series data input the following code:

`download.file("https://pkgstore.datahub.io/core/global-temp/annual_csv/`

`data/a26b154688b061cdd04f1df36e4408be/annual_csv.csv",destfile="temperature.csv",method="libcurl")`

The maps data is already in csv format and can be downloaded directly from the website listed in the references section.

```
library(dplyr)         # data wrangling
```

```
##
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
library(readr)         # importing data
library(leaflet)       # creating maps
library(dygraphs)      # creating maps
library(htmlwidgets)   # saving widgets
maps <- read_csv('data/simplemaps-worldcities-basic.csv')
```

```
## Parsed with column specification:
## cols(
##   city = col_character(),
##   city_ascii = col_character(),
##   lat = col_double(),
##   lng = col_double(),
##   pop = col_double(),
##   country = col_character(),
##   iso2 = col_character(),
##   iso3 = col_character(),
##   province = col_character()
## )
```

```
time <- read_csv('data/temperature.csv')
```

```
## Parsed with column specification:
## cols(
##   Source = col_character(),
##   Year = col_integer(),
##   Mean = col_double()
## )
```

Leaflet Mapping

Creating Basic Maps using leaflet

The package `leaflet` is capable of creating simple but aesthetic maps. This section looks at maps both using manually inputted coordinates and using a data frame with a list of cities and coordinates.

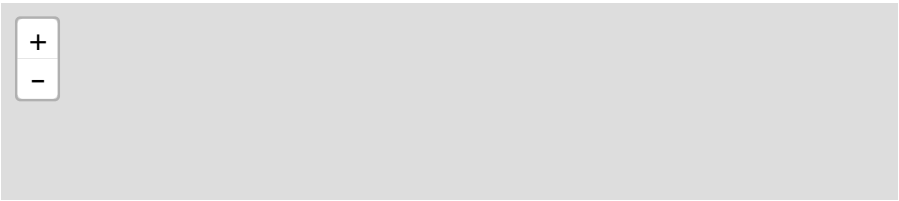
Functions used in preceding chunks

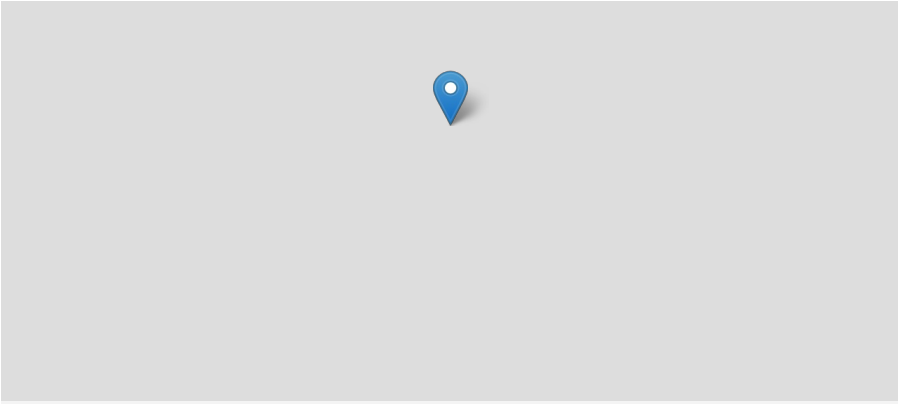
- The functions below were used in the preceding chunk:
  - `leaflet()` to create a map widget
  - `addTiles()` to add a layer to the map that contains the base structure (default OpenStreetMap tiles)
  - `addProviderTiles()` to add a layer to the map that contains the base structure from an alternative library of options
  - `addMarkers()` to add a layer to the map that contains the location marker
    - `label` to display text whenever the marker is hovered over
    - `popup` to display text whenever the marker is clicked
  - `addCircles()` to add a layer to the map that contains the location marker in the shape of a circle with the radi based on a specified variable (`addCircleMarkers()` is an simpler alternative)
  - `setView()` to set the center of the map view and the zoom level
  - `function(x){}` to create the function that changes the color of the markers.
  - `addAwesomeMarkers()` to add a layer to the map that contains the location marker but specified with a certain color
  - `clearBounds()` to set the view automatically to latitudal/longitudal data provided
  - `labelOptions()` to determine the the label visualization.
    - `noHide` specification within `labelOptions()` that determines whether the labelis displayed when the mouse is hovered over the marker
  - `markerClusterOptions()` for large data sets clusters the marker tiles into larger circles until you zoom in.
  - `labelOptions()` to alter the properties of the label that appears
  - `paste0()` to combine the coordinates to be displayed
  - `addLayersControl()` to create a panel for the changing ouputs of the interactive map
  - `filter()` a part of the dplyr package that filters the data by a specification
  - `sapply()` combines and returns results of function as a vector
  - `awesomeIcons()` to add specifications over the custom icon
  - `addRectangles()` to add a rectangle using specified latitude and longitutde coordinates.
  - `addLabelOnlyMarkers()` to add a layer with only the label text.

1st Example: Altering the Base Maps

These initial maps are meant to look at data involving a single locaion and are ideal for a one store location business. The functions below simply changes the base map (or background) of the data.

```
# Basic Leaflet Map
a <- leaflet() %>%
  addProviderTiles(providers$Esri.WorldStreetMap) %>%  # Add default
  addMarkers(lng=151.18518, lat=-33.92001, popup="Where Nemo was taken")
# Click on the icon to see the message
a
```



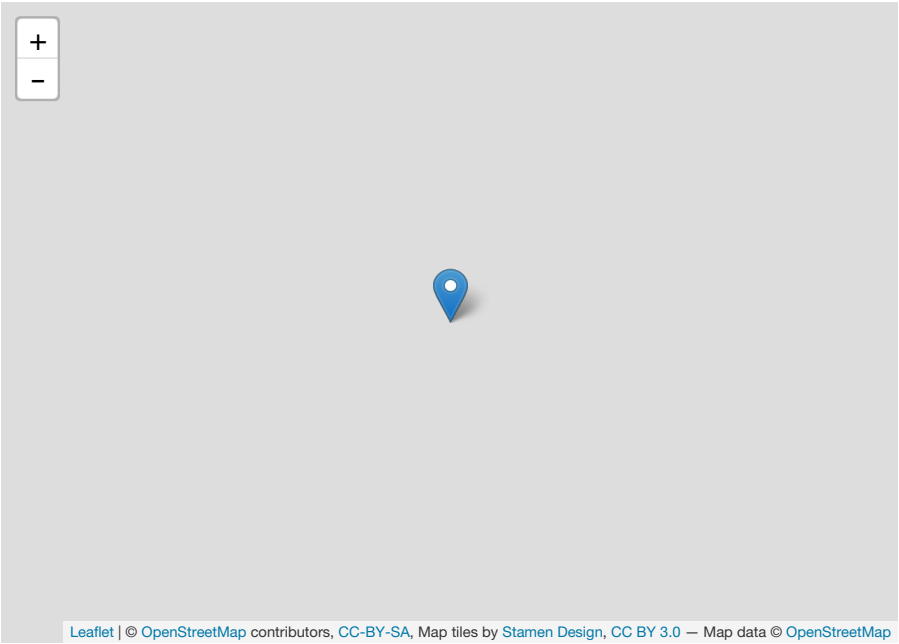


Leaflet | Tiles © Esri — Source: Esri, DeLorme, NAVTEQ, USGS, Intermap, iPC, NRCAN, Esri Japan, METI, Esri China (Hong Kong), Esri (Thailand), TomTom, 2012

```
# Some of the options provided by leaflet() to change the Basemap
head(names(providers))
```

```
## [1] "OpenStreetMap"               "OpenStreetMap.Mapnik"
## [3] "OpenStreetMap.BlackAndWhite" "OpenStreetMap.DE"
## [5] "OpenStreetMap.France"        "OpenStreetMap.HOT"
```

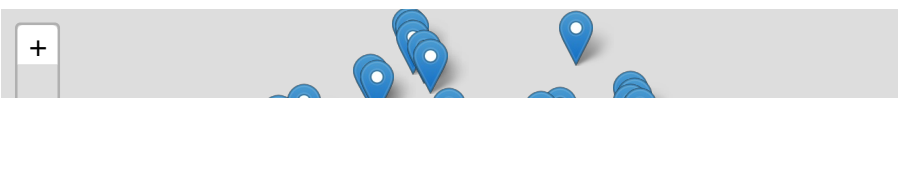
```
# Changing the Base map to Stamen.Toner
b <- leaflet() %>%
  addTiles() %>%
  addMarkers(lng=151.18518, lat=-33.92001, label="Where Nemo was taken")%>%
  addProviderTiles(providers$Stamen.Toner) %>%
  setView(151.18518,-33.92001, 9)
# Hover over the icon to view the associated text
b
```

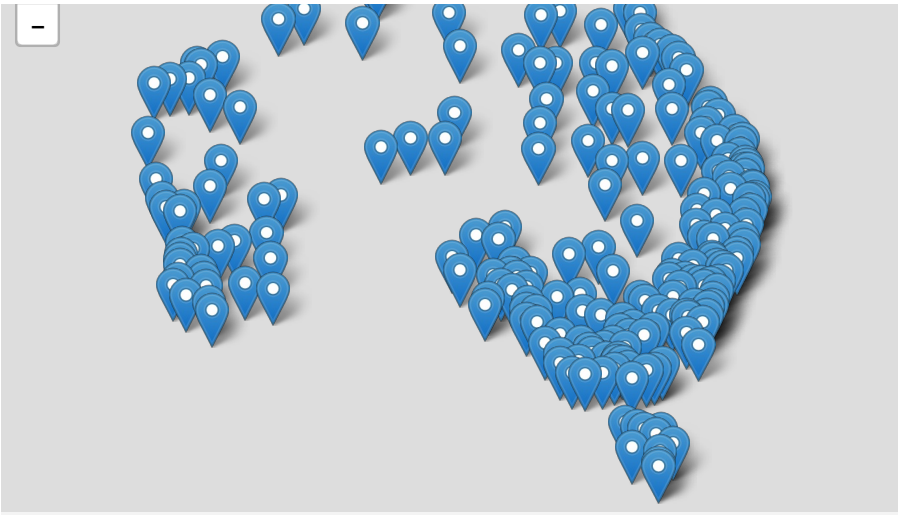


2nd Example: Changing the Markers and Popups while using a Data Set

Now that we understand the base map feature, lets look at a particular country of our data set and create different markers and pop ups based on this data set. This part looks specifically at Australia to provide markers on all the major cities in Australia, it also displays the population of the corresponding city when you hover over the marker.

```
# Using dplyr to filter the data set
AUS <- filter(maps, iso3 == 'AUS')

# Pop up icon Markers that display the city name once clicked on
c <- leaflet(data = AUS) %>%
  addProviderTiles(providers$Esri.WorldStreetMap) %>%
  addMarkers(~lng, ~lat, popup = ~as.character(city),
             labelOptions = labelOptions(noHide = T, textOnly = TRUE))%>%
  clearBounds()
# Click on the markers to see the name
c
```



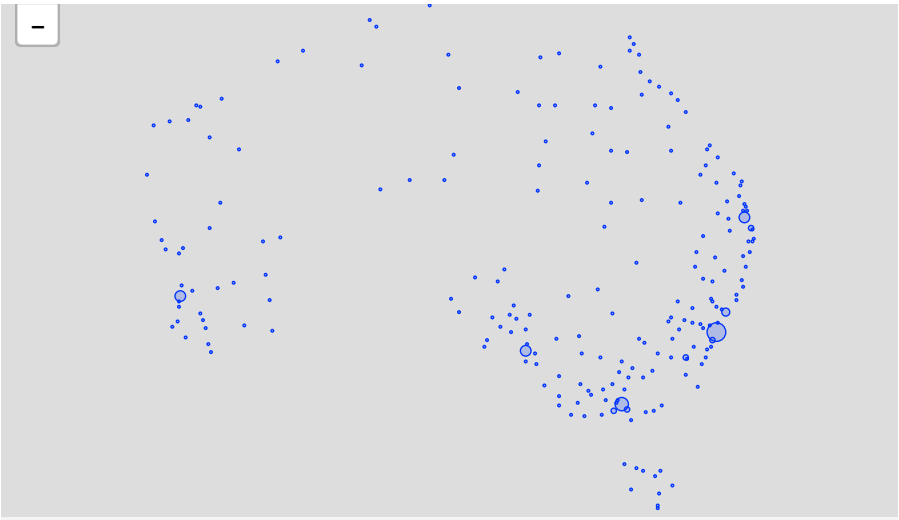


Leaflet | Tiles © Esri — Source: Esri, DeLorme, NAVTEQ, USGS, Intermap, iPC, NRCAN, Esri Japan, METI, Esri China (Hong Kong), Esri (Thailand), TomTom, 2012

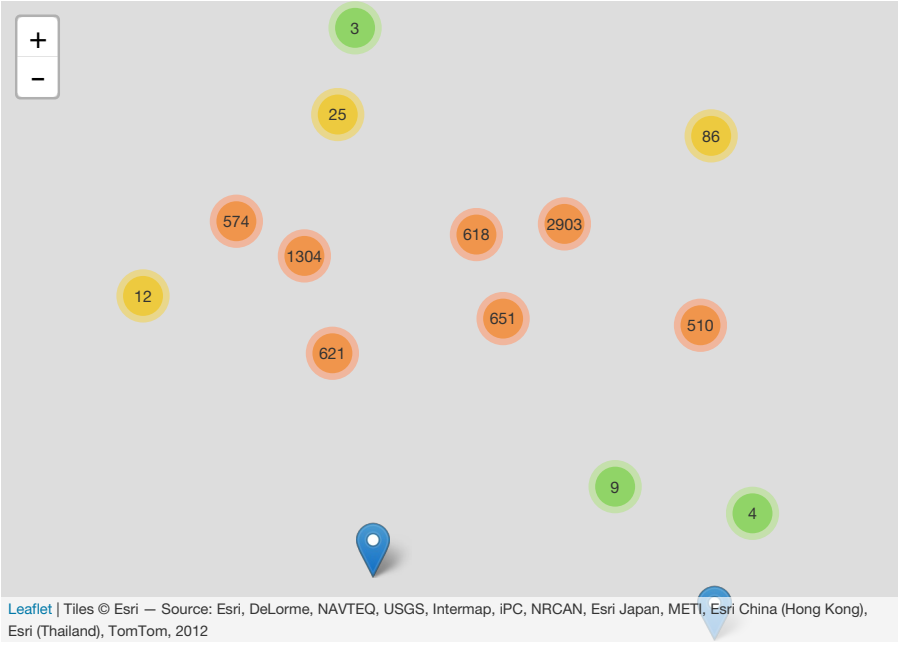
```
# Label icon Marker that changes color by population
getColor <- function(maps) {
  sapply(maps$pop, function(pop) {
  if(pop <= 5000) {
    "green"
  } else if(pop <= 50000 & pop > 5000) {
    "orange"
  } else if(pop >= 50000) {
    "red"
  } })
}

icons <- awesomeIcons(
  icon = 'ios-close',
  iconColor = 'black',
  library = 'ion',
  markerColor = getColor(AUS)
)

d <- leaflet(AUS) %>%
  addProviderTiles(providers$Esri.WorldStreetMap) %>%
  addAwesomeMarkers(~lng, ~lat, icon=icons,
                    label=~as.character(city))
d
```

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

```
# Map with Circle icons based on population
e <- leaflet(AUS) %>%
  addProviderTiles(providers$Esri.WorldStreetMap) %>%
  addCircles(~lng, ~lat, weight = 1, radius = ~sqrt(pop) * 30,
             label=~as.character(city))
e
```



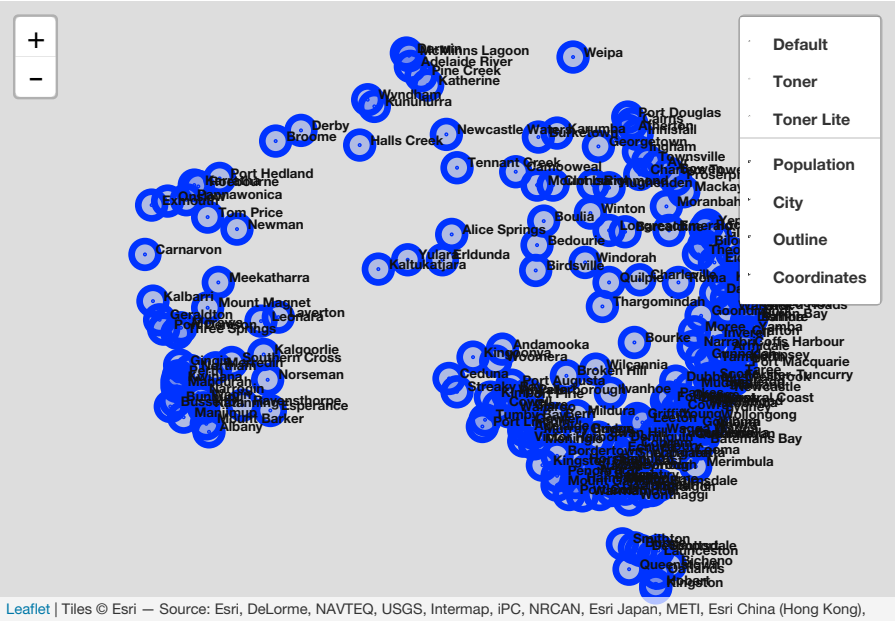

```
# For large sets of Data use the Cluster Option
f <- leaflet(maps) %>%
  addProviderTiles(providers$Esri.WorldStreetMap) %>%
  addMarkers(~lng, ~lat, clusterOptions = markerClusterOptions(),
             label=~as.character(city))
# Click on the large numbered circles to zoom in and reveal that number of data points until all that you can see are the location markers
f
```



3rd Example: Creating an Interactive Layer Display

This last example of the leaflet package, it explores the interactive elements embedded within the functions panel. In this example you can toggle between tabs to display different elements on the base map.

```
g <- leaflet(AUS) %>%
  # Base map structure
  addProviderTiles(providers$Esri.WorldStreetMap,
                   group = "Default") %>%
  addProviderTiles(providers$Stamen.Toner,
                   group = "Toner") %>%
  addProviderTiles(providers$Stamen.TonerLite,
                   group = "Toner Lite") %>%

  # Layers that involve markers overlayed on the map
  addCircles(~lng, ~lat, weight = 1, radius = ~sqrt(pop) * 40,
             label=~as.character(pop), group = "Population") %>%
  addCircleMarkers(~lng, ~lat,label = ~paste0(lng, ",", lat),
                   group = "Coordinates") %>%
  addRectangles(
    lng1=150.600, lat1= -34.8829,
    lng2=151.815, lat2=-32.84535,
    fillColor = "", group= "Outline")%>%
   addLabelOnlyMarkers(~lng, ~lat,label = ~as.character(city),
    labelOptions = labelOptions(noHide = T, textOnly = TRUE),
    group= "City") %>%

  # Panel that controls all of the layers specified above
  addLayersControl(
    baseGroups = c("Default", "Toner", "Toner Lite"),
    overlayGroups = c("Population","City", "Outline", "Coordinates"),
    options = layersControlOptions(collapsed = FALSE)
  )
# Click on the checkboxes in the top righthand corner of the map to alter the elements in the graph
g
```



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

Dygraphs

Creating Interactive line maps using dygraphs

The package `dygraphs` is capable of creating simple but aesthetic maps. This section looks at maps both using manual input and using a data frame.

Functions used in preceding chunks

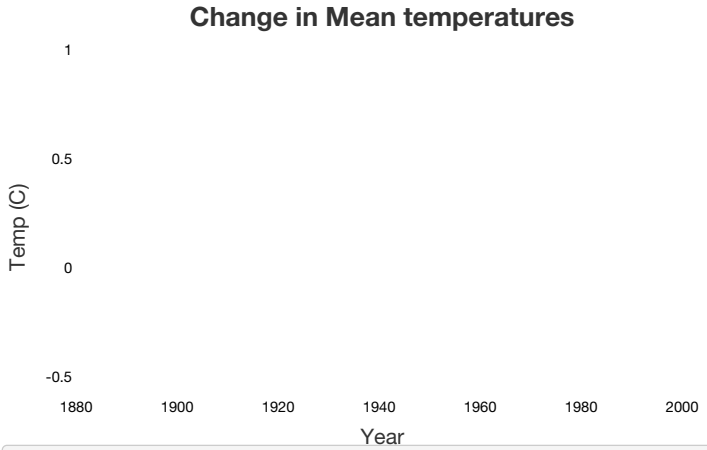
- The functions below were used in the preceding chunk:
  - `filter()` to alter the original data frame by seperating it by source
  - `arrange()` to arrage the data frame in increasing order
  - `dygraph()` to create a simple line graph, with the x-axis as the time variable
  - `dyOptions()` to add modifications to the original graph
    - `drawPoints` to display the data points in addition to the line connecting the points
    - `fillGraph` to fill the area under the line to the x-axis
    - `fillAlpha` to determine the opacity of the chosen modification
  - `cbind()` to combine the two data frames
  - `dyHighlight()` to highlight a certain variable or line with certain specifications
  - `dySeries()` to specify conditions for all the graphs connected to that graph
  - `dyRangeSelector()` to create an interactive range bar at the bottom of the plot

1st Example: Basic Line Plot using dygraphs

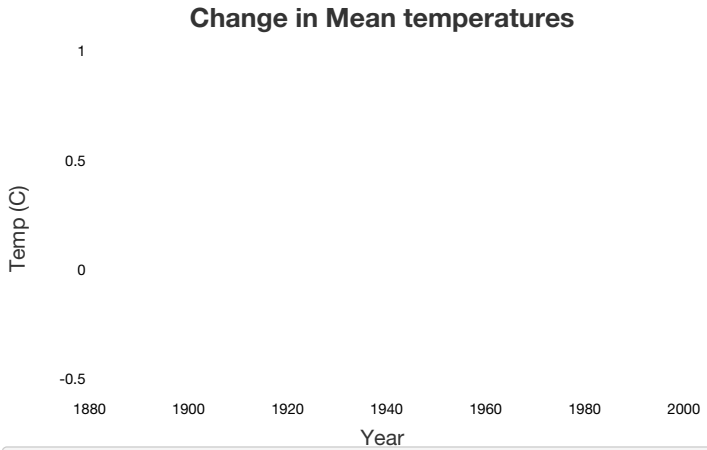
1st Example: Basic Line Plot using dygraphs

This portion looks at plotting the change in the global mean temperature based on the GIS source data. The first graph is a simple line plot, the second one has the points from the data set plotted as points on the line, and the third graph includes a fill under the line to help show the area under the curve.

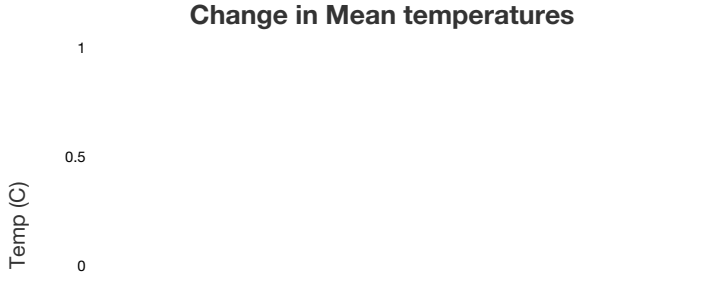
```
#Data Wrangling

# Creating two data frames based on Source
GCtemp <- filter(time, Source == 'GCAG')
GItemp <- filter(time, Source == 'GISTEMP')

# Removing the Source column
GCtemp$Source <- NULL
GItemp$Source <- NULL

# reordering the data in increaseing order of year
GCtemp <- arrange(GCtemp,Year)
GItemp <- arrange(GItemp,Year)

# Basic Plot
dygraph(GCtemp, main = "Change in Mean temperatures",
        ylab = "Temp (C)",xlab = "Year")
```




```
# Hover over the lines to reveal the data points in the upper right corner

# Plot with line and additional points from data set
dygraph(GCtemp, main = "Change in Mean temperatures",
        ylab = "Temp (C)", xlab = "Year") %>%
  dyOptions(drawPoints = TRUE, pointSize = 2)
```



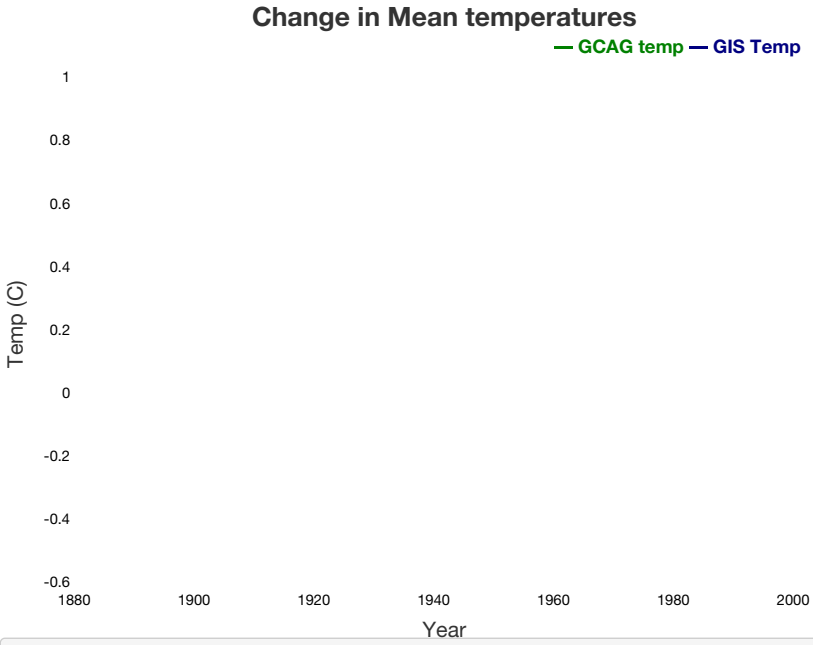

```
# Plot with the are under the curve filled in
dygraph(GCtemp, main = "Change in Mean temperatures",
        ylab = "Temp (C)",xlab = "Year") %>%
  dyOptions(fillGraph = TRUE, fillAlpha = 0.4)
```



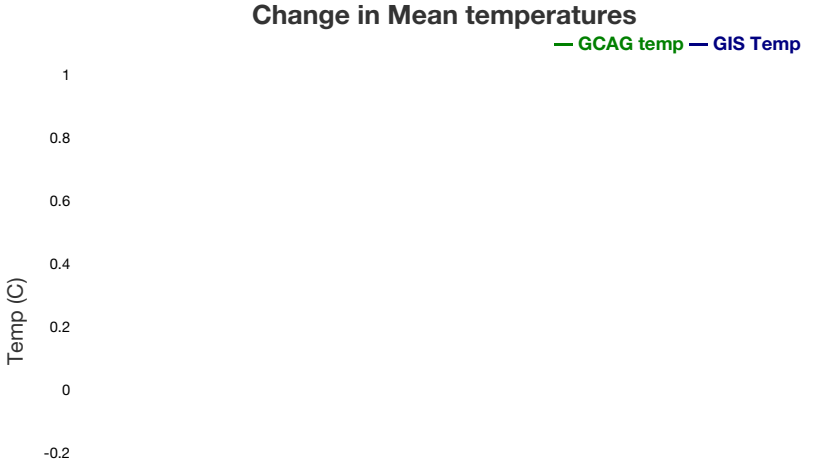
2nd Example: Plotting multiple Lines

This portions looks at plotting the change in the global temperature based on both the GIS and GCAG source data. The first graph is a simple line plot of the two lines, the second one highlights the line that the cursor is currently on, and the third graph includes a fill under both of the lines to help show the area under the curve.

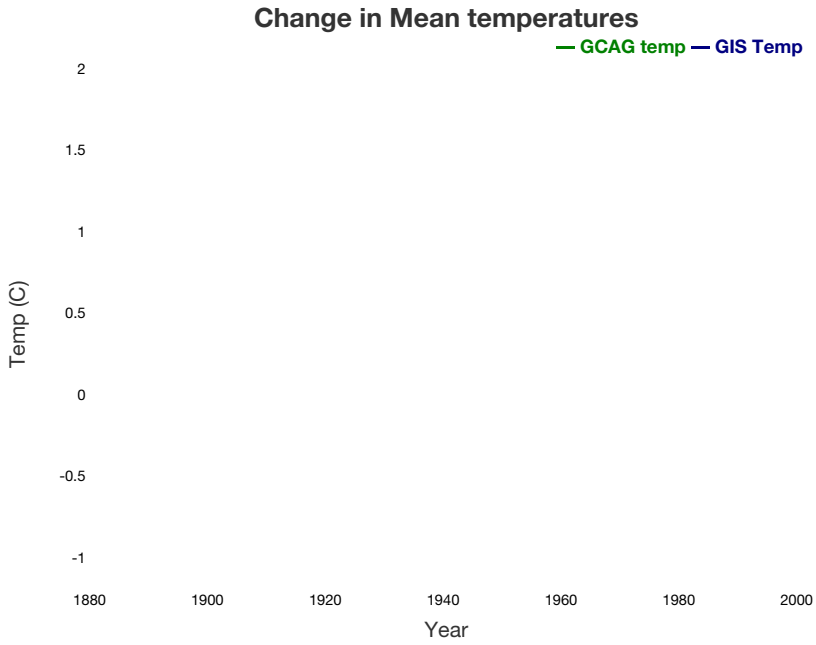
```
# More Data Wrangling

# Combining the two sets of Data
temp <- cbind(GCtemp, GItemp)

# Removing the extraneous year category
temp[,3] <- NULL

# Renaming the columns
colnames(temp) <- c("Year", "GCAG temp", "GIS Temp")


# 2 lines on one graph
dygraph(temp, main = "Change in Mean temperatures",
        ylab = "Temp (C)", xlab = "Year")
```




```
# 2 lines on one graph with a highlighted variable
dygraph(temp, main = "Change in Mean temperatures",
        ylab = "Temp (C)", xlab = "Year") %>%
  dyHighlight(highlightCircleSize = 5,
              highlightSeriesBackgroundAlpha = 0.2,
              highlightSeriesOpts = list(strokeWidth = 3),
              hideOnMouseOut = FALSE)
```



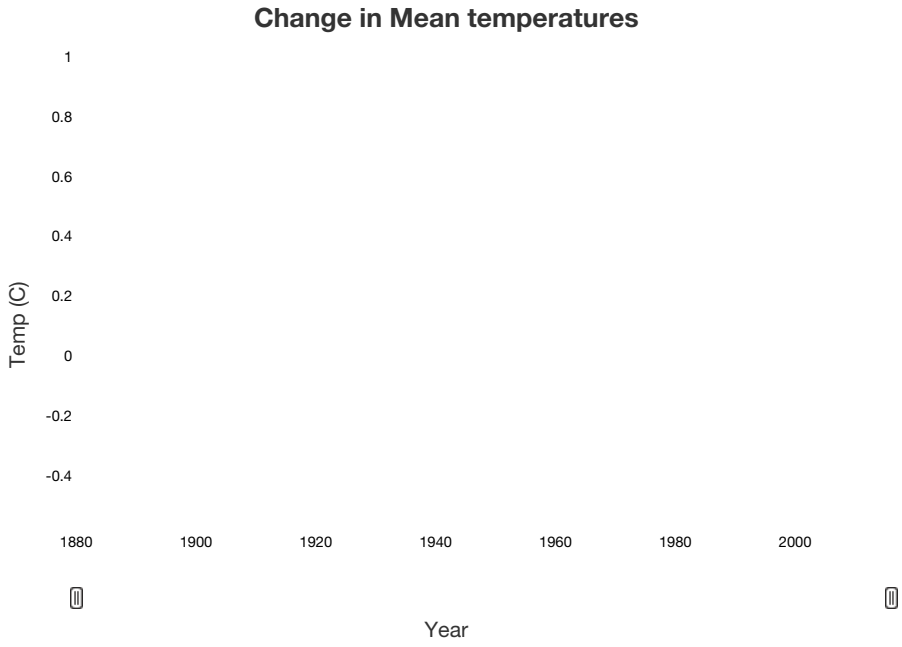

```
# Hover over the lines to reveal the data points in the upper right corner

# 2 lines stacked on a graph with fill under the curve
dygraph(temp, main = "Change in Mean temperatures",
        ylab = "Temp (C)", xlab = "Year") %>%
  dySeries("GCAG temp", label = "GCAG temp") %>%
  dySeries("GIS Temp", label = "GIS Temp") %>%
  dyOptions(stackedGraph = TRUE)
```



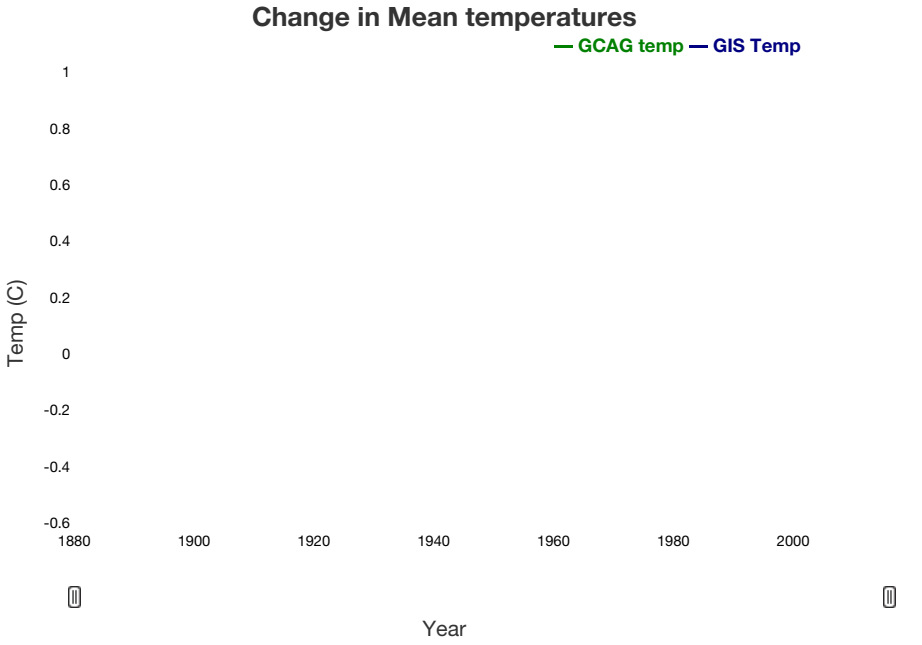
3rd Example: Line Plot with range selection

This portions looks at plotting the change in the global mean temperature based on both the GIS and GCAG source data, but respective of a range specified by the viewer. By changing the width of the bar at the bottom the plot changes to display the designated data within that time frame. The first graph contains only one line graph and the second one contains both line graphs.

```
# One line plotted of the GC sourced temp with an interactive range selection
dygraph(GCtemp, main = "Change in Mean temperatures",
        ylab = "Temp (C)",xlab = "Year") %>%
  dyRangeSelector()
```



```
# Move the edges of the slider to change the data plotted on the graph

# Two lines on one graph with range selection
dygraph(temp, main = "Change in Mean temperatures",
        ylab = "Temp (C)", xlab = "Year") %>%
  dyHighlight(highlightCircleSize = 5,
              highlightSeriesBackgroundAlpha = 0.2,
              hideOnMouseOut = FALSE) %>%
  dyRangeSelector()
```



> **Purpose:**
>
> - To gain a better understanding of the features of html widgets within R markdown.
> - To practice working with real raw data collected from online data services.

Conclusions

This was my first time creating an interactive element outside of Shinyapp, as well as using data visualization to display Time Series data. To summarize what I have discussed here, leaflet is a really useful package that allows you to create informative maps with a few lines of code. Dygraphs can be used to create interactive line graphs that allow you to analyze time series data while simultaneously providing an interactive output. Leaflet's interactive elements included a click box that controls the image output, marker titles that appear when hovered over, and pop-up text that appears when clicked on. Dygraph's interactive elements include a moving point cursor that displays the values of the data set as the cursor moves along the line, a highlighted feature that highlights the current line when hovered over, and a range selector that changes the dates that the graph plots by sliding the edges of the data range. I hope you found all the information here helpful.

References:

This file contains both information I learnt in class and information I researched. Below is a list of the references I used:

1. **Data Collected from:**
   - Global Temperature Time Series
   - Geographic Data
   - How to Download Data Directly from a URL
   - Label only leaflet
2. **Leaflet:**
   - Leaflet basics
   - Advanced Leaflet
3. **Dygraphs:**
   - html widgets
   - dygraphs basics
   - dygraphs tutorial
4. **List of references:**
   - rmd cheat sheet