Developing Data Products Notes

Coursera Course by John Hopkins University

INSTRUCTORS: Dr. Brian Caffo, Dr. Jeff Leek, Dr. Roger D. Peng

Contents

Intro

Github Link for Lectures

Developing Data Products' lectures on GitHub

Course Book

The book for this course is available on leanpub

Instructor's Note

"*This course is about building tools for improving the data analysis process, making data driven decisions, or for other infrastructure that supports other data products...*
To preview some of the topics we'll be covering. With the R Markdown package you can create websites, PDFs, presentations, and even e-books from a single file in R in a way that you're very comfortable and familiar with. We'll teach the latest features of the Shiny package which you can use to create interactive web applications in R. We'll talk about interactive graphics using Plotly and Leaflet which allows you to create beautiful maps that you can share online. We'll do similar things with the package GoogleViz which allows you to create maps and interactive graphics and tables. We've also added how to use swirl and swirlify to design courses in R so that you can share your knowledge...

- *Brian Caffo and the Data Science Track Team*"

Shiny, GoogleVis, and Plotly

Shiny Part 1

Shiny Overview (1.1)

What is Shiny?

- Shiny is a web development framework in R, meaning one only needs to know R to use it.
  - Helps one get around a possible lack of resources or knowledge in JavaScipt, HTML, etc.
- Shiny needs a server to run on
  - One can use their own, RStudio's limited free hosting service, or something like Amazon AWS. RStudio also has a paid version of their service.

- Shiny application vs. Shiny server
  - Apps a ran locally and use RStudio's service for hosting the app on their servers, on a platform called shinyapps.io
    * the free version only allows one to run 5 apps for a certain amount of time per month

    * RStudio will send one a message if the limit is reached

        * Should one hit the 25-hour per month limit they can send an email to *shinyapps-support@rstudio.com* to request an increase in their limit so one can continue working on thier project (for this course)

    - A Shiny server is required such that one can host a shiny app for the world
        * It requires understanding a little linux server administration and won't be covered in this course.
- Although everything is done in R it is helpful to have some knowledge of HTML to know what the commands in R are actually doing.

- Shiny uses `Bootstrap` (no relation to the statistical method) style, which suffices for aesthetics, rendering, and resizing to fit different screens.

HTML, CSS, and Javascript Tutorials

- **Mozilla Developer Network Tutorials**

- **HTML & CSS from Khan Academy**

- **Tutorials from Free Code Camp**

A Shiny Project

- There is **a tutorial for shiny on RStudio**, however these notes will sort of walk through that tutorial anyway.

- A computer interfaces with the app to generate new plots/results based on the users' input on the app.

- A shiny project consists of a directory with at least two files:
    - `ui.R` (for user interface) controls how your app looks.

    - `server.R` that controls what your app does, held on the server.

    - NOTE: The app doesn't actually need these two files but rather the functions the contain, as such it's also an option to just have a `app.R` file.

- **RStudio has some examples of shiny apps too**
- I also found someone **made a game using Shiny**. The game's app also contains **a link to the GitHub repo.**

Shiny Code Demos (1.2)

A demo is located in `./helloShiny`

Set-Up

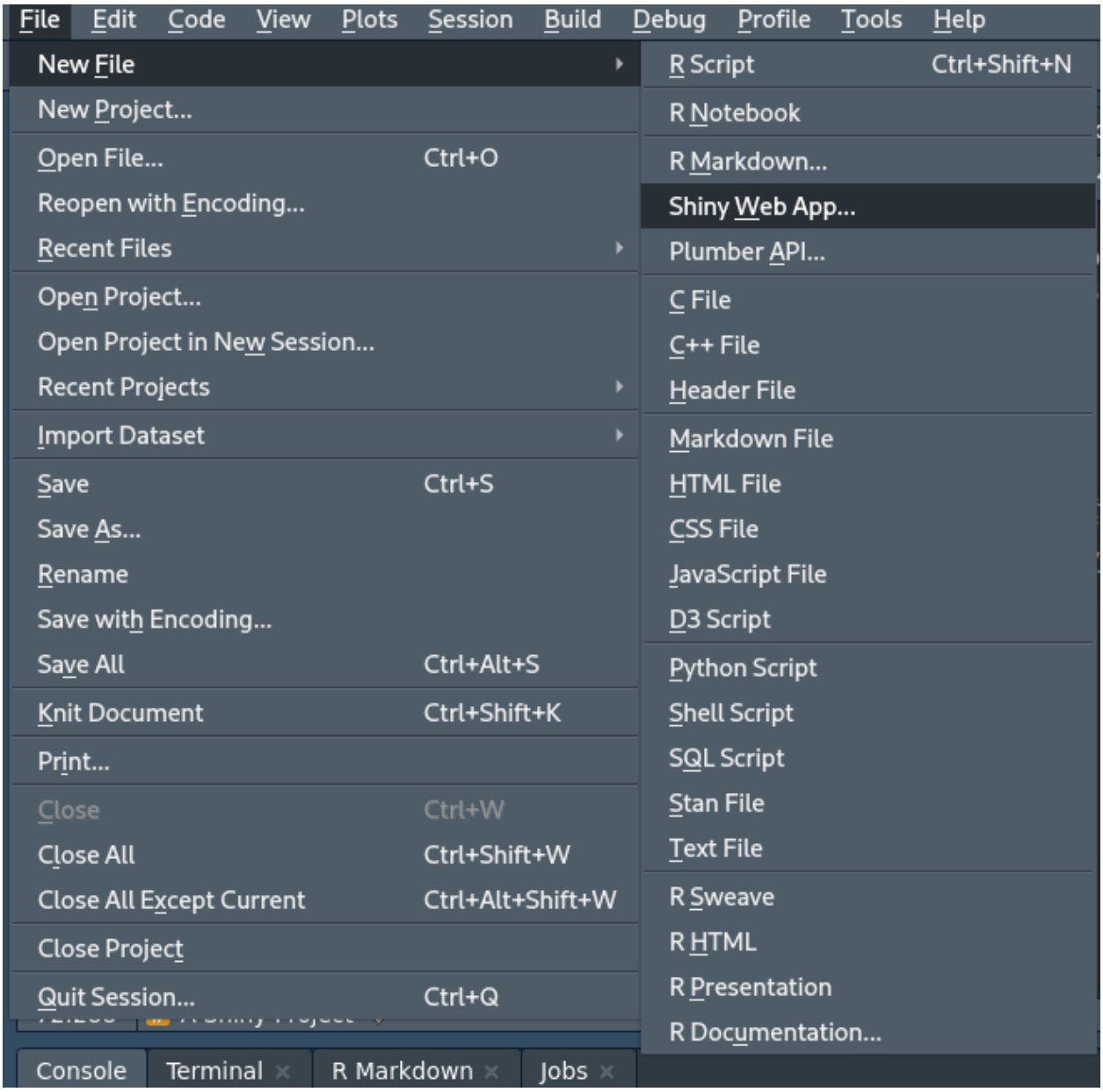

Figure 1: New File

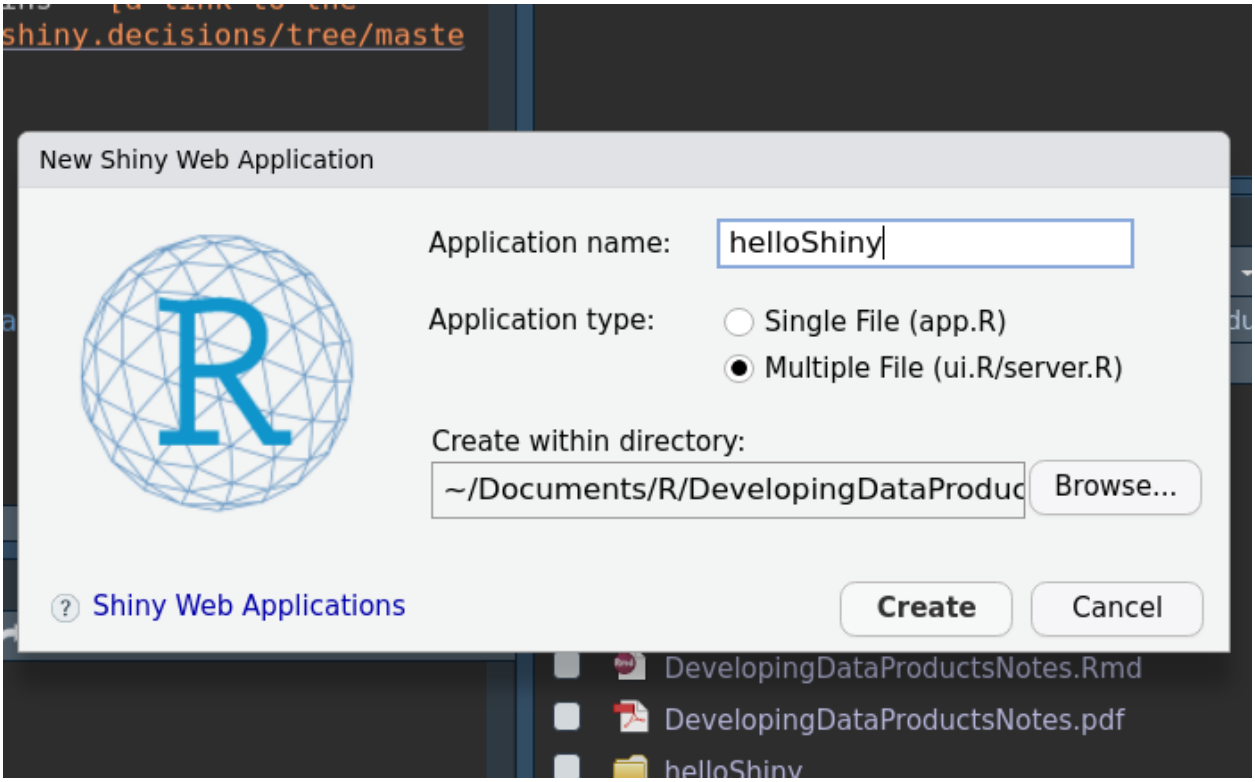

Figure 2: Create app in GUI

ui.R

- requires the function `shinyUI`
  - `fluidPage(` by default has a main panel, `mainPanel(` and a sidebar panel, `sidebarLayout(sidebarPanel(`

server.R

- requires the function `shinyServer`
  - Any logical computations are done here

Shiny HTML Tags (1.3)

- Shiny provides several wrapper functions for using standard HTML tags in your `ui.R`, including:
  - `h1()` through `h6()` for headlines
  - `p()` for paragraphs
  - `br()` for line-breaks
  - `a()` for inserting hyperlinks
  - `div()` denotes a section styled with CSS

  - `span()` to color a part of the text
  - See the help page, `?builder` for more details.
  - A guide of HTML tags **can be found on this site**
- Some of these tags are tested in `./helloShiny/`

Shiny Apps with I/O (1.4)

- In this first demo we'll be looking at a slider input, which will just show the value to the user. This demo is found in `./helloSlider/`

Shiny Apps with Plots (1.5)

- Allowing users to manipulate data nd see the reults of their manipulations as a plot can be very useful
- Shiny provides the `plotOutput()` function for `ui.R`
- ...and the `renderPlot()` function for `sever.R`
- The demo of this is in `./helloPlot/`
- Also contains:
  - in/de-crementing arrows for a textbox
  - checkboxes for a logical if labels are present

Reminder to Commit (01), Delete this line *AFTER* Committing

Shiny Part 2

Reactivity (2.1)

- A reactive expression manipulates inputs from Shiny and returns a value.
- Reactivity provides a way for your app to respond, as inputs will change depending on how users interact with your UI
- Expressions that are subject to change should be wrapped with the function `reactive()`
  - Because of the syntax of Shiny this may look a bit "unorthodox" for R code (Having `({...})`)
- Creating a reactive expression is like creating a function:

```
calc_sum <- reactive({
  input$box1 + input$box2
})

# ...
```

```
calc_sum()
```

- Later we'll look at having the user push a button to have the app react, which is helpful for more computational heavy computations.

`reactive` Example (2.2)

- An example that uses the `reactive` function can be found in `./helloReactive/`

Delayed Reactivity (2.3)

- One may not want an app to immediately react to changes in user input, such as in cases of long-running calculation
- In order to prevent reactive expressions from reacting one can include a submit button in the app.

- An example of this can be found in `./helloSubmit/`, which is a "fork" of `./helloReactive/` but with the submit button (only comments about the submit button are present in this document).

Tabs (2.4)

- There are several other kinds of UI components that one can add to an app, such as tabs
  - tabs - gives app multiple views
  - navbars - to click around multiple tabs, like a top menu
  - sidebars - just like navbars but on the side
- There are functions for managing these tabs:
  - `tabsetPanel()` - specifies a group of tabs
  - `tabPanel()` - specifies the contents of an individual tab
- An example of tabs can be found in `./helloTabs/`

Using your own HTML (2.5)

- To use custom HTML one would save the code as `index.html` in a directory, named `www`, which is a child of the directory containing the `server.R` file

- Most of the style from Shiny's bootstrap setup in the header will no longer be needed

Interactive Graphics (2.6)

- A feature of Shiny is the ability to create graphics that a user can interact with.

- One method that can be used to select multiple data points on a graph is by specifying the `brush` parameter in `plotOutput()` in the `ui.R` file, then using the `brushedPoints()` function in the `server.R` file.

- The `./helloBrush/` directory gives an example that draws a line of best fit, given user selected points.

Shiny Gadgets

Overview (1.1)

- Shiny Gadgets provide a way to use Shiny's interactivity and UI as a part of a data analysis.
- A function is created that opens a small Shiny app in the RStudio viewer pane
  - Since it's smaller we use the `miniUI` package for manipulating the GUI
  - A gadget is a singular function that contains it's own `ui` and `server` functions.
- An example of a gadget to use can be found in `myFirstGadget.R` (were you expecting `helloGadget`?), the function's code is displayed below.

```
library(shiny)
library(miniUI)

myFirstGadget <- function() {
    ui <- miniPage(
        gadgetTitleBar("My First Gadget")
    )
    server <- function(input, output, session) {
        # Put in a Done button to close the app
        observeEvent(input$done, {
            stopApp()
        })
    }
    runGadget(ui,server)
}
```

Shiny Gadgets (1.2)

- An advantage of Shiny Gadgets is that since theya re functions they can take values as arguments and then return values.
- We'll create a simple example of a Gadget that has I/O
  - It will take two different vectors of numbers as arguments & use them to populate two `selectInputs`. The user can then choose two numbers within the Gadget and their product will be returned
  - (Note: I won't be creating a separate file for this one)

```
library(shiny)
library(miniUI)

multiplyNumbers <- function(numbers1, numbers2) {
  ui <- miniPage(
```

```
    gadgetTitleBar("Multiply Two Numbers"),
    miniContentPanel(
      #Make two drop-down boxes of selection options
      selectInput("num1", "First Number", choices = numbers1),
      selectInput("num2", "Second Number", choices = numbers2)
    )
  )

  server <- function(input, output, session) {
    observeEvent(input$done, {
                 num1 <- as.numeric(input$num1)
                 num2 <- as.numeric(input$num2)
                 stopApp(num1 * num2)
                 })
  }
  runGadget(ui, server)
}
```

Gadgets with Interactive Graphics (1.3)

- Gadgets can be used to create interactive plots, which can be helpful during the exploratory process or presentation of data.

- Below we'll be looking at an example of this.

```
library(shiny)
library(miniUI)

pickTrees <- function() {
  #User Interface
  ui <- miniPage(
    #Title
    gadgetTitleBar("Select Points by Dragging your Mouse"),
    #Show plot with brush interaction
    miniContentPanel(
      plotOutput("plot", #Plot is named "plot"
                 height = "100%", brush = "brush") #Brush is named "brush"
    )
  )

  #Logic
  server <- function(input, output, session) {
    #Display plot
    output$plot <- renderPlot({
      plot(trees$Girth, trees$Volume, main = "Trees!",
           xlab = "Girth", ylab = "Volume")
    })
```

```
    #On Done button, get selected points & print df of selection to console
    observeEvent(input$done, {
      stopApp(brushedPoints(trees, input$brush,
                            xvar = "Girth", yvar = "Volume"))
    })
  }

  runGadget(ui, server)
}
```

- The original data frame row numbers are retained, as such this can be helpful to quickly identify outliers in one's data

GoogleVis

GoogleVis (1.1)

- `googleVis` is a package that connects R to Google's visualization API
- Allows one to create visuals that would otherwise be difficult to create without it.
- googleVis charts can be embedded into the HTML when using Knitter too
- Btw googleVis uses flash, so unless they change something this will be obsolete after browsers stop supporting flash entirely.
  - Maybe check out the suggested packages in the answer to **this StackOverflow question.**

```
library(googleVis)
M <- gvisMotionChart(Fruits, #Example data included with GVis
                     idvar = "Fruit", timevar = "Year")

#To view the chart in a web browser with flash from Command line use
# plot(M)
# print(M, "chart") #prints HTML to run flash
```

Motion Chart Example

Charts in googleVis (1.2)

- Motion charts: `gvisMotionChart`
- Interactive maps: `gvisGeoChart`
- Interactive tables: `gvisTable`
- Line charts: `gvisLineChart`

- Bar charts: `gvisColumnChart`
- Tree maps: `gvisTreeMap`
- **The full documentation can be found on CRAN**

```
G <- gvisGeoChart(Exports,
                  locationvar = "Country", colorvar = "Profit")
# plot(G) #Commented out because... flash
```

Plots on Maps Example

- **Options that can be used with googleVis can be found here (Archived here, just in case)**

```
G <- gvisGeoChart(Exports, "Country", "Profit")
T1 <- gvisTable(Exports)
M <- gvisMotionChart(Fruits, "Fruit", "Year")
GT <- gvisMerge(G, T1, horizontal = FALSE) #Vertical combo (like cbind)
GTM <- gvisMerge(GT, M, horizontal = TRUE, #Horizontal combo
                 tableOptions = "bgcolor=\"#CCCCCC\" cellspacing=10")
#plot(GTM)
```

Combing Multiple Plots Together

```
M <- gvisMotionChart(Fruits, "Fruit", "Year")

#View locally (setting results = "asis" in Rmd would
# make the HTML code paste right in)
print(M)

# Save HTML to a file
print(M, 'chart', file = "myfilename.html")
```

Viewing the HTML code

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 based documents
  - Set `results="asis"` in the chunk options

  - Can be used with `kniter` and `slidify`
  - Sometimes clearing `kniter` cache can help the (static) charts display

For More Info

- `demo(googleVis)` will show off some plots... using flash when needed
- **CRAN vignette**
- **CRAN package info**
- **Google's documentation**
- **Google's FAQ**

Reminder to Commit (04), Delete this line *AFTER* Committing

Plotly

Plotly (1.1)

Plotly (1.2)

Plotly (1.3)

Plotly (1.4)

Plotly (1.5)

Plotly (1.6)

Plotly (1.7)

Plotly (1.8)

Reminder to Commit (05), Delete this line *AFTER* Committing

Quiz 1

1.
2.
3.
4.
5.

Reminder to Commit (Q1), Delete this line *AFTER* Committing

R Markdown and Leaflet

R Markdown

R Markdown (1.1)

R Markdown (1.2)

R Markdown (1.3)

R Markdown (1.4)

R Markdown (1.5)

R Markdown (1.6)

Sharing R Markdown Documents

Reminder to Commit (06), Delete this line *AFTER* Committing

Leaflet

Leaflet (1.1)

Leaflet (1.2)

Leaflet (1.3)

Leaflet (1.4)

Leaflet (1.5)

Leaflet (1.6)

Reminder to Commit (07), Delete this line *AFTER* Committing

Quiz 2

1.
2.
3.
4.
5.
6.

Reminder to Commit (Q2), Delete this line *AFTER* Committing

Course Project 1

Reminder to Commit (P1), Delete this line *AFTER* Committing

Building R Packages

R Packages

R Packages Overview

R Packages (1.1)

R Packages (1.2)

Building R Packages Demo

Reminder to Commit (08), Delete this line *AFTER* Committing

R Classes and Methods

R Classes and Methods (1.1)

R Classes and Methods (1.2)

Reminder to Commit (09), Delete this line *AFTER* Committing

Quiz 3

1.
2.
3.
4.

Reminder to Commit (Q3), Delete this line *AFTER* Committing

Course Project 2

Reminder to Commit (P2), Delete this line *AFTER* Committing

Swirl

Swirl

Swirl (1.1)

Swirl (1.2)

Swirl (1.3)

Reminder to Commit (10), Delete this line *AFTER* Committing

Course Project 3

Reminder to Commit (P3), Delete this line *BEFORE* Committing