Google Charts for Institutional Research Websites

Jorge Martinez, University of Houston

February 11, 2018

Introduction

Welcome to the 2018 TAIR Conference Workshop *Google Charts for Institutional Research Websites*. In this workshop, we will learn how to use Google Visualization API to construct sleek and interactive graphics for your websites. We will learn how to transform traditional tables of institutional data into easily consumable graphics.

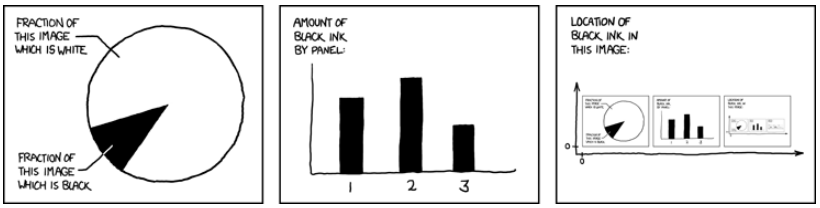

Figure 1: Self-Description by xkcd (xkcd.com).

What is Google Charts API?

Google APIs, or Application Programming Interfaces, allow developers to integrate Google services with other services. These services include Google search features, translations, YouTube, calendars, and maps on websites. Developers can also use Google APIs to enhance smart phone apps, such as retrieving geocoordinates or calculating drive times for delivery or navigation services. In this workshop we will use Google Charts API.

Google Charts "allows you to create charts and reporting applications over structured data and helps integrate these directly into your website," (Google Charts, FAQ). The data *lives* inside the HTML (webpage) file that is downloaded by the user when they go to your site. This means all of the data that goes into each visualization is processed locally - there is no data transferred to Google. The only data that is transferred between the HTML file and Google are the data and chart configurations. These are sent through a formatted URL that specifies what kind of chart you are making and the options you want to apply to the data in the HTML file. They tell your browser how to draw the chart on your page. Charts are rendered using HTML5/SVG and can be viewed on web browsers, smart phones, or tablets.

Jorge Martinez, *Senior Research Analyst*, jxm@uh.edu

In order to communicate with the Google API and to create your charts, you will need to use JavaScript. JavaScript is one of the three programming language of the web. They are:

1. **HTML** defines the contents of a page, like text, links, and images.
2. **CSS** specifies the layout of the page, such as styles and how HTML elements will be displayed.
3. **JavaScript** programs the behavior of web pages, such as what happens when a user scrolls over a bar graph or what happens when a user clicks on an element on the web page.

In this workshop, we will touch on select elements of these languages that will help us construct our charts. You do not need a background in these languages beyond what will be presented. We will rely on the R Statistical Programming Package to help us get beyond this knowledge gap and have it write the HTML, CSS, and JavaScript for us.[1]

[1] For more information about R and the programs used in this workshop, please refer to your pre-workshop handout for download instructions.

Your Tool Belt

We will use `R` to construct all of our charts, chart options, and page layout. R is a programming language primarily used for statistical computing and graphics.[2] It is an open-sourced program that is popularly used in academia, but has been growing significantly in private and public industries.[3] If you know other programming languages such as SAS, SPSS, STATA, Python, etc, you will pick-up R relatively quickly. If you don't have any programming experience, don't worry - we will make this as easy as possible.

[2] See *What is R?*

[3] See *"The Popularity of Data Analysis Software"*, *"SAS, R, or Python Survey 2016"*, and *"Data Analyst Captivated by R's Power"*

After constructing our charts, we will use `Brackets` to help us finely tune our webpages. `Brackets` is an open-sourced program created by Adobe used to edit the languages of the web: HTML, CSS, and JavaScript. We will use this tool towards the end of the workshop to create the layout for our page and to see what our charts will look like on the web.

Why Visualize a Data Dashboard?

Websites are important tools for establishing good first impressions. Parents and students will navigate to university websites to help them determine the academic quality of that institution. Among the top 23 reasons for choosing a college, students ranked "information from a website" as number 11 among the factors they said were "very important" in influencing their final college selection.[4] Naturally, academic reputation, job placement, financial aid, cost of attendance, visiting campus, student life, size, graduate programs, graduation rates, and geographic location superseded website information in the decision-making processes. Interestingly, the website was decidedly more important than national rankings and parental influences, which makes website content all the more important.

[4] *Freshmen Students Say Rankings Aren't Key Factor in College Choice*, US News 2013.

Although not all parents and students are likely to seek-out IR homepages to make their decisions, we do know that university leadership, community members, and other stakeholders visit our sites for more granular detail. Traditionally, we have used tabular data in our *Facts-At-A-Glance* and *Statistical Handbook* to provide detailed information about our students, faculty, and academics.[5] Other universities have adopted expensive licenses to utilize Tableau and SAS Visual Analytics to create interactive data dashboards for their institutional metrics.[6] Although these are good tools for data visualization, they can be inaccessible for some IR offices. With a little bit of programming, we can overcome this obstacle and construct nice-looking dashboards with open-sourced platforms.

[5] Click for *Facts at a Glance* and *Statistical Handbook*.

[6] See *Texas A&M Accountability*, *University of Texas Institutional Reporting, Research and Information Systems* for examples using Tableau, and the *University of North Carolina* or the *University of Texas System seekUT Dashboard* for SAS VA examples.

Prepare Your Data

Google Charts requires specific data structures for specific charts. All charts are constructed using a `DataTable`. A `DataTable` is a two-dimensional table composed of rows and columns. Each column is a specific data type (string, number) with a label (ethnicity, count). `DataTable` requirements vary by chart type. The simplest `DataTable` has at least two columns, one for the data label and one for the value. These are used in line, bar, and pie charts. Let's start with a simple `dataTables` (`Table 1`). The first data type is string (Ethnicity). The other three columns are numeric data types (Male, Female, Total). We will discuss more complicated `dataTables` as we encounter them later in the workshop (see Data Tree example on pg.16)

Table 1: Example `DataTables`: Enrollment by Race/Ethnicity and Gender, Fall 2017.

Ethnicity	Male	Female	Total
African American	1896	2505	4401
Asian American	4770	4570	9340
Hawaiian/P.I.	51	24	75
Hispanic	6550	7323	13873
International	2218	1647	3865
Multiracial	685	663	1348
Native American	34	37	71
Unknown	362	400	762
White	6190	5439	11629

Chart Anatomy

Each chart uses JavaScript embedded in the HTML file. That JavaScript tells your browser what to do with the data and how to draw the chart. Here is a complete example of code for a pie chart:

```
<html>
  <head>
    <!--Load the AJAX API-->
    <script type="text/javascript" src="https://www.gstatic.com/charts/loader.js"></script>
    <script type="text/javascript">

      // Load the Visualization API and the corechart package.
      google.charts.load('current', {'packages':['corechart']});

      // Set a callback to run when the Google Visualization API is loaded.
      google.charts.setOnLoadCallback(drawChart);

      // Callback that creates and populates a data table,
      // instantiates the pie chart, passes in the data and
      // draws it.
      function drawChart() {

        // Create the data table.
        var data = new google.visualization.DataTable();
        data.addColumn('string', 'Topping');
        data.addColumn('number', 'Slices');
        data.addRows([
          ['Mushrooms', 3],
          ['Onions', 1],
          ['Olives', 1],
          ['Zucchini', 1],
          ['Pepperoni', 2]
        ]);

        // Set chart options
        var options = {'title':'How Much Pizza I Ate Last Night',
                       'width':400,
                       'height':300};

        // Instantiate and draw our chart, passing in some options.
        var chart = new google.visualization.PieChart(document.getElementById('chart_div'));
        chart.draw(data, options);
      }
    </script>
```

```
  </head>

  <body>
    <!--Div that will hold the pie chart-->
    <div id="chart_div"></div>
  </body>
</html>
```

The resulting chart would look like this:

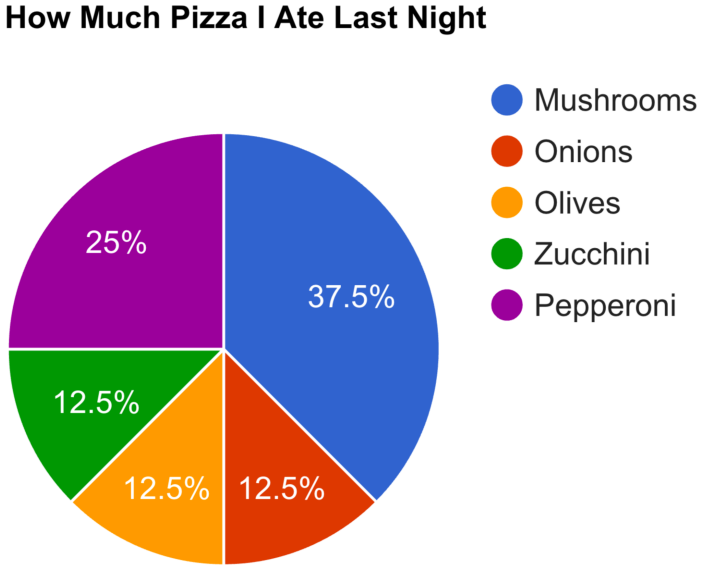

Figure 2: How much pizza I ate last night, an example. Source: Google.

You could copy and paste this entire code as a .html file on your computer and see it using your web browser. The first line identifies the file as HTML. The `<head>` is the element that contains all of the HTML metadata, or data about data. This is placed before the `<body>`, or the container that holds the web content. The metadata defines various things such as titles, styles, links, scripts, etc. In our case, our chart JavaScript code will live in the `<head>`.

The first part of the JavaScript in the `<head>` loads the loader. You only need to define this once no matter how many charts you produce in this one HTML file. The part describing `google.charts.load` loads the packages necessary for the charts you will construct. `Current` specifies that you want the current chart configurations, and `corechart` gives the configurations for most charts, including bar, column, line, area, stepped area, bubble, pie, donut, combo, candlestick, histogram, and scatter. We will load different chart types later. The callback pulls the chart configurations after they are loaded and used when the object of the callback is defined as it is in the next part, the function.

Inside `function drawChart()` you will have all the `DataTable` ele-

ments along with the data you want to visualize followed by the chart options which include color, chart titles, dimensions, etc. Finally, after the data and options are set, you call the specific chart you are creating (PieChart) and draw the chart. The chart is drawn but it does not appear until you call it in the body. We will place the drawn charts in device containers using HTML. Device containers are essentially placeholders for the object being called, in this case the pie chart that was previously drawn in the JavaScript.

So far, all of this seems a little complicated and esoteric if you do not know JavaScript. Fortunately, we have R to help us write the JavaScript in a program that can be reproduced each time you want to update your web charts.

Constructing Your Charts in R

If you haven't already done so, please install R and R Studio as outlined in the Pre-Conference Workshop Handout. In this section we will learn how to:

1. Load the `googleVis` package
2. Load the data into R
3. Write code to specify our charts
4. Write a file containing all the necessary JavaScript to use in our websites.

Load the `googleVis` Package

Next, we will install the package that contains the commands for the Google Charts. Execute the following command:

```
install.packages('googleVis')
```

```
# Load library
library(googleVis)
```

```
## Creating a generic function for 'toJSON' from package 'jsonlite' in package 'googleVis'
```

```
##
## Welcome to googleVis version 0.6.2
##
## Please read Google's Terms of Use
## before you start using the package:
## https://developers.google.com/terms/
##
## Note, the plot method of googleVis will by default use
## the standard browser to display its output.
```

```
##
## See the googleVis package vignettes for more details,
## or visit http://github.com/mages/googleVis.
##
## To suppress this message use:
## suppressPackageStartupMessages(library(googleVis))
```

Loading Your Data

You can load your data in one of two ways. You can point to a file using the "Import Dataset" feature in your environment tab, or you can execute the following code that points to your data file:

```
data <- read.csv("C:/File/Path/chart_data.csv")
```

For example

```
trendUH <- read.csv("I:/IR Staff Area/Jorge/TAIR Workshop 2017/trendUHln.csv")
```

After loading this data, it will appear in your environment as a new `Data Frame`. These will be the `DataTables` mentioned earlier and will contain all the necessary data for your visualizations.

Chart Functions

This package makes it easy to define our charts without having to write the JavaScript. The naming convention for each function starts with `'gvis + ChartType'`. For example, `gvisLineChart` will define line charts, `gvisColumnChart` will define bar charts, etc. Here is a sample function:

```
trendUHln <- gvisLineChart(trendUH, xvar="Year", yvar="Count",
                           options=list(), chartid = "trendUHln")
```

The `<-` defines an object called `trendUHln` as a Google line chart using the function `gvisLineChart`. This function uses the data `trendUH` that has an x variable for the chart called "Year" and a y variable for enrollment "Count" to be plotted. We give the chart some options inside the `()` in `options=list()`. This is where we will define the chart dimensions, labels, titles, colors, etc. Finally, we give the chart a unique identifier using `chartid= "chartName"` which we will use to call the chart in our HTML documents. Let's make some charts.

Line Chart

Line charts show bivariate relationships between one variable along the x axis and one along the y axis. In this example, we will show enrollment over time:

```
# Load library
library(googleVis)

# Load your data
trendUH <- read.csv("I:/IR Staff Area/Jorge/TAIR Workshop 2017/trendUHln.csv")

# view your data
View(trendUH)

# see data structure
str(trendUH)
```

```
## 'data.frame':    83 obs. of  2 variables:
##  $ Year: int  1934 1935 1936 1937 1938 1940 1941 1942 1943 1944 ...
##  $ Fall: int  909 948 1249 1285 1563 2488 2494 1508 1104 2720 ...
```

```
# run a summary of the data
summary(trendUH)
```

```
##       Year           Fall
##  Min.   :1934   Min.   :  909
##  1st Qu.:1956   1st Qu.:11918
##  Median :1976   Median :28907
##  Mean   :1976   Mean   :23371
##  3rd Qu.:1996   3rd Qu.:32474
##  Max.   :2017   Max.   :45364
```

```
# create a simple line chart
trendUHln <- gvisLineChart(trendUH, xvar="Year", yvar="Fall",
                           options=list(), chartid = "trendUHln")

# plot the chart
plot(trendUHln)
```

R will open the default web browser on your computer to draw the chart. In this example, we can see all the values in our DataTable between the min and max values, years 1934 and 2017, respectively. Lets pull in another DataTable for a more complex look at a line chart.

```
# load data
stuDegTrend <- read.csv("I:/IR Staff Area/Jorge/TAIR Workshop 2017/stuDegTrendLn.csv")

# view data
View(stuDegTrend)
head(stuDegTrend,5)
```

```
##   Year Certificate Associate Bachelors
## 1 1935          NA        27        81
## 2 1936          NA        28       118
## 3 1937          NA        28       154
## 4 1938          NA         9       165
## 5 1939          NA         7       139
##   Masters Doctoral Professional Total
## 1      NA       NA           NA   108
## 2      NA       NA           NA   146
## 3      NA       NA           NA   182
## 4      NA       NA           NA   174
## 5      NA       NA           NA   146
```

```
stuDegTrendLn <- gvisLineChart(stuDegTrend, xvar="Year",
                               yvar=c("Bachelors", "Masters",
                                      "Doctoral", "Professional"),
                               chartid = "stuDegTrend")

plot(stuDegTrendLn)
```

In this example, we pull in more than two columns to represent multiple categories. The `c()` function is used to create a vector of values. You could define variable `z` as a vector of numbers 1-5, and `a` as another vector with numbers 6-10:

```
z <- c(1,2,3,4,5)
z
```

```
## [1] 1 2 3 4 5
```

```
a <- c(6,7,8,9,10)
a
```

```
## [1]  6  7  8  9 10
```

```
# you can also use c() to concatenate vectors
az <- c(z,a)
az
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10
```

In this case, we are concatenating all of the values for each degree type, Bachelors, Masters, etc.

Exercise 1: Enrollment trends over time

Construct a line graph to show your institution's enrollment over time. What changes would you make for the aesthetics?

Customizing Your Charts

Each chart type requires similar yet specific `DataTable` structures and each has their own customization options. In this section, we will go through some of the common customization features that fall into the `options=list()` argument.

Colors

Color is pretty, but color is also meaningful. Your color choices should be carefully considered. Each element of your graphic should be informative and necessary. For example, is it necessary to use customized colors for the singular line in `trendUHln`? Why or why not?

Specifying colors is pretty easy. Let specify some colors to the degrees conferred line chart:

```
stuDegTrendLn <- gvisLineChart(stuDegTrend, xvar="Year",
                               yvar=c("Bachelors", "Masters",
                                      "Doctoral", "Professional"),
                               chartid = "stuDegTrend",
                               options = list(
                                 colors="['#C8102E', '#00B388',
                                 '#F6BE00', '#888B8D', '#960C22']"))
```

Size

```
stuDegTrendLn <- gvisLineChart(stuDegTrend, xvar="Year",
                               yvar=c("Bachelors", "Masters",
                                      "Doctoral", "Professional"),
                               chartid = "stuDegTrend",
                               options = list(
                                 colors="['#C8102E', '#00B388',
                                 '#F6BE00', '#888B8D', '#960C22']",
                                 width=500,
                                 height=500))
```

Titles

```
stuDegTrendLn <- gvisLineChart(stuDegTrend, xvar="Year",
                               yvar=c("Bachelors", "Masters",
                                      "Doctoral", "Professional"),
                               chartid = "stuDegTrend",
                               options = list(
                                 colors="['#C8102E', '#00B388',
                                 '#F6BE00', '#888B8D', '#960C22']",
                                 width=800, height=500,
                                 title="Academic Level, 1935-2017",
                                 hAxis="{title:'Year'}"))
plot(stuDegTrendLn)
```

Although the main configuration is all you will probably need for your charts, there are still plenty customizable features. Google offers a very helpful guide to all chart types: `https://developers.google.com/chart/interactive/docs/`. We will use this resource to help us customize our charts. A faster and more direct way to get to each charts' options can be found by using `help()` for each of the gvis functions (line, bar, pie, etc.):

```
help('gvisLineChart')
help('gvisBarChart')
```

Exercise 2: Enrollment by Level

Create a line chart using your institutional data by academic level. If you cannot access that information quickly, use UH data by navigating to uh.edu/ir/reports and click on "Quick Semester Statistics" for Fall 2012 to Fall 2017. Include the following specifications:[7]

1. Give a title for your chart.
2. Add labels for the horizontal and vertical axes.
3. Choose different colors for each line.
4. Set chart dimensions to 800W x 500H.
5. Place the legend at the bottom of the chart.
6. Add points to your lines.

[7] Hint: use the `help()` function to find customization parameters.

Bar and Column Charts

From experience, bar charts are ubiquitously popular in institutional research. Bar charts are a favorite because they are very easy to understand. Google charts has two types of bar charts: horizontal (bar charts) and vertical (column charts). They work the same way. In the

next example, we construct a bar chart to show degrees conferred by race/ethnicity.

```
# load data
stuDegR <- read.csv("I:/IR Staff Area/Jorge/TAIR Workshop 2017/stuDegRBar.csv")

# see data
stuDegR
```

```
##          Ethnicity Total
## 1 African American   835
## 2   Asian American  1764
## 3    Hawaiian/P.I.    24
## 4         Hispanic  2444
## 5    International  1163
## 6      Multiracial   235
## 7 Native American     17
## 8          Unknown   155
## 9            White  2811
```

```
# create bar chart
stuDegRBar <- gvisBarChart(stuDegR, xvar="Ethnicity",
                                yvar="Total",
                                chartid="stuDegRBar")

# plot chart
plot(stuDegRBar)
```

This representation is a bit on the ugly side. Let's order the data:

```
# order data descending
stuDegR2 <- stuDegR[order(-stuDegR$Total),]

stuDegRBar2 <- gvisBarChart(stuDegR2, xvar="Ethnicity",
                                yvar="Total",
                                chartid="stuDegGenRBar")

plot(stuDegRBar2)
```

Notice how the `xvar` is set to Ethnicity and `yvar` is set to Total even though the chart x-axis is the total value. If you substitute the function `gvisBarChart` with `gvisColumnChart`, you get ethnicity on the x-axis and total on the y-axis as a column chart. Remember that `xvar` will be associated with the major categories and `yvar` with the values, not necessarily what is drawn on the axis in the resulting chart.

Exercise 3: Degrees Conferred Bar and Column Charts

Using your data or the sample data below, create one bar and one column chart showing number of degrees conferred by Race/Ethnicity and by Gender. Give the following attributes:

1. Chart title
2. Axes titles
3. Colors
4. Sort by greatest to smallest values

Bonus: Change the legend names from Female/Male to Women/Men.

Ethnicity	Female	Male	Total
African American	598	311	909
Asian American	958	848	1806
Hawaiian/P.I.	8	5	13
Hispanic	1363	1041	2404
International	527	650	1177
Multiracial	129	104	233
Native American	4	10	14
Unknown	73	55	128
White	1489	1423	2912

HTML Tooltips

Rolling over chart elements offers detailed information used to construct it. These are called tool tips and they can be very helpful when you want to include custom information about your data. We create special role columns that follow the column of the data you want to customize. For example:

Ethnicity	Female	Female.html.tooltip	Male	Male.html.tooltip
African American	598	African American, Female, 598 (65.8%)	311	African American, Male, 311 (34.2%)
Asian American	958	Asian American, Female, 958 (53%)	848	Asian American, Male, 848 (47%)
Hawaiian/P.I.	8	Hawaiian/P.I., Female, 8 (61.5%)	5	Hawaiian/P.I., Male, 5 (38.5%)
Hispanic	1363	Hispanic, Female, 1363 (56.7%)	1041	Hispanic, Male, 1041 (43.3%)
International	527	International, Female, 527 (44.8%)	650	International, Male, 650 (55.2%)
Multiracial	129	Multiracial, Female, 129 (55.4%)	104	Multiracial, Male, 104 (44.6%)
Native American	4	Native American , Female, 4 (28.6%)	10	Native American , Male, 10 (71.4%)
Unknown	73	Unknown, Female, 73 (57%)	55	Unknown, Male, 55 (43%)
White	1489	White, Female, 1489 (51.1%)	1423	White, Male, 1423 (48.9%)

When specifying role columns, it must follow the column of the

data it describes and it must include the name of the column it describes plus `html.tooltip`, in this example `Female.html.tooltip`. Then, include these tooltip columns in the `yvar` portion of the chart function in your `R` code.[8]

[8] For more information on working with roles, follow *this link*.

Exercise 4: Customizing Tooltips

Using your chart from *Exercise 3*, modify your `DataTable` to include customized tooltips. You can create any content you'd like for the tooltips. For one of the tooltips cells, type in the following: `<img src="https://goo.gl/SD3Oid"/>`. Then, in your options list parameter, include the option `tooltip="{isHtml:'TRUE'}"`. What does this do to the tooltip for the cell where you entered the URL?

Pie Charts

People tend to dislike pie charts because they don't find them useful. I agree to a certain degree, but they are effective at showing proportionality to a whole. I also find Google makes nice pie charts and they become much more effective with the tooltips. The `DataTable` structure is very simple for pie charts. All you need is the category and the value for each category. The variable naming convention is a little different, however:

```
stuRes <- read.csv("I:/IR Staff Area/Jorge/TAIR Workshop 2017/stuResPie.csv")

stuResPie <- gvisPieChart(stuRes,
                          labelvar="Residence",
                          numvar="Number",
                          chartid = "stuResPie",
                          options=list(width=500, height=350,
                                       title="Residency",
                                       colors="['#C8102E','#960C22', '#640817',
                                       '#F6BE00','#00B388']"))

plot(stuResPie)
```

Exercise 5: Student Enrollment by Classification Pie Chart

Create a pie chart using your data or table 4 in the pre-conference workshop handout. Give it a title, custom dimensions, and colors for each category. Modify your code to create a donut chart and experiment with the size of this parameter. Finally, remove the slice labels to encourage users to hover over each slice. Remember to use the `help()` function to find the names for each option.

GeoCharts

Situating data along spatial boundaries is not only fun, but informative for recruiting and enrollment. GeoCharts allows us to plot a numeric value along some political boundary. The `DataTable` for GeoCharts is also a simple one: one variable for location and another for color, or the color assigned to a range of values. Location can be one of two formats. The first can be latitude & longitude coordinates. The second can be an address, country name, region name, or US metropolitan area. You must also specify three important options:

1. Region: The area to display; can be 'world', continent or subcontinent, country, or a state.
2. Display Mode: If you want to display color on regions (e.g., states), set this parameter to 'region'. If you want to place markers on regions, set it to 'markers'. Think of these as drop-pins on a map. Finally, you can superimpose text from the `DataTable` onto a region.
3. Resolution: This option draws the lines on your region. These can be 'countries' inside of continents, 'provinces' (or states) within a country, or 'metros' within provinces.

```
stateUG <- read.csv("I:/IR Staff Area/Jorge/TAIR Workshop 2017/stuEnrUGMap.csv")

stuEnrUGMap <- gvisGeoChart(stateUG, locationvar='State', colorvar='Percent',
                            chartid = "stuEnrUGMap",
                            options=list(region="US",
                                         displayMode="regions",
                                         resolution="provinces",
                                         width=500, height=350,
                                         colorAxis="{colors:['#C8102E']}"))

plot(stuEnrUGMap)
```

Exercise 6: GeoChart of Student Enrollment

Using your data, construct a GeoChart showing your institution's student enrollment by state. What happens if you change `displayMode` from `"regions"` to `"text"`? What about setting the same option to `"markers"`? What happens when you hover over New York or New Jersey using text and markers modes?

Tree Maps

Tree maps, like the name suggests, visualize hierarchical data structured from one main root to its many branches. In institutional re-

search, tree maps are effective at showing university, college, department, etc. level data. You start with the main level and drill down to the branching levels. This representation requires a specific `DataTable` structure:

ID	Parent	CountCurrent	CountLast	Diff
College		43774	40747	3027
Architecture	College	700	711	-11
Arts	College	1527	1576	-49
Business	College	6182	6039	143
Education	College	2522	2629	-107
Engineering	College	4969	4602	367
HRM	College	1055	1182	-127
Law	College	800	875	-75
Lib Arts & Soc Sci	College	10948	10347	601
NSM	College	5523	5681	-158
Nursing	College	94	0	94
Optometry	College	433	436	-3
Pharmacy	College	717	1241	-524
Social Work	College	482	402	80
Technology	College	6356	3712	2644
Exploratory Studies	College	1466	1314	152
School of Art	Arts	796	854	-58
Arts Interdept	Arts	35	0	35
Music	Arts	526	550	-24
Theatre and Dance	Arts	169	172	-3
Accountancy & Taxation	Business	1162	1265	-103
Business Interdepartmental	Business	3180	2966	214
Dec. & Information Sci	Business	753	783	-30
Finance	Business	559	532	27
Management	Business	181	185	-4
Marketing & Entrepre	Business	347	308	39
Curriculum & Instruction	Education	1168	1459	-291
Educ Ldrshp & Pol Std	Education	237	102	135
Education Interdept	Education	5	28	-23
Psych Health & Learning Sci	Education	1112	1038	74

The first column of the `DataTable` contains the `ID`. The `ID` must have at least two levels. The first row will be the main root of the treemap, in this case College. Since this row is the main root, it will not have a value for the `Parent` column. The `Parent` column associates the following rows with the main root. This is where we input values for each college at the university. The next parent level should be associated with a name in the higher level. In this case, the next

parent level is the department. Here is the sample code:

```
stuEnrCD <- read.csv("I:/IR Staff Area/Jorge/TAIR Workshop 2017/stuEnrCDTree.csv",
                     na.strings="")

stuEnrCDTree <- gvisTreeMap(stuEnrCD, idvar="ID", parentvar="Parent",
                            sizevar = "CountCurrent", colorvar = "Diff",
                            chartid = "stuEnrCDTree",
                            options=list(width=1100, height=500,
                                         minColor='#d7191c',
                                         midColor='#ffffbf',
                                         maxColor='#1a9641',
                                         fontColor='black',
                                         title="Change in College & Department
                                                Enrollment, Fall 2012 to 2017",
                                         showScale=TRUE,
                                         showTooltips=TRUE))

plot(stuEnrCDTree)
```

Merging Charts

There are two ways to creating a dashboard with Google Charts. The first involves creating `<div>` containers that define divisions or section in an HTML document. This method is discussed later under *Situating Charts on Your Webpage*. The second method utilizes the `gvisMerge` function. `gVisMerge` takes two gvis objects and places them onto one page. Here is one example:

```
# Undergrad State Map
stuEnrUGMap <- gvisGeoChart(stateUG, locationvar='State', colorvar='Percent',
                            chartid = "stuEnrUGMap",
                            options=list(region="US",
                                         displayMode="regions",
                                         resolution="provinces",
                                         width=500, height=350,
                                         colorAxis="{colors:['#C8102E']}"))

# Student Residency
stuResPie <- gvisPieChart(stuRes,
                          labelvar="Residence",
                          numvar="Number",
                          chartid = "stuResPie",
                          options=list(width=500, height=350,
                                       title="Residency",
```

```
                                    colors="['#C8102E','#960C22', '#640817',
                                    '#F6BE00','#00B388']"))

# gvisMerge
merge <- gvisMerge(stuEnrUGMap, stuResPie, horizontal=TRUE)
plot(merge)
```

Exercise 7: Merging Charts using `gvisMerge`

Using the charts you constructed in exercises 1-6, create a 3x3 chart dashboard.

Printing the Underlying Code

`R` easily plots `googleVis` objects as self-containing `HTML` files in your default browser from the console. Once you are satisfied with your charts, you can print and save the `HTML` source code easily with the following commands:

```
print(stuResPie, 'chart')
```

```
## <!-- PieChart generated in R 3.4.3 by googleVis 0.6.2 package -->
## <!-- Wed Feb 07 16:06:36 2018 -->
##
##
## <!-- jsHeader -->
## <script type="text/javascript">
##
## // jsData
## function gvisDatastuResPie () {
## var data = new google.visualization.DataTable();
## var datajson =
## [
##  [
## "Harris County",
## 24545
## ],
## [
## "Adjacent Counties",
## 11163
## ],
## [
## "Other Texas Counties",
## 4629
## ],
```

```
## [
## "Out-of-State",
## 1162
## ],
## [
## "International",
## 3865
## ]
## ];
## data.addColumn('string','Residence');
## data.addColumn('number','Number');
## data.addRows(datajson);
## return(data);
## }
##
## // jsDrawChart
## function drawChartstuResPie() {
## var data = gvisDatastuResPie();
## var options = {};
## options["allowHtml"] = true;
## options["width"] = 500;
## options["height"] = 350;
## options["title"] = "Residency";
## options["colors"] = ['#C8102E','#960C22', '#640817',
##                                         '#F6BE00','#00B388'];
##
##
##     var chart = new google.visualization.PieChart(
##     document.getElementById('stuResPie')
##     );
##     chart.draw(data,options);
##
##
## }
##
##
## // jsDisplayChart
## (function() {
## var pkgs = window.__gvisPackages = window.__gvisPackages || [];
## var callbacks = window.__gvisCallbacks = window.__gvisCallbacks || [];
## var chartid = "corechart";
##
## // Manually see if chartid is in pkgs (not all browsers support Array.indexOf)
## var i, newPackage = true;
```

```
## for (i = 0; newPackage && i < pkgs.length; i++) {
## if (pkgs[i] === chartid)
## newPackage = false;
## }
## if (newPackage)
##   pkgs.push(chartid);
##
## // Add the drawChart function to the global list of callbacks
## callbacks.push(drawChartstuResPie);
## })();
## function displayChartstuResPie() {
##   var pkgs = window.__gvisPackages = window.__gvisPackages || [];
##   var callbacks = window.__gvisCallbacks = window.__gvisCallbacks || [];
##   window.clearTimeout(window.__gvisLoad);
##   // The timeout is set to 100 because otherwise the container div we are
##   // targeting might not be part of the document yet
##   window.__gvisLoad = setTimeout(function() {
##   var pkgCount = pkgs.length;
##   google.load("visualization", "1", { packages:pkgs, callback: function() {
##   if (pkgCount != pkgs.length) {
##   // Race condition where another setTimeout call snuck in after us; if
##   // that call added a package, we must not shift its callback
##   return;
## }
## while (callbacks.length > 0)
## callbacks.shift()();
## } });
## }, 100);
## }
##
## // jsFooter
## </script>
##
## <!-- jsChart -->
## <script type="text/javascript" src="https://www.google.com/jsapi?callback=displayChartstuResPie"></s
##
## <!-- divChart -->
##
## <div id="stuResPie"
##   style="width: 500; height: 350;">
## </div>

# Save as HTML file
print(stuResPie, 'chart', file='I:/IR Staff Area/Jorge/TAIR Workshop 2017/stuResPie.html')
```

Situating Charts on Your Webpage

This section of the workshop will take the charts you have constructed in R and compile them in an HTML file for the web. The most basic website has the following features:

```
<html>

  <head>

  Your Google Chart code will go here. These include
  jsData, jsDrawChart, and jsDisplayChart.

  </head>

  <body>

  All webpage elements go in the body, including the <div> container that calls your chart.

  </body>

</html>
```

The best practice (so far) is to save the HTML output from R and embed that code in an existing IR webpage. The webpage will already contain the necessary style elements for your site, including university logos, colors, banners, navigation, links, etc. The first step will be to identify where your charts will live on your website and save that page to your computer by right-clicking the page and saving as a complete webpage. You can then copy and paste the chart code into the HTML file you just saved and edit using Brackets or any other HTML editor.

R will produce an HTML file with the following sections:

1. jsHeader: This signals the beginning of a JavaScript chunk.
2. // jsData: This section holds all of the DataTable information with the appropriate columns and rows containing your institutional data.
3. // jsDrawChart: This section provides all of the specifications and attributes to your visualizations, including dimensions, legends, titles, colors, etc.
4. // jsDisplayChart: This part is responsible for loading all the necessary packages for displaying the jsData and the chart specifications in jsDrawChart.
5. // jsFooter: These lines of script load the charts.

The above components should be copied into the <head> of your HTML file. You will call these charts to be drawn in the <body> using

 tags. tags define a division or section in an HTML document. We will call each chart using tags by their unique chart IDs we created in R. Here is an example:

```
<h3 style="text-align: center;">Enrollment</h3>
<div class="row">
<div class="col-md-6 col-sm-6 col-xs-12">
<div align="center" class="chart" id="stuEnrGenRCol"></div>
</div>
<div class="col-md-6 col-sm-6 col-xs-12">
<div align="center" class="chart" id="stuFtptCol"></div>
</div>
</div>
```

This code block shows one tag that defines a row with two more tags that create a space for two charts. Each of these tags are nested and are closed with corresponding tags. The chart will be called using the chart id defined inside the tag. This is where your chart will appear on your website.

Exercise 8: Create your webpage

Download the website where you envision your charts will go. Open brackets to embed your chart code in the HTML file you just downloaded. Include some navigation tabs at the top of your page.

Additional Resources

The purpose of this workshop was to introduce a basic understanding of how to construct Google Charts so that you can take these skills and develop your charts further. As a group, we created a small subset of charts, including bar, pie, line, geo, and treemap charts. There are plenty more charts are you disposal. Your mastery of the basic charts are easily transferable to any available charts.

To help keep you moving forward, I have assembled a list or helpful resources for further exploration below:

1. Google Charts Reference Guide: https://developers.google.com/chart/
2. Introduction to googleVis 0.6.2: https://cran.r-project.org/web/packages/googleVis/vignettes/googleVis.pdf
3. googleVis package documentation: https://cran.r-project.org/web/packages/googleVis/googleVis.pdf
4. An Introduction to R: https://cran.r-project.org/doc/manuals/r-release/R-intro.pdf

5. Swirl - Learn R in R: `http://swirlstats.com/`
6. W3Schools HTML Tutorial: `https://www.w3schools.com/html/`
7. `ggmap` Spatial Visualization with ggplot2: `https://journal.r-project.org/archive/2013-1/kahle-wickham.pdf`