**Welcome**

I'm glad that you decided to take *Developing Data Products*, part of the [Data Science Specialization](https://www.coursera.org/specialization/jhudatascience/1?utm_medium=listingPage) from Johns Hopkins Biostatistics!

A data product is the production output from a statistical analysis. Data products automate complex analysis tasks or use technology to expand the utility of a data informed model, algorithm or inference. This course covers the basics of creating data products using Shiny, R packages, and interactive graphics. This course focuses on the statistical fundamentals of creating a data product that can be used to tell a story about data to a mass audience.

You will learn how to communicate using statistics and statistical products. Emphasis will be paid to communicating uncertainty in statistical results. You will learn how to create simple Shiny web applications and R packages for their data products. In addition, we'll cover reproducible presentations and interactive graphics.

We believe that the key word in Data Science is "science". Our specialization is focused on providing you with three things: (1) an introduction to the key ideas behind working with data in a scientific way that will produce new and reproducible insight, (2) an introduction to the tools that will allow you to execute on a data analytic strategy, from raw data in a database to a completed report with interactive graphics, and (3) on giving you plenty of hands on practice so you can learn the techniques for yourself. This course represents the final cog in a data science application, creating an end-usable data product.

We are excited about the opportunity to attempt to scale Data Science education. We intend for the courses to be self-contained, fast-paced, and interactive.

Some Basics

A couple of first week housekeeping items. First, make sure that you've had R Programming and the Data Scientist's Toolbox. Reproducible Research would be helpful, but is not mandatory. At a minimum you must know: very basic git, basic R and very basic knitr.

An important aspect of this class is to peruse the materials in the github repository. All of the most up to date material can be found here

[https://github.com/bcaffo/courses/tree/master/09\_DevelopingDataProducts](https://eventing.coursera.org/redirect/71l0TKqzo36iY2eav5Sc9VVoe_wEIiVpgQts6Se2v72ClSQoofTKrg7-lRitnqfJb4mnWUYuHfNRZOypknu0ew.ZZKZ_PsfcVe6s8ixGMdTnQ.8J_xuMkaPR8GWD_faLUpndKTMd6SyWCuzXNhO-qpAFxidoVaBHhhKdam496pI7Dzcr6gJDKey0G85NrcNmNLyoOhXaiBagAmvKFPlMqrRAnZbRjXGqMb5K1-2WFMXTxRBv4WntCirnZ3Tw2yaToyA7EdLXeEnAYscfRokeuv8jY6jp4ciPvSJXFJIkzyjdh663OpKM7JXVcrV1ob_QxtE0d7rAb3ZGf3O5738ldRXGqrA28OzFDwDGZc9wamqBq2QF4YStDLkqGY7uV8NolopWgzEvDywQ0CbpeNzHRS5kIgA4Gv_-hu8yoE6MW0su0fNc-u5fFM39cRCbloSSfmOA)

You should clone this repository as your first step in this class and make sure to fetch updates periodically. (Please send pull requests too!) It is one of the most essential components of the Specialization that you start to use Git frequently. We're practicing what we preach as well by using the tools in the series to create the series, especially git.

You can clone the whole repo with (http)

git clone <https://github.com/bcaffo/courses.git>

or (ssh)

git clone git@github.com:bcaffo/courses.git

The lectures are in the index.Rmd lecture files. In this class, we'll cover how to create these sorts of slides. You will see all of the R code to recreate the lectures. Going through the R code is the best way to familiarize yourself with the lecture materials.

**The lecture material for this class is largely front-loaded. This is because the latter time of the class is devoted to developing your data application.**Thus the class should be doable in about a month's time or maybe less. Though make sure you're keeping up with the classes at the beginning so that you have some space in your schedule later on for app development!

If you'd like to keep up with the instructors I'm [@bcaffo](https://twitter.com/bcaffo) on twitter, Roger is [@rdpeng](https://twitter.com/rdpeng) and Jeff is [@jtleek](https://twitter.com/jtleek). The Department of Biostat here is @jhubiostat.

Course Description

A data product is the production output from a statistical analysis. Data products automate complex analysis tasks or use technology to expand the utility of a data informed model, algorithm or inference. This course covers the basics of creating data products using Shiny, R packages, and interactive graphics. The course will focus on the fundamentals of creating a data product that can be used to tell a story about data to a mass audience.

In this class students will learn a variety of core tools for creating data products in R and R Studio in specific. Students will be evaluated via quizzes and a culminating project.

Course Content

The lectures will be taught over four weeks with the third week dedicated to creating R packages.

The weeks are organized as follows

1. Shiny, rCharts, manipulate, googleVis
2. Presenting data analysis, slidify, R Studio presenter.
3. Students creating and deploying their projects
4. Creating R packages, classes and methods, yhat.

Github repository

The most up to date information on the course lecture notes will always be in the Github repository

The data science specialization is here

[https://github.com/bcaffo/courses/tree/master/09\\_DevelopingDataProducts](https://github.com/bcaffo/courses/tree/master/09_DevelopingDataProducts)

Please issue pull requests so that we may improve the materials.

YouTube

If you'd prefer to watch the videos on YouTube, most of them can be found here:

<https://www.youtube.com/playlist?list=PLpl-gQkQivXhr9PyOWSA3aOHf4ZNTrs90>

Book: Developing Data Products in R

This book introduces the topic of Developing Data Products in R. A data product is the ideal output of a Data Science experiment. This book is based on the Coursera Class "Developing Data Products" as part of the Data Science Specialization. Particular emphasis is paid to developing Shiny apps and interactive graphics.

The book is available here: <https://leanpub.com/ddp>

It's variable pricing, including free! It also includes some content (like leaflet) that was not covered in the class and omits some other. It's a little rough, but as I work on it you'll get all of the updates.

Weekly quizzes

* There are three weekly quizzes.
* You must earn a grade of at least 80% to pass a quiz
* You may attempt each quiz up to 3 times in 8 hours.
* The score from your most successful attempt will count toward your final grade.

Course Project

The Course Project is an opportunity to demonstrate the skills you have learned during the course. It is graded through peer assessment. You must earn a grade of at least 80% to pass the course project.

Grading policy

You must score at least 80% on all assignments (Quizzes & Project) to pass the course.

Your final grade will be calculated as follows:

* Quiz 1 = 20%
* Quiz 2 = 20%
* Quiz 3 = 20%
* Course project = 40%

*Developing Data Products in R*

This companion book for the class introduces the topic of Developing Data Products in R. You can get a copy here:[https://leanpub.com/ddp](https://leanpub.com/ddp" \t "_blank)

It has variable pricing, including free!



The book is available under variable pricing, including free! It also includes some content (like leaflet) that was not covered in the class and omits some other. It's a little rough, but as I work on it you'll get all of the updates.

We're excited to announce that we've created a site using GitHub Pages: <http://datasciencespecialization.github.io/>to serve as a directory for content that the community has created. If you've created materials relating to any of the courses in the Data Science Specialization, please send us a pull request so we can add a link to your content on our site. You can find out more about contributing here:[https://github.com/DataScienceSpecialization/DataScienceSpecialization.github.io#contributing](https://github.com/DataScienceSpecialization/DataScienceSpecialization.github.io" \l "contributing" \t "_blank)

We can't wait to see what you've created and where the community can take this site!

**01 : Shiny**

Remember to read the book chapters on Shiny.

Shiny is an important enough topic to devote a lot of time to it. Shiny is another product by RStudio and it is described by RStudio as “A web application framework for R”. They further add “Turn your analyses into interactive web applications No HTML, CSS, or JavaScript knowledge required”. This is mostly true, though a little HTML at least would be useful for understanding some of the concepts. [Here’s](http://www.w3schools.com/html/)  <http://www.w3schools.com/html/> a useful site for learning html basics. We’ll proceed as if your html knowledge is very basic and no more advanced than understanding heading levels for fonts. It is important to dinstiguish between a Shiny applications (app) and a Shiny server. A Shiny server is required to host a shiny app for the world. Otherwise, only those who have have shiny installed and have access to your code could run your web page (really defeating the purpose of making a web page in the first place).

In this class, we won’t cover creating a shiny server, as that requires understanding a little linux server administration. Instead, we’ll run our apps locally and use RStudio’s service for hosting shiny apps (their servers) on a platform called shinyapps.io. In other words, RStudio does the server work for your so that all you need to worry about is building your app. Shinyapps.io is free up to a point in that you can only run 5 apps for a certain amount of time per month. This will be fine for our purposes, but if you’re really going to get into making Shiny apps, you’ll have to spring for a paid plan or run your own server.

**What is Shiny?**

* Shiny is a platform for creating interactive R programs embedded into a web page.
* Suppose that you create a prediction algorithm, with shiny you can *very easily* create web input form that calls R and thus your prediction algorithm and displays the results.
* Using Shiny, the time to create simple, yet powerful, web-based interactive data products in R is minimized.
  + However, it lacks the flexibility of full featured (and more complex) solutions.
* Shiny is made by the fine folks at R Studio.

**Some mild prerequisites**

* Shiny doesn't really require it, but as with all web programming, a little knowledge of html, css and js(java script) is very helpful
  + html gives a web page structure and sectioning as well as markup instructions
  + css gives the style
  + js for interactivity
* There are too many tutorials online to count for getting basic proficiency in these topics to count.
* Shiny uses [bootstrap](http://getbootstrap.com/) http://getbootstrap.com/ (no relation to the statistics bootstrap) style, which (to me) seems to look nice and renders well on mobile platforms

**What else is out there?**

* Creating any solution requiring fairly deep knowledge of web client/server programming
* [OpenCPU](https://public.opencpu.org/) <https://www.opencpu.org/> by Jerome Ooms, is a really neat project providing an API for calling R from web documents
  + And he even hosts an OpenCPU server, but you can create your own

**Context**

* You created a novel prediction algorithm to predict risk for developing diabetes.
  + You're hoping patients and caregivers will be able to enter their data and, if needed, take preventative measures.
* You want to create a web site so that users can input the relevant predictors and obtain their prediction.
* Your prediction algorithm (ok, so you're not going to be saving the world with this one)
  + [link for a real prediction score](http://www.ncbi.nlm.nih.gov/pubmed/12610029) http://www.ncbi.nlm.nih.gov/pubmed/12610029

diabetesRisk <- function(glucose) glucose/200

**Getting started**

* Make sure you have the latest release of R installed
* If on windows, make sure that you have Rtools installed
* install.packages("shiny")
* libray(shiny)
* Great tutorial at <http://rstudio.github.io/shiny/tutorial/>
* Basically, this lecture is walking through that tutorial offering some of our insights
* Note, some of the proposed interactive plotting uses of Shiny could be handled by the very simple manipulate function[rstudio manipulate](http://www.rstudio.com/ide/docs/advanced/manipulate)
* Also, rCharts is will be covered in a different lecture.

**A Shiny project**

* A shiny project is a directory containing at least two parts
  + One named ui.R (for user interface) controls how it looks.
  + One named server.R that controls what it does.

**ui.R**

library(shiny)

shinyUI(pageWithSidebar(

headerPanel("Data science FTW!"),

sidebarPanel(

h3('Sidebar text')

),

mainPanel(

h3('Main Panel text')

)

))

**server.r**

library(shiny)

shinyServer(

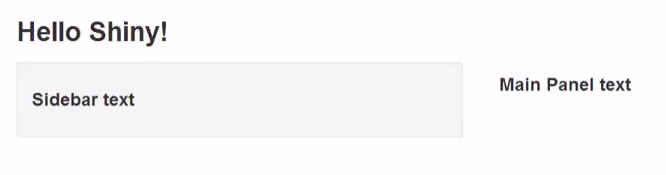
function(input, output) {

}

)

**To run it**

* In R, change to the directories with these files and type runApp()
* or put the path to the directory as an argument
* It should open an browser window with the app running



**R functions for HTML markup**

ui.R

shinyUI(pageWithSidebar(

headerPanel("Illustrating markup"),

sidebarPanel(

h1('Sidebar panel'),

h1('H1 text'),

h2('H2 Text'),

h3('H3 Text'),

h4('H4 Text')

),

mainPanel(

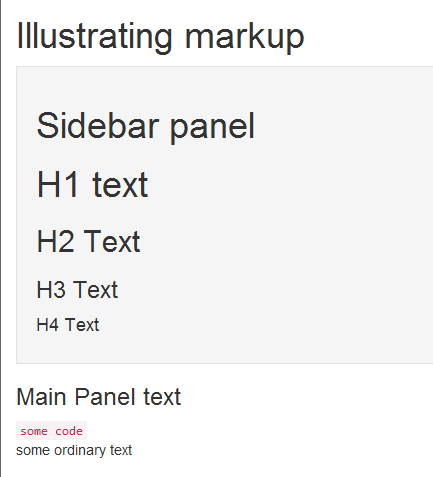
h3('Main Panel text'),

code('some code'),

p('some ordinary text')

)

))



**Illustrating inputs ui.R**

shinyUI(pageWithSidebar(

headerPanel("Illustrating inputs"),

sidebarPanel(

numericInput('id1', 'Numeric input, labeled id1', 0, min = 0, max = 10, step = 1),

checkboxGroupInput("id2", "Checkbox",

c("Value 1" = "1",

"Value 2" = "2",

"Value 3" = "3")),

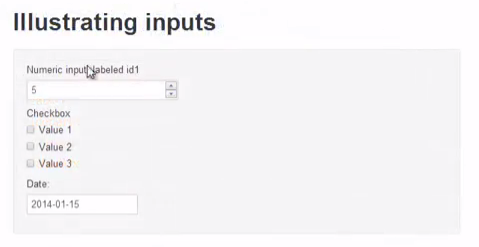
dateInput("date", "Date:")

),

mainPanel(

)

))



**Part of ui.R**

mainPanel(

h3('Illustrating outputs'),

h4('You entered'),

verbatimTextOutput("oid1"),

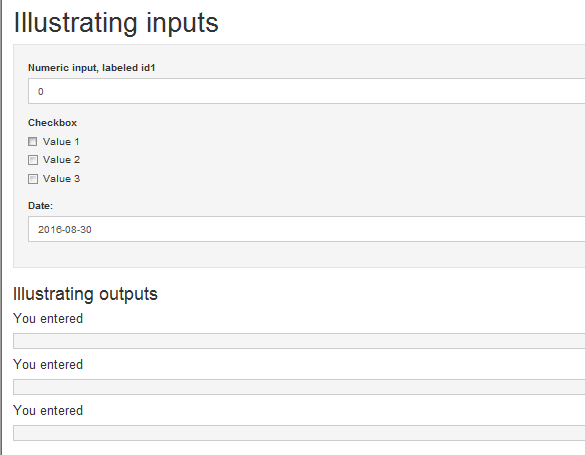
h4('You entered'),

verbatimTextOutput("oid2"),

h4('You entered'),

verbatimTextOutput("odate")

)



**server.R**

shinyServer(

function(input, output) {

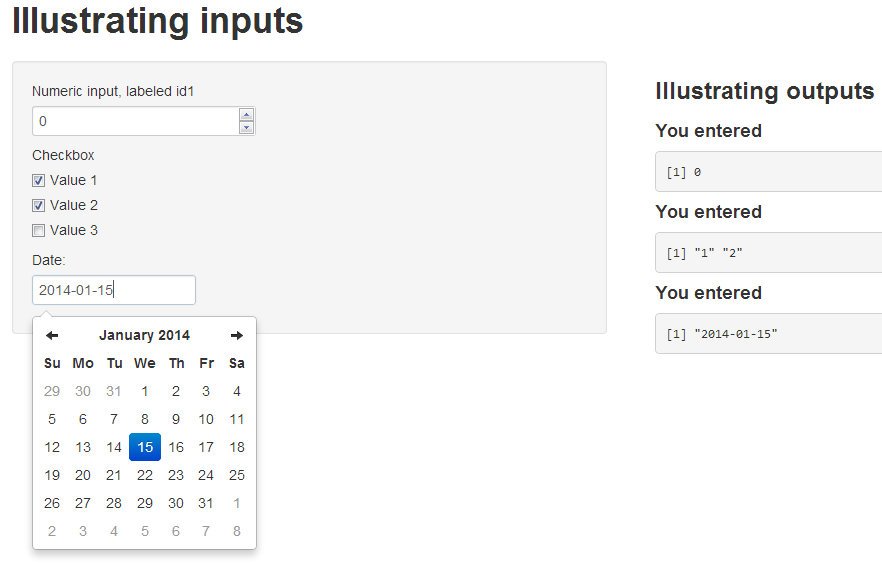
output$oid1 <- renderPrint({input$id1})

output$oid2 <- renderPrint({input$id2})

output$odate <- renderPrint({input$date})

}

)

[](https://github.com/bcaffo/courses/blob/master/09_DevelopingDataProducts/shiny/fig/inputsOutputs.png)

**Let's build our prediction function**

shinyUI(

pageWithSidebar(

# Application title

headerPanel("Diabetes prediction"),

sidebarPanel(

numericInput('glucose', 'Glucose mg/dl', 90, min = 50, max = 200, step = 5),

submitButton('Submit')

),

mainPanel(

h3('Results of prediction'),

h4('You entered'),

verbatimTextOutput("inputValue"),

h4('Which resulted in a prediction of '),

verbatimTextOutput("prediction")

)

)

)

**server.R**

diabetesRisk <- function(glucose) glucose / 200

shinyServer(

function(input, output) {

output$inputValue <- renderPrint({input$glucose})

output$prediction <- renderPrint({diabetesRisk(input$glucose)})

}

)

**The result**

[](https://github.com/bcaffo/courses/blob/master/09_DevelopingDataProducts/shiny/fig/predictionApp.png)

**Image example**

* Let's build an example with an image
* How about we create a histogram of data
* Put a slider on so that the user has to guess the mean

**ui.R**

shinyUI(pageWithSidebar(

headerPanel("Example plot"),

sidebarPanel(

sliderInput('mu', 'Guess at the mean',value = 70, min = 62, max = 74, step = 0.05,)

),

mainPanel(

plotOutput('newHist')

)

))

**server.R**

library(UsingR)

data(galton)

shinyServer(

function(input, output) {

output$newHist <- renderPlot({

hist(galton$child, xlab='child height', col='lightblue',main='Histogram')

mu <- input$mu

lines(c(mu, mu), c(0, 200),col="red",lwd=5)

mse <- mean((galton$child - mu)^2)

text(63, 150, paste("mu = ", mu))

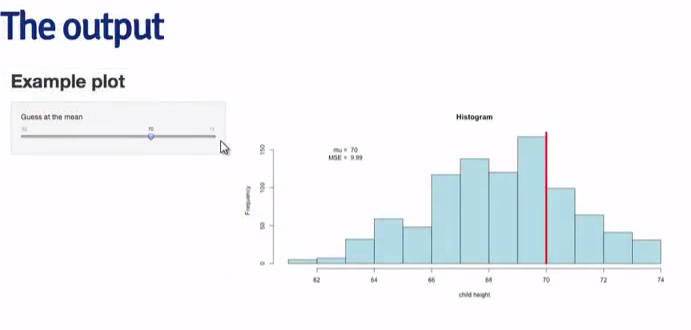
text(63, 140, paste("MSE = ", round(mse, 2)))

})

}

)

**The output**



**Tighter control over style**

* All of the style elements are handled through ui.R
* Instead, you can create a www directory and then an index.html file in that directory
  + [This link](http://rstudio.github.io/shiny/tutorial/#html-ui) <http://shiny.rstudio.com/> goes through the html needed
  + You just have to have specific js libraries and appropriately name ids and classes. This is beyond the scope of this class
  + For students with a lot of experience in html, js, css it would be a breeze and probably easier and more flexible than the R html controls in ui.R

**Other things Shiny can do**

* Allow users to upload or download files
* Have tabbed main panels
* Have editable data tables
* Have a dynamic UI
* User defined inputs and outputs
* Put a submit button so that Shiny only executes complex code after user hits submit

**Distributing a Shiny app**

* The quickest way is to send (or put on github or gist or dropbox or whatever) someone the app directory and they can then call runApp
* You could create an R package and create a wrapper that calls runApp  
  + Of course, these solutions only work if the user knows R
* Another option is to run a shiny server
  + Requires setting up a (Shiny server)[<http://www.rstudio.com/shiny/server/>]
    - Probably easiest if you use one of the virtual machines where they already have Shiny servers running well (for example, on AWS)
  + Setting up a Shiny server is beyond the scope of this class as it involves some amount of linux server administration
  + Groups are creating a Shiny hosting services that will presumably eventually be a fee for service or freemium service
  + BTW, don't put system calls in your code (this is one of the first things many of us do for fun, but it introduces security concerns)

Some people in this session let us know that they are concerned about running up against the 25-hour per month limit on the free tier of [shinyapps.io](https://www.rstudio.com/products/shinyapps/).

Should you hit the limit on the free plan, RStudio will send you a message. If you receive the message and are more than a few days from getting a fresh 25 hours on your monthly renewal, please send an email to shinyapps-support@rstudio.com with the email address you use on the service and the account name you are using (the first part of the URL). RStudio will then increase your limit so you can continue working on your project.

Since there are a lot of folks in the class we'd appreciate if you only emailed RStudio after you get the message and only if you feel you'll need more time.

www.shinyapps.io/coursera/

<https://www.shinyapps.io/admin/#/signup?referral=coursera>

Remember to read the book chapter on Shiny (called Shiny Part II)

In this module, we cover some of the less basic aspects of Shiny and use experiments to figure out what it's doing.

## 02 : More advanced Shiny

## Shiny revisited

* In the last lecture, we covered basic creation of Shiny applications
* If you tried it and are like most, you had an easy time with ui.R but a harder time with server.R
* In this lecture, we cover some more of the details of shiny
* Since writing the last lecture, a more detailed tutorial has been created that is worth checking out (<http://shiny.rstudio.com/tutorial/>)

## Details

* Code that you put before shinyServer in the server.R function gets called once when you do runApp()
* Code inside the unnamed function of shinyServer(function(input, output){ but not in a reactive statement will run once for every new user (or page refresh)
* Code in reactive functions of shinyServer get run repeatedly as needed when new values are entered (reactive functions are those like render\*)

### Experiment (code in the slidify document)

ui.R

shinyUI(pageWithSidebar(

headerPanel("Hello Shiny!"),

sidebarPanel(

textInput(inputId="text1", label = "Input Text1"),

textInput(inputId="text2", label = "Input Text2")

),

mainPanel(

p('Output text1'),

textOutput('text1'),

p('Output text2'),

textOutput('text2'),

p('Output text3'),

textOutput('text3'),

p('Outside text'),

textOutput('text4'),

p('Inside text, but non-reactive'),

textOutput('text5')

)

))

server.R Set x <- 0 before running

library(shiny)

x <<- x + 1

y <<- 0

shinyServer(

function(input, output) {

y <<- y + 1

output$text1 <- renderText({input$text1})

output$text2 <- renderText({input$text2})

output$text3 <- renderText({as.numeric(input$text1)+1})

output$text4 <- renderText(y)

output$text5 <- renderText(x)

}

)

## Try it

* type runApp()
* Notice hitting refresh incriments y but enterting values in the textbox does not
* Notice x is always 1
* Watch how it updated text1 and text2 as needed.
* Doesn't add 1 to text1 every time a new text2 is input.
* Important try runApp(display.mode='showcase')

## Reactive expressions

* Sometimes to speed up your app, you want reactive operations (those operations that depend on widget input values) to be performed outside of a render\*1 statement
* For example, you want to do some code that gets reused in several render\* statements and don't want to recalculate it for each
* The reactive function is made for this purpose

## Example

server.R

shinyServer(

function(input, output) {

x <- reactive({as.numeric(input$text1)+100})

output$text1 <- renderText({x() })

output$text2 <- renderText({x() + as.numeric(input$text2)})

}

)

## As opposed to

shinyServer(

function(input, output) {

output$text1 <- renderText({as.numeric(input$text1)+100 })

output$text2 <- renderText({as.numeric(input$text1)+100 +

as.numeric(input$text2)})

}

)

## Discussion

* Do runApp(display.mode='showcase')
* (While inconsequential) the second example has to add 100 twice every time text1 is updated for the second set of code
* Also note the somewhat odd syntax for reactive variables

## Non-reactive reactivity (what?)

* Sometimes you don't want shiny to immediately perform reactive calculations from widget inputs
* In other words, you want something like a submit button

## ui.R

shinyUI(pageWithSidebar(

headerPanel("Hello Shiny!"),

sidebarPanel(

textInput(inputId="text1", label = "Input Text1"),

textInput(inputId="text2", label = "Input Text2"),

actionButton("goButton", "Go!")

),

mainPanel(

p('Output text1'),

textOutput('text1'),

p('Output text2'),

textOutput('text2'),

p('Output text3'),

textOutput('text3')

)

))

## Server.R

shinyServer(

function(input, output) {

output$text1 <- renderText({input$text1})

output$text2 <- renderText({input$text2})

output$text3 <- renderText({

input$goButton

isolate(paste(input$text1, input$text2))

})

}

)

## Try it out

* Notice it doesn't display output text3 until the go button is pressed
* input$goButton (or whatever you named it) gets increased by one for every time pushed
* So, when in reactive code (such as render or reactive) you can use conditional statements like below to only execute code on the first button press or to not execute code until the first or subsequent button press

if (input$goButton == 1){ Conditional statements }

## Example

Here's some replaced code from our previous server.R

output$text3 <- renderText({

if (input$goButton == 0) "You have not pressed the button"

else if (input$goButton == 1) "you pressed it once"

else "OK quit pressing it"

})

## More on layouts

* The sidebar layout with a main panel is the easiest.
* Using shinyUI(fluidpage( is much more flexible and allows tighter access to the bootstrap styles
* Examples here (<http://shiny.rstudio.com/articles/layout-guide.html>)
* fluidRow statements create rows and then the column function from within it can create columns
* Tabsets, navlists and navbars can be created for more complex apps

### Directly using html

* For more complex layouts, direct use of html is preferred (<http://shiny.rstudio.com/articles/html-ui.html>)
* Also, if you know web development well, you might find using R to create web layouts kind of annoying
* Create a directory called www in the same directory with server.R
* Have an index.html page in that directory
* Your named input variables will be passed to server.R <input type="number" name="n" value="500" min="1" max="1000" />
* Your server.R output will have class definitions of the form shiny- <pre id="summary" class="shiny-text-output"></pre>

## Debugging techniques for Shiny

* Debugging shiny apps can be tricky
* We saw that runApp(displayMode = 'showcase') highlights execution while a shiny app runs
* Using cat in your code displays output to stdout (so R console)
* The browser() function can interupt execution and can be called conditionally (<http://shiny.rstudio.com/articles/debugging.html>)

**03: Manipulate**

Remember to read the book chapter on Manipulate.

Suppose that you want to create a quick interactive graphic and you have to do it now. You’re not concerned about accessibility to your interactive graph, you just need it for you or others who also use RStudio. The wonderful little R package manipulate is for you. Manipulate is an R package created by RStudio and must be used within that development environment to work. It is described in very good detail [here](http://www.rstudio.com/ide/docs/advanced/manipulate). <https://support.rstudio.com/hc/en-us/articles/200551906-Interactive-Plotting-with-Manipulate> It offers simple controls for graphics. So, you’re not going to win any visualization awards for your manipulate output, but it will solve your problem quickly.

**Manipulate**

* Suppose that you want to create a quick interactive graphic
  + You have to do it *now*
  + The intended users also use Rstudio
* manipulate is a really cool solution that is often all you need to quickly make interactive graphics

**Documentation**

* Manipulate is well documented at the Rstudio web site here
  + <http://www.rstudio.com/ide/docs/advanced/manipulate>
* From there, try this

library(manipulate)

manipulate(plot(1:x), x = slider(1, 100))

* You can create a slider, checkbox, or picker (drop down) and have more than one

**Example from the regression class**

library(manipulate)

myHist <- function(mu){

hist(galton$child,col="blue",breaks=100)

lines(c(mu, mu), c(0, 150),col="red",lwd=5)

mse <- mean((galton$child - mu)^2)

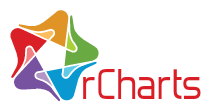
text(63, 150, paste("mu = ", mu))

text(63, 140, paste("MSE = ", round(mse, 2)))

}

manipulate(myHist(mu), mu = slider(62, 74, step = 0.5))

**04: rCharts**

[](https://github.com/bcaffo/courses/blob/master/09_DevelopingDataProducts/rCharts/fig/rCharts.png)

* rCharts is a way to create interactive javascript visualizations using R
* So
  + You don't have to learn complex tools, like D3
  + You simply work in R learning a minimal amount of new syntaxt
* rCharts was written by Ramnath Vaidyanathan (friend of the Data Science Series), who also wrote slidify, the framework we use for all of the lectures in the class
* This lecture is basically going through (<http://ramnathv.github.io/rCharts/>)

**Example**

require(rCharts)

haireye = as.data.frame(HairEyeColor)

n1 <- nPlot(Freq ~ Hair, group = 'Eye', type = 'multiBarChart',

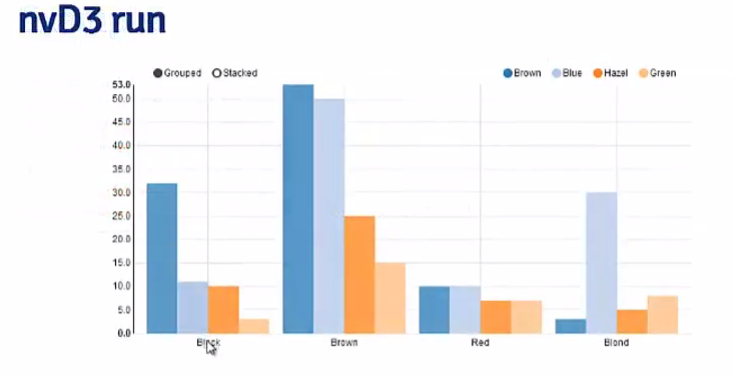
data = subset(haireye, Sex == 'Male')

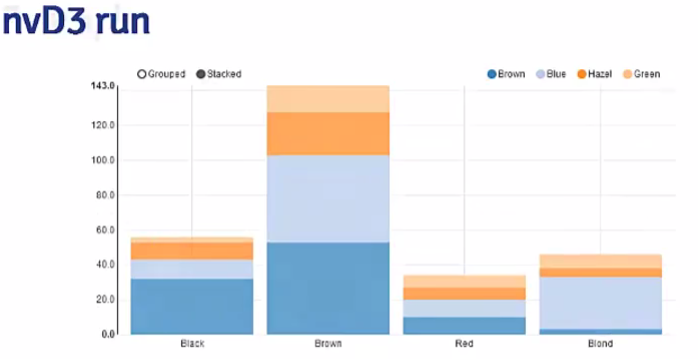
)

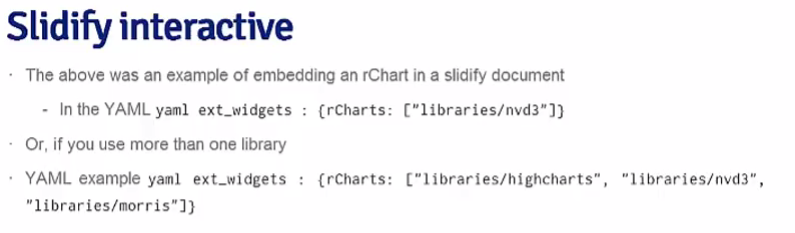
n1$save('fig/n1.html', cdn = TRUE)

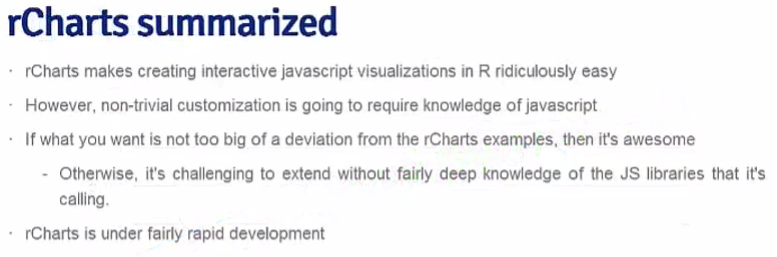
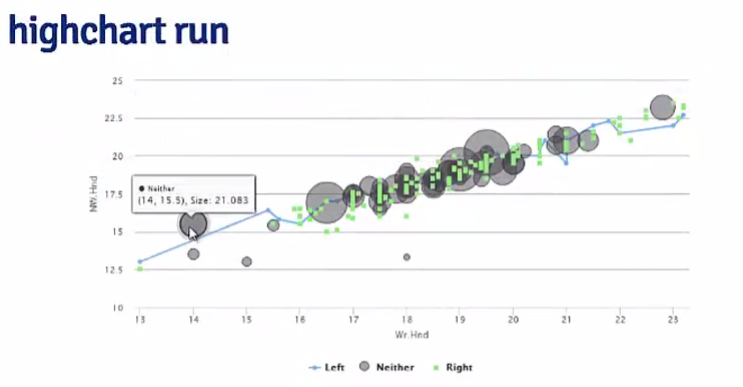
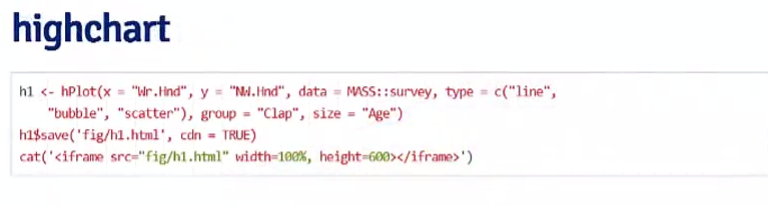
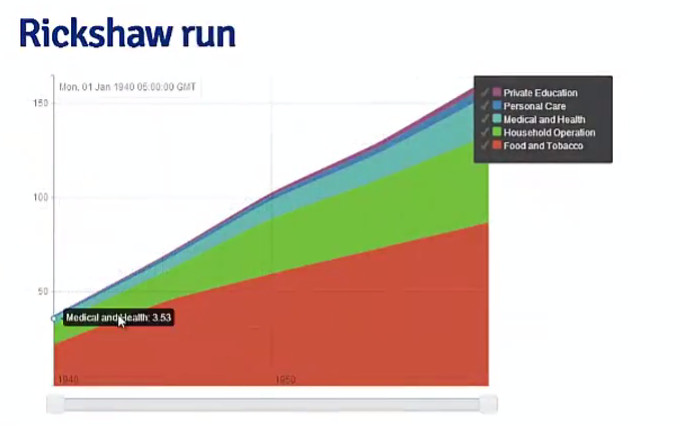
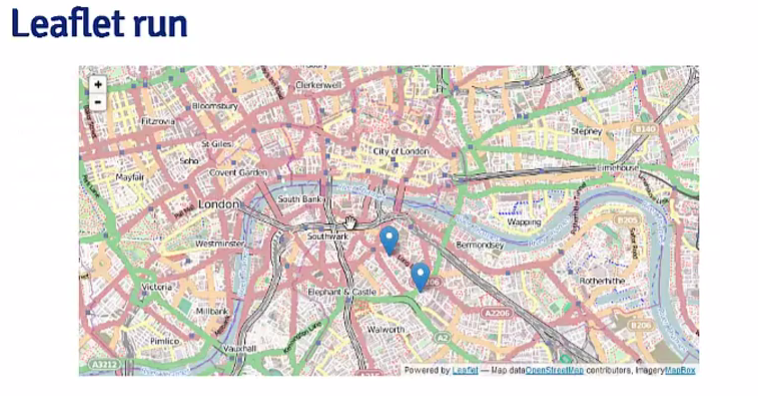
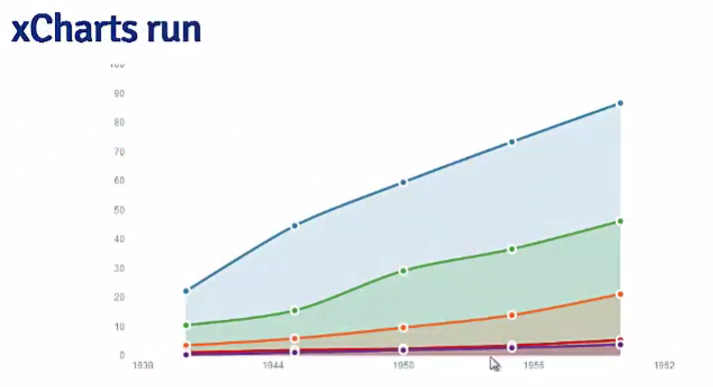
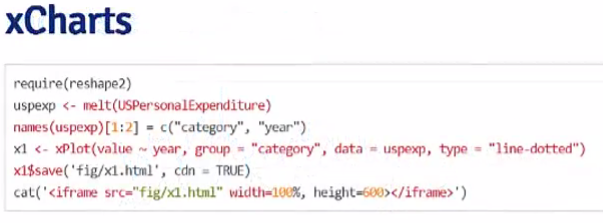
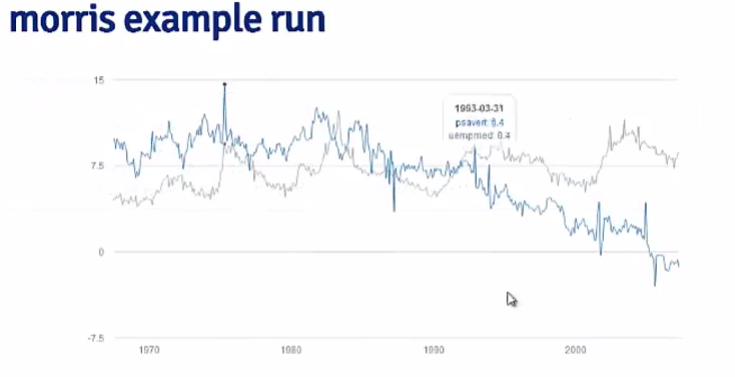
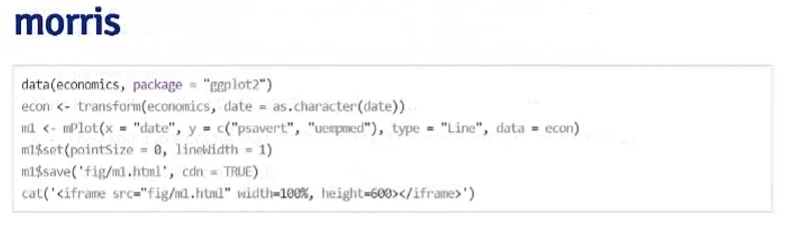
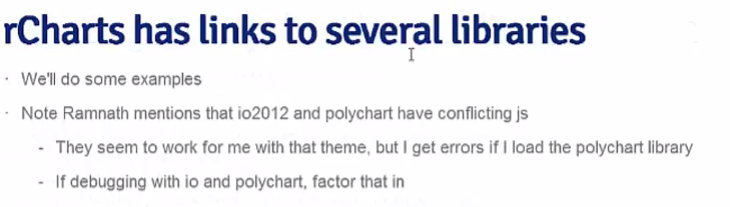
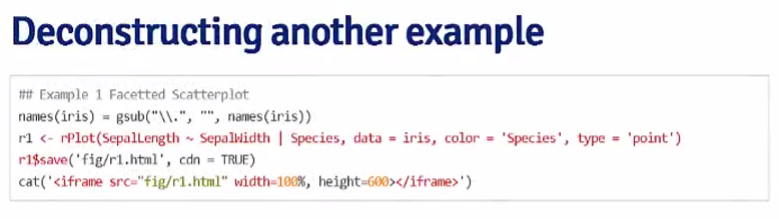
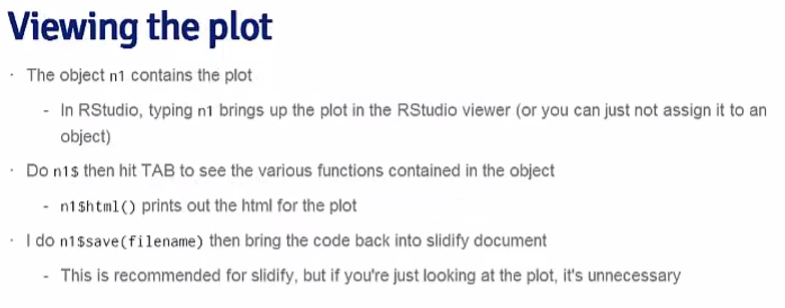
cat('<iframe src="fig/n1.html" width=100%, height=600></iframe>')

**nvD3 run**



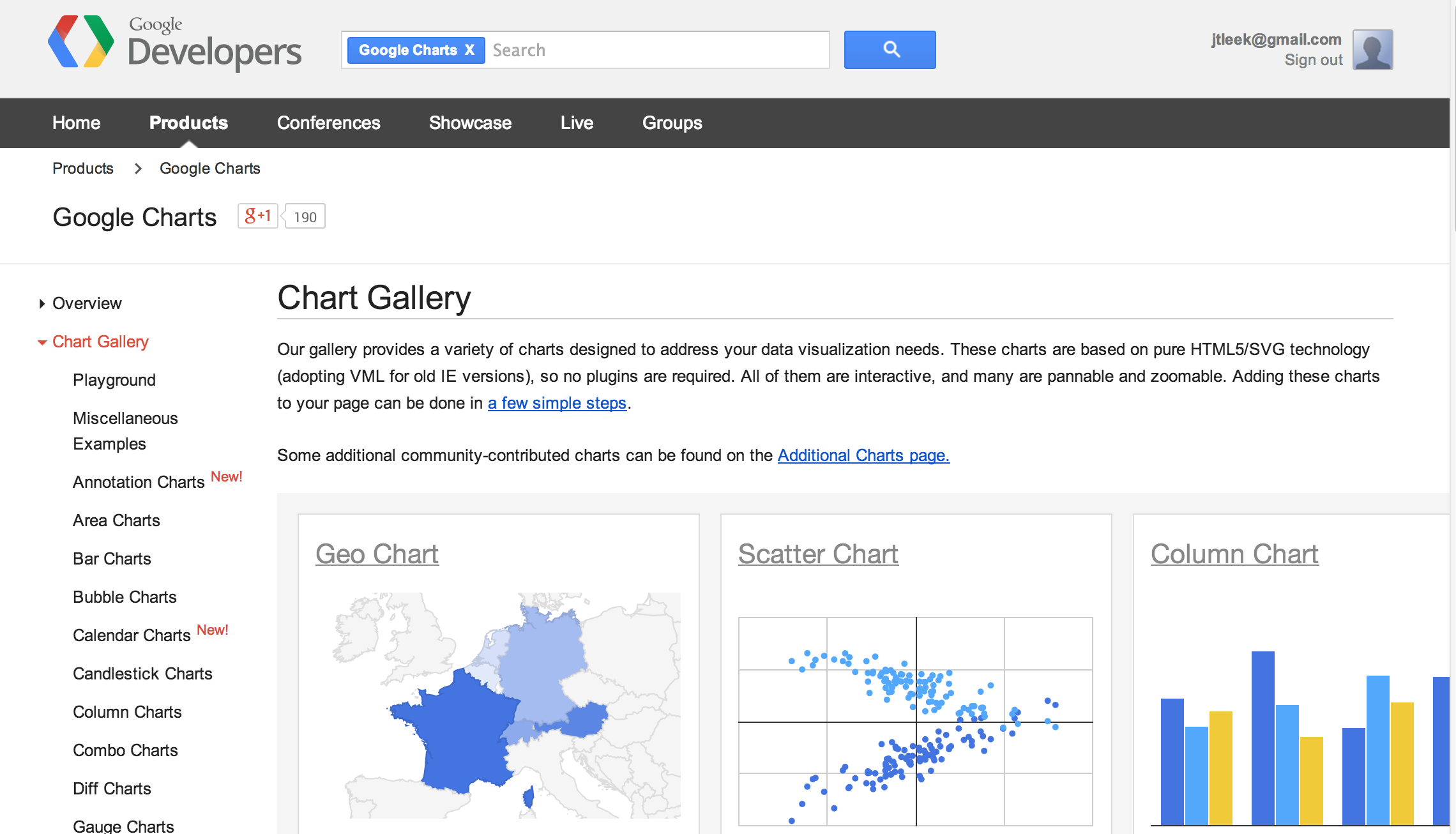






**05: GoogleVis**

**Google Vis API**

[](https://github.com/bcaffo/courses/blob/master/assets/img/googlecharts.png)

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

**Basic idea**

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

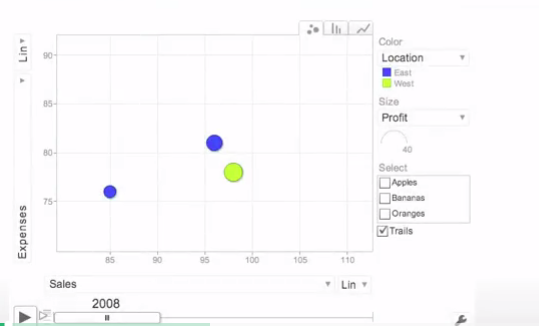
**Example**

suppressPackageStartupMessages(library(googleVis))

## Warning: package 'googleVis' was built under R version 3.0.3

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

print(M, "chart")



**Charts in googleVis**

"gvis + ChartType"

* Motion charts: gvisMotionChart
* Interactive maps: gvisGeoChart
* Interactive tables: gvisTable
* Line charts: gvisLineChart
* Bar charts: gvisColumnChart
* Tree maps: gvisTreeMap

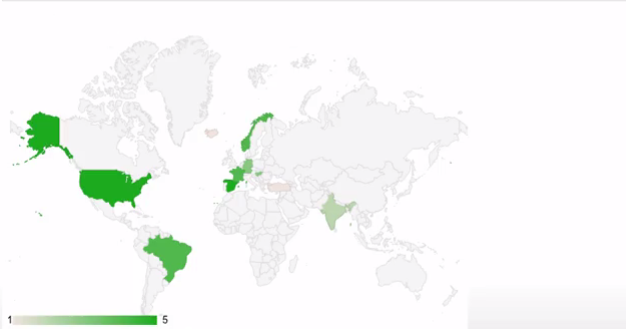
<http://cran.r-project.org/web/packages/googleVis/googleVis.pdf>

**Plots on maps**

G <- gvisGeoChart(Exports, locationvar = "Country", colorvar = "Profit", options = list(width = 600,

height = 400))

print(G, "chart")

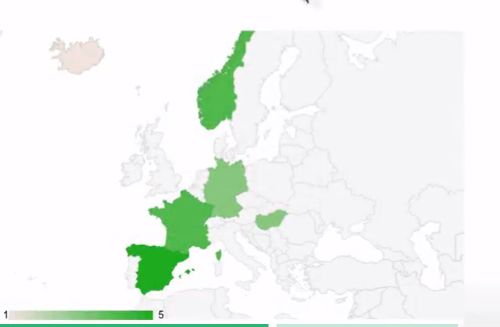


**Specifying a region**

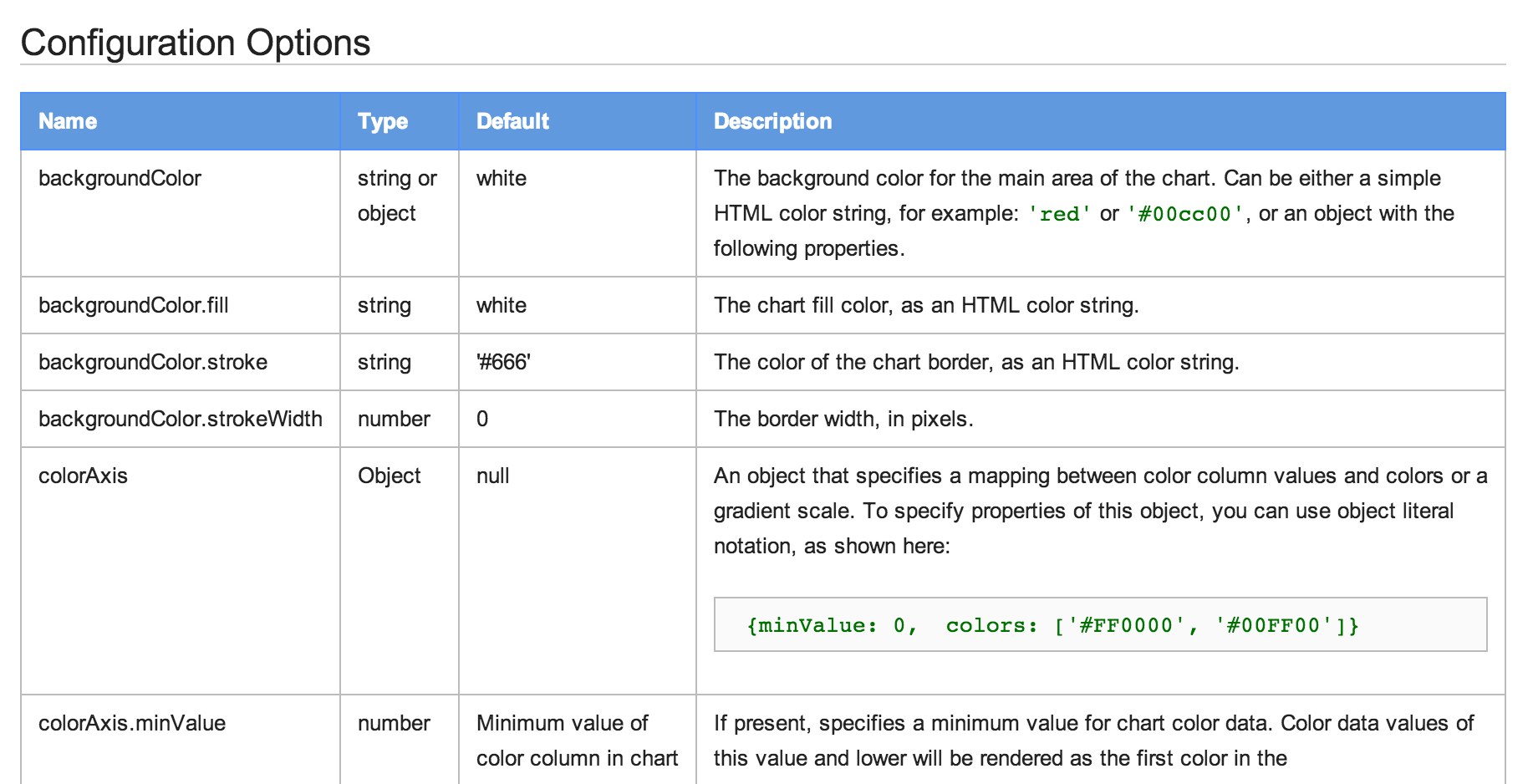
G2 <- gvisGeoChart(Exports, locationvar = "Country", colorvar = "Profit", options = list(width = 600,

height = 400, region = "150"))

print(G2, "chart")



**Finding parameters to set under options**

[](https://github.com/bcaffo/courses/blob/master/assets/img/configoptions.png)

<https://developers.google.com/chart/interactive/docs/gallery/geochart>

**Setting more options**

df <- data.frame(label=c("US", "GB", "BR"), val1=c(1,3,4), val2=c(23,12,32))

Line <- gvisLineChart(df, xvar="label", yvar=c("val1","val2"),

options=list(title="Hello World", legend="bottom",

titleTextStyle="{color:'red', fontSize:18}",

vAxis="{gridlines:{color:'red', count:3}}",

hAxis="{title:'My Label', titleTextStyle:{color:'blue'}}",

series="[{color:'green', targetAxisIndex: 0},

{color: 'blue',targetAxisIndex:1}]",

vAxes="[{title:'Value 1 (%)', format:'##,######%'},

{title:'Value 2 (\U00A3)'}]",

curveType="function", width=500, height=300

))

<https://github.com/mages/Introduction_to_googleVis/blob/gh-pages/index.Rmd>

**Setting more options**

print(Line, "chart")



**Combining multiple plots together**

G <- gvisGeoChart(Exports, "Country", "Profit",options=list(width=200, height=100))

T1 <- gvisTable(Exports,options=list(width=200, height=270))

M <- gvisMotionChart(Fruits, "Fruit", "Year", options=list(width=400, height=370))

GT <- gvisMerge(G,T1, horizontal=FALSE)

GTM <- gvisMerge(GT, M, horizontal=TRUE,tableOptions="bgcolor=\"#CCCCCC\" cellspacing=10")

**Combining multiple plots together**

print(GTM, "chart") 

**Seeing the HTML code**

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

print(M)

## <!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>MotionChartID23187d102a5b</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;

## }

## </style>

## </head>

## <body>

## <!-- MotionChart generated in R 3.0.2 by googleVis 0.5.2 package -->

## <!-- Thu May 08 16:19:09 2014 -->

##

##

## <!-- jsHeader -->

## <script type="text/javascript">

##

## // jsData

## function gvisDataMotionChartID23187d102a5b () {

## var data = new google.visualization.DataTable();

## var datajson =

## [

## [

## "Apples",

## 2008,

## "West",

## 98,

## 78,

## 20,

## "2008-12-31"

## ],

## [

## "Apples",

## 2009,

## "West",

## 111,

## 79,

## 32,

## "2009-12-31"

## ],

## [

## "Apples",

## 2010,

## "West",

## 89,

## 76,

## 13,

## "2010-12-31"

## ],

## [

## "Oranges",

## 2008,

## "East",

## 96,

## 81,

## 15,

## "2008-12-31"

## ],

## [

## "Bananas",

## 2008,

## "East",

## 85,

## 76,

## 9,

## "2008-12-31"

## ],

## [

## "Oranges",

## 2009,

## "East",

## 93,

## 80,

## 13,

## "2009-12-31"

## ],

## [

## "Bananas",

## 2009,

## "East",

## 94,

## 78,

## 16,

## "2009-12-31"

## ],

## [

## "Oranges",

## 2010,

## "East",

## 98,

## 91,

## 7,

## "2010-12-31"

## ],

## [

## "Bananas",

## 2010,

## "East",

## 81,

## 71,

## 10,

## "2010-12-31"

## ]

## ];

## data.addColumn('string','Fruit');

## data.addColumn('number','Year');

## data.addColumn('string','Location');

## data.addColumn('number','Sales');

## data.addColumn('number','Expenses');

## data.addColumn('number','Profit');

## data.addColumn('string','Date');

## data.addRows(datajson);

## return(data);

## }

##

## // jsDrawChart

## function drawChartMotionChartID23187d102a5b() {

## var data = gvisDataMotionChartID23187d102a5b();

## var options = {};

## options["width"] = 600;

## options["height"] = 400;

##

##

## var chart = new google.visualization.MotionChart(

## document.getElementById('MotionChartID23187d102a5b')

## );

## chart.draw(data,options);

##

##

## }

##

##

## // jsDisplayChart

## (function() {

## var pkgs = window.\_\_gvisPackages = window.\_\_gvisPackages || [];

## var callbacks = window.\_\_gvisCallbacks = window.\_\_gvisCallbacks || [];

## var chartid = "motionchart";

##

## // 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(drawChartMotionChartID23187d102a5b);

## })();

## function displayChartMotionChartID23187d102a5b() {

## 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=displayChartMotionChartID23187d102a5b"></script>

##

## <!-- divChart -->

##

## <div id="MotionChartID23187d102a5b"

## style="width: 600px; height: 400px;">

## </div>

## <div><span>Data: Fruits &#8226; Chart ID: <a href="Chart\_MotionChartID23187d102a5b.html">MotionChartID23187d102a5b</a> &#8226; <a href="https://github.com/mages/googleVis">googleVis-0.5.2</a></span><br />

## <!-- htmlFooter -->

## <span>

## R version 3.0.2 (2013-09-25)

## &#8226; <a href="https://developers.google.com/terms/">Google Terms of Use</a> &#8226; <a href="https://google-developers.appspot.com/chart/interactive/docs/gallery/motionchart">Documentation and Data Policy</a>

## </span></div>

## </body>

## </html>

print(M, "chart", file = "myfilename.html")

**Things you can do with Google Vis**

* The visualizations can be embedded in websites with HTML code
* Dynamic visualizations can be built with Shiny, Rook, and R.rsp
* Embed them in [R markdown](http://www.rstudio.com/ide/docs/authoring/using_markdown) based documents
  + Set results="asis" in the chunk options
  + Can be used with [knitr](http://cran.r-project.org/web/packages/knitr/index.html) and [slidify](http://slidify.org/)

**For more info**

Library(gooleVis)

demo(googleVis)

* <http://cran.r-project.org/web/packages/googleVis/vignettes/googleVis.pdf>
* <http://cran.r-project.org/web/packages/googleVis/googleVis.pdf>
* <https://developers.google.com/chart/interactive/docs/gallery>
* <https://developers.google.com/chart/interactive/faq>

**05 : plotly**

load("courseraData.rda")

## Make sure that you've followed the first few set up steps

## https://plot.ly/ggplot2/getting-started/

## Particularly set\_credentials\_file(username=FILL IN, api\_key=FILL IN)

library(plotly)

library(ggplot2)

## First do a bar plot in ggplot

g <- ggplot(myData, aes(y = enrollment, x = class, fill = offering))

g <- g + geom\_bar(stat = "identity")

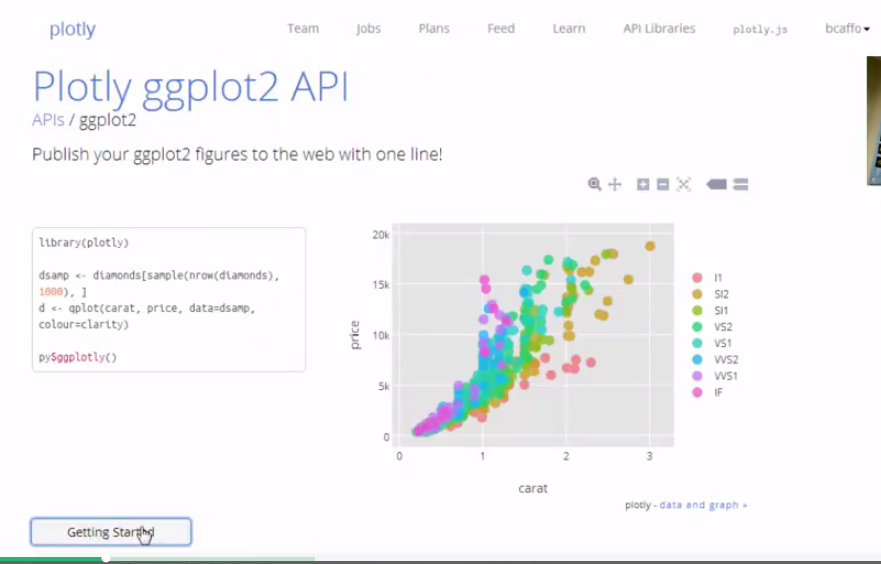
g

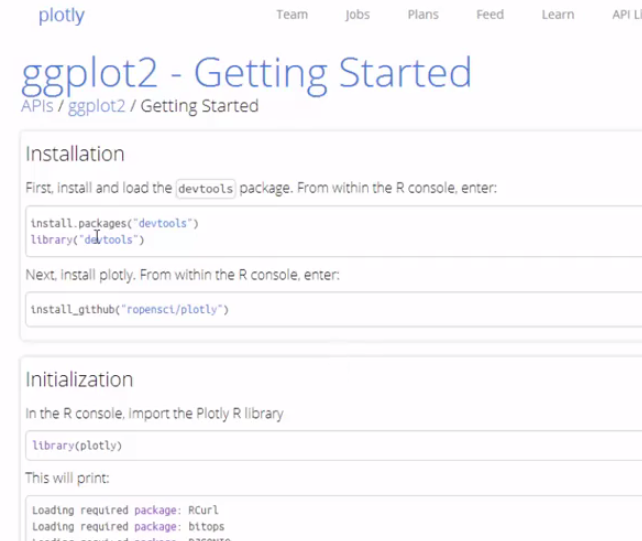
## Let's try to get it into plot.ly

py <- plotly()

out <- py$ggplotly(g)

out$response$url





# https://plot.ly/r/getting-started/

# Getting Started with Plotly for R

Plotly is an R package for creating interactive web-based graphs via the open source JavaScript graphing library [plotly.js](http://plot.ly/javascript). As of version 2.0 (November 17, 2015), Plotly graphs are rendered locally through the [htmlwidgets](http://www.htmlwidgets.org/) framework.

Installation

Plotly is now on CRAN!

install.packages("plotly")

RStudio users should download the latest [RStudio release](https://www.rstudio.com/products/rstudio/download/) for compatibility with htmlwidgets.

Simple example

library(plotly)

p <- plot\_ly(midwest, x = percollege, color = state, type = "box")

p



**06 : Interactive Graphics**

Remember to read the book chapter on interactive graphics.

R has an increasingly diverse set of tools for creating interactive graphs. You’ve already seen some with manipulate and shiny. These tools, however, rely on running R in the background. Other tools convert the graphs into javascript displays that can be embedded into web pages and html presentations.