Shiny

Course Title: **Operations Research**
Course Number: **ETM540/640**
Instructor: **Dr.Anderson**
Term: **Winter**
Year: **2013**
Author(s): **Dale Frakes**

ETM OFFICE USE ONLY
Report No.:
Type: Student Project
Note:

ETM540 Class Project
Winter 2013

Dale Frakes

21 March 2013

1 Introduction

The goal of this project is to explore the use of Shiny, a web-scripting front-end for R, for making Operations Research tools, and specifically DEA (Data Envelopment Analysis) tools, available to web users without requiring they install software on their local machines.

R[1] is a powerful open-source tool for performing statistical and mathematical analysis. It's highly extensible and by adding the GLPK package[2], it can be used to solve DEA problems.

Shiny[3] is a graphical front-end for R that can be run directly from an R or RStudio[4] session, or can be installed as a web-server. The web-server capability is still in *beta*, but when installed on a server machine, is able to allow users to interact with specific R/Shiny scripts without having to install R on their local machine.

The ultimate goal of this project is to make various DEA applications available to users in a way that they can upload their own datasets to the server, then apply DEA to their datasets without installing any software on their own machines.

2 The Key Questions

Without going very far in describing the work of this project, I'd like to address the main questions that this project is established to address. They are each listed below as subsections in this document followed by my answers.

[1] http://www.r-project.org/
[2] http://cran.r-project.org/web/packages/Rglpk/index.html
[3] http://shiny.rstudio.org/
[4] http://www.rstudio.com/

2.1 What do we need to do to get Shiny running? Preferably at the College of Engineering

"We have an RStudio instance running (rstudio.cecs.pdx.edu) which needs an Intel Lab Account. Ex. Do we need to have anything else installed on the MCECS RStudio server?"

As outlined further in this document, there are two basic ways to interact with Shiny. One is to run RStudio (either as a server or running locally) the load the Shiny library and run Shiny applications. The other is to install Shiny Server on a Linux box and allow users to interact directly with the Shiny Applications.

I personally would recommend setting up the Shiny Server, even though it's still in *beta* according to GitHub.

2.2 How do we run an R-Script that uses the LPSolveAPI package? (Our DEA and TFDEA relies on this.)

I'm still learning how to use R, but I suspect that anything you currently do in R can be done with a Shiny application using R. If LPSolveAPI currently works with R, a Shiny app referring to LPSolveAPI should also work. I still need to verify this.

2.3 How do we "load" new data using an R/Shiny script?

"Perhaps a CSV file from a local computer. We have a MySQL database at MCECS but it is only accessible from within the MCECS domain."

The example codes show how to load a csv file up to the server using Shiny Server. But it doesn't work with the sample csv file that the "download example" provided, so there is some debugging to do.

Once that is solved, there are two main questions that need to be addressed, assuming this works:

- How do we deal with making sure the data that's uploaded matches what's expected by the script? I don't know the answer to this yet but will be working on it.
- What happens if two different users load the same filename? Will one overwrite the other, or will they each have their own userspace?

2.4 Can you get one of my DEA or TFDEA scripts to run?

I haven't gotten this far in working with R/Shiny. But I'm close to being able to try to make this happen. The existing script will need to be re-written to use Shiny's *reactivity* variables. That is, anything that's a variable needing user input or is an output to be displayed needs to be re-factored using the *reactivity* of Shiny. This sounds complicated at first but so far it doesn't really seem to be (at least in very basic examples).

From the documentation about Reactivity[5], a simple example is shown below:

```
datasetInput <- reactive({
   switch(input$dataset,
          "rock" = rock,
          "pressure" = pressure,
          "cars" = cars)
})
```

In the script that defines the UI portion of the Shiny application, there is an input *combo-box* called *dataset* that allows the user to select from "rock", "pressure", and "cars". The above piece of code shows how the user's selection is put into an R variable called *datasetInput*. At that point, *datasetInput* can be used like any other R variable.

Outputting some variables, tables, or charts in Shiny also work through *Reactivity* and from the same example, that looks like:

```
output$view <- renderTable({
   head(datasetInput(), n = input$obs)
})
```

This is where I need to do more work with programming for Shiny itself. And this is where I'm a little weak because I'm not yet proficient in using R (yet).

2.5 What do we add to the web page to make this work?

If you install Shiny Server, you can simply point a link to it from any other part of the website. By default, it actually listens to a different port than a normal web server (8100 instead of 80). This can be remedied by installing Shiny Server behind a proxy and allowing a standard web server like Apache handle the main connection and then redirect it to the Shiny server and applications.

It should be possible to have a *Shiny Apps* web page that lists details about the various applications/scripts you want to make available and then provide a link to that specific application. Such a page could even provide example data templates that can be downloaded and populated to then be uploaded to the Shiny App.

2.6 Where is the computational load imposed? (in a heavy computational load, are the MIPs consumed on the server or the browser/client.)

The load will be imposed on the machine actually running the Shiny Server. The extent of that load could be measured using standard Linux tools like *top* and *htop*.

I also suspect that the load could be limited and managed using standard Linux tools like *nice* or *renice*, though there is probably a way to do this in the *init* code/script that launches Shiny at boot.

[5] http://rstudio.github.com/shiny/tutorial/#reactivity

Once I get further into this, I would like to do some load-testing to see what the impact of running computationally-challenging scripts is on the server. However I suspect that there will not be much difference between running a native R script vs. running an equivalent Shiny script. Shiny may add a bit of overhead, but I suspect it will be negligible.

3 Installing Shiny

There are two main ways to interact with Shiny. The first is simply to install the Shiny package into a session of R and then running a Shiny script. This is probably the best way to work on the development Shiny applications. This is also sufficient for deploying a Shiny application to someone who already has access to R or RStudio on their workstation.

This can also work when setting RStudio as a web service. In this mode, the user connects to the computer with their web-browser and is then able to work with RStudio like they would if it were running locally. From here, they can easily load and run Shiny scripts. The downside to this is that they must have an account on that machine.

The second way is to install Shiny-Server on a machine capable of working as a web-server. The biggest advantage to doing this is that the user does not need to have an account on the machine.

3.1 Running Shiny From Within RStudio

If you're already running in RStudio, it's pretty easy to get started using Shiny. First you have to install the Shiny packages in your account or on the machine for everyone. The following line in R will accomplish this:

```
1 install.packages('shiny')
```

Note, to make this work "for everyone", then you must be running R as *root*. This will then install the Shiny package where it will be available to all users on this machine.

Once you have Shiny installed in R, you can see it in action by loading the library and running one of the example programs that come built-in with Shiny.

```
1 library(shiny)
2 runExample("01_hello")
```

Note that when the *runExample* code is executed, Shiny will be running (and no other commands can be run) and it will output "Listening on port 8100" and then attempt to open a new browser window or table and point it to *http://localhost:8100/*.

This should work fine if you're running RStudio as a local application. However, if you're running this as an RStudio server session, the server won't actually be on *localhost*. For example, I currently have an RStudio server running on a Linode virtual computer at *http://192.81.129.82:8787* and when I run those commands, the window opens still pointing to *http://localhost:8100/*. To actually access the Shiny application, you need to change that to *http://192.81.129.82:8100/*, pointing it to the actual server located at *192.81.129.82*.

3.2 Running Shiny as a Web Server

After having my Linode machine hacked, I decided to experiment with building and installing a machine on my local network. With that machine I installed Shiny Server and am able to interact with it as I would any other web server. It's located at: *http://50.53.36.167:3838*

When you go to that link, you'll see the "folders" for this server installation. The "example" link are the canned examples that come with Shiny - and they work as expected.

The "sandbox" link is a set of shiny scripts that were on the link for installing Shiny Server - and they have examples of things we want to do (upload files as examples) however they don't seem to work properly yet.

4 Notes and Recommendations

Below are my general recommendations and observations while working on this project.

- As far as I can tell, the web-server versions of RStudio and Shiny are only available on Linux based machines. So to install these tools, a server running Linux will be necessary. Github has instructions[6] for installing Shiny on an Ubuntu Linux machine. These instructions should work (possibly with some modifications) on other non-Ubuntu based Linux machines.

- Development of Shiny and development using Shiny is currently very active. A great resource for seeing other people's problems and some solutions is the Google Group dedicated to Shiny[7]. It's not unusual to see up to 35 messages each day on this group.

- Shiny Server may not work as well with Firefox as the client. In a few cases, I've seen some example programs that did not function properly in Firefox, but seemed to work just fine with Internet Explorer and Chrome/Chromium. However this may be a function of the ad and script blocking add-ons I use with Firefox

- Per the RStuio installation page[8], it seems best, from a security and reliability point-of-view, to configure RStudio Server and/or Shiny Server behind an *industry standard* web server like Apache and set up proxy to direct the connections to the appropriate server. The details for making this configuration with Apache can be found here on the RStudio website[9].

5 Further Work and Questions

There is still further work to be done and I hope to continue pursuing these challenges over the next few weeks. Among them are:

[6] https://github.com/rstudio/shiny-server/wiki/Ubuntu-step-by-step-install-instructions
[7] http://groups.google.com/group/shiny-discuss
[8] http://www.rstudio.com/ide/docs/server/getting_started/
[9] http://www.rstudio.com/ide/docs/server/running_with_proxy

Answer Remaining Key Questions Among the key questions are:

- How to upload data
- Making sure LPSolveAPI works
- Implementing the TFDEA scripts in Shiny

User-Uploaded Data How do you do ensure the structure of the data is proper for doing DEA analysis? Or how do you identify columns of data for each type of analysis?

Concurrent Users Does each user have their own "workspace"? What happens if two different users both upload a file with the same name e.g. two users concurrently upload a file named *mydata.csv*.

Implement Basic DEA Program Dr. Burkett[10], an economics professor at the University of Rhode Island (burkett@uri.edu) has a very nice and simple set of DEA examples implemented in R[11], and I have his permission to use his examples as easy test-cases of DEA programs to implement in Shiny. I'd like to try and implement them as a trial-run to more difficult examples.

[10] http://www.uri.edu/research/isiac/burkett.htm
[11] http://www.uri.edu/artsci/ecn/burkett/DEAnotes.pdf