Co-lab Shiny Workshop - Hello Shiny!

Normal Probability Density Histogram, Reactivity, ggplot, OPM Overview

October 24, 2019

thomas.balmat@duke.edu
rescomputing@duke.edu

1 Overview

- Preliminaries
  - What is R?
  - What is Shiny?
  - What can Shiny do for you?
  - What are your expectations of this workshop?
  - Interest in R Day?
- Examples
  - Example visualizations
  - Example Shiny apps
- Resources
- Shiny use cases
  - Personal use to expedite research and results
  - Develop an app for public use
- Workshop material
  - From github (execute locally)
  - From RStudio Cloud
- Anatomy of a Shiny app
- First app - Histogram of random, normally distributed values
  - Version 1: single parameter prompt, immediate histogram rendering
  - Version 2: two parameter prompt
  - Version 3: mean and std dev parameters, improved script structure, use of defined functions, parameter validation, side bar panel
  - Version 4: curve control parameters, further HTML formatting, CSS, incremental plot construction
  - Version 5: editing script in an external IDE, execution using `server.r` and `ui.r`

- Reactivity
- HTML
- Debugging
- Second app - OPM Central Personnel Data File summary
  - Development of analysis in R
  - Shiny implementation
  - Slider bar for plot animation by a selected independent variable
- Creating a pseudo app server environment
  - Combine several `ui()` and `server()` functions in a single `navbarPage()`
  - Execute apps within individual R environments
  - Execute apps in parallel within a single R environment

2 Examples

2.1 Visualizations

- `ggplot` gallery: https://www.r-graph-gallery.com/all-graphs.html
- `ggplot` extensions: https://www.ggplot2-exts.org/gallery/

2.2 Shiny Apps

- Duke Data+ project, *Big Data for Reproductive Health*, http://bd4rh.rc.duke.edu:3838
- Duke Data+ project, *Water Quality Explorer*, http://WaterQualityExplorer.rc.duke.edu:3838
- Duke Med H2P2 Genome Wide Association Study, http://h2p2.oit.duke.edu
- Shiny gallery: https://shiny.rstudio.com/gallery/

3 Resources

- R
  - Books
    - Norm Matloff, *The Art of R Programming*, No Starch Press
    - Wickham and Grolemund, *R for Data Science*, O'Reilly
    - Andrews and Wainer, *The Great Migration: A Graphics Novel*, https://rss.onlinelibrary.wiley.com/doi/pdf/10.1111/j.1740-9713.2017.01070.x
    - Friendly, *A Brief History of Data Visualization*, http://datavis.ca/papers/hbook.pdf
  - Reference cards
    - R reference card: https://cran.r-project.org/doc/contrib/Short-refcard.pdf
    - Base R: https://rstudio.com/wp-content/uploads/2016/10/r-cheat-sheet-3.pdf
    - Shiny, `ggplot`, `markdown`, `dplyr`, `tidy`: https://rstudio.com/resources/cheatsheets/
- Shiny help
  - `?shiny` from the R command line
  - Click `shiny` in the `Packages` tab of RStudio

  - https://cran.r-project.org/web/packages/shiny/shiny.pdf

- ggplot help
  - ?ggplot2 from the R command line
  - Click ggplot2 in the Packages tab of RStudio
  - https://cran.r-project.org/web/packages/ggplot2/ggplot2.pdf

- Workshop materials: https://github.com/tbalmat/Duke-Co-lab/tree/master/Session-1

4 Shiny use cases

4.1 Personal Use

Shiny can expedite identification of model and visualization parameter values that illustrate important properties of systems that you want to document or publish. Instead of repetitious modification of "hard coded" values in scripts, followed by execution, Shiny can re-execute a script using parameter values taken from on-screen prompts (text inputs, radio buttons, slider bars, etc.). Figure 1 is a screen-shot of a Shiny app that was developed to compare incidence proportion of words appearing in case opinions of two categories. The adjustable p-window parameter aids in identifying filtering bounds on proportion that reveal significant associations (high p in both dimensions), while eliminating spurious ones (low p in either dimension). Slider bar adjustment of p-value replaces iterative R script editing and manual plotting. Shiny scripts are available at https://github.com/tbalmat/Duke-Co-lab/tree/master/Examples/Law/OpinionWordProportionXY

Figure 2 is a screen-shot from another legal text analysis app that identifies, for various case and opinion types, edge frequencies that reveal word pair proportions and correlation. Choice of small edge frequency values fails to reveal all important associations, while large values cause a cluttered graph that hides primary associations. The edge frequency slider input replaces iterative R script parameter value specification and manual plotting. Shiny scripts are available at https://github.com/tbalmat/Duke-Co-lab/tree/master/Examples/Law/OpinionTextCorrelationGraph

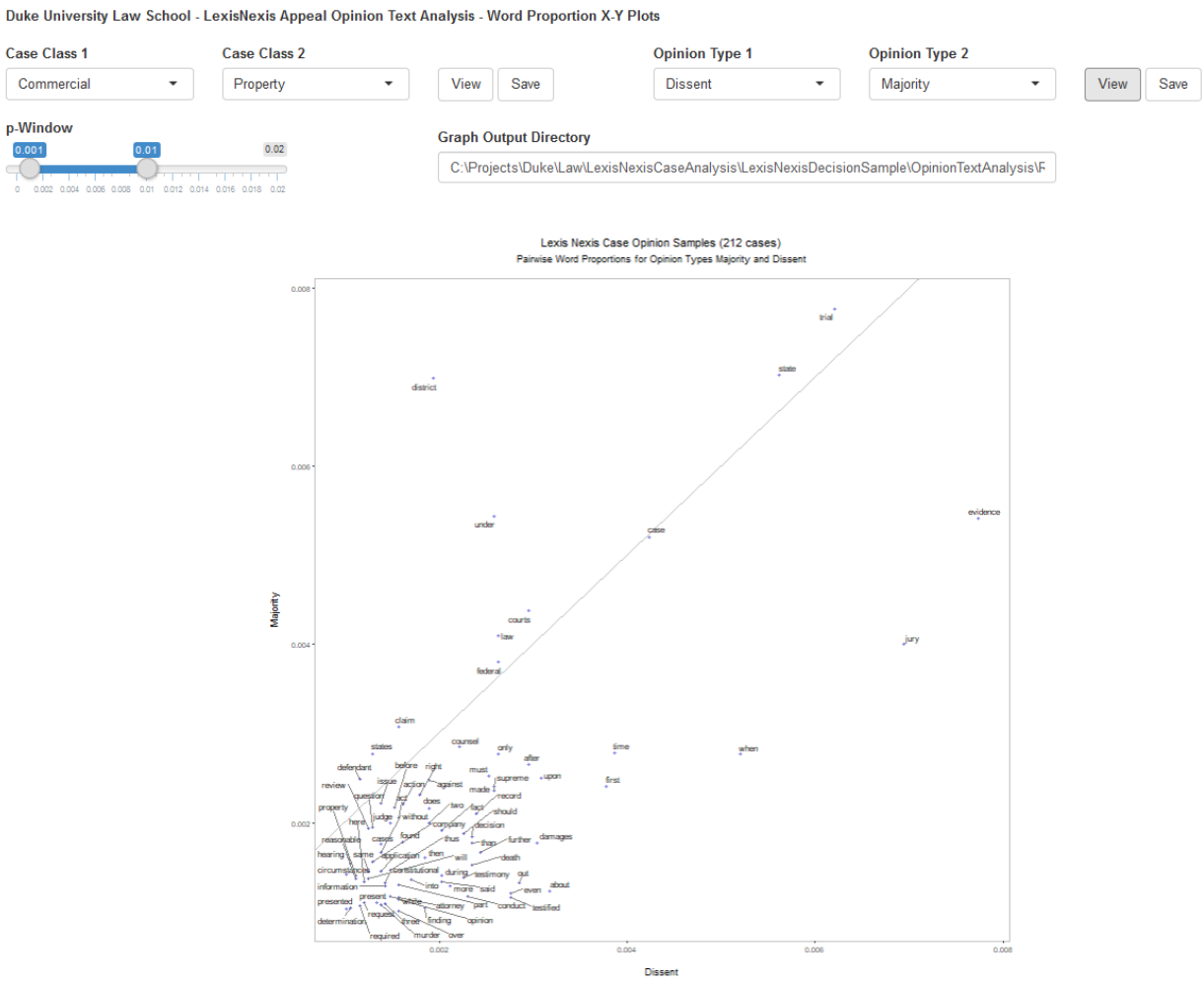

Figure 1: Shiny app developed to explore two-way incidence proportions of words in legal text.

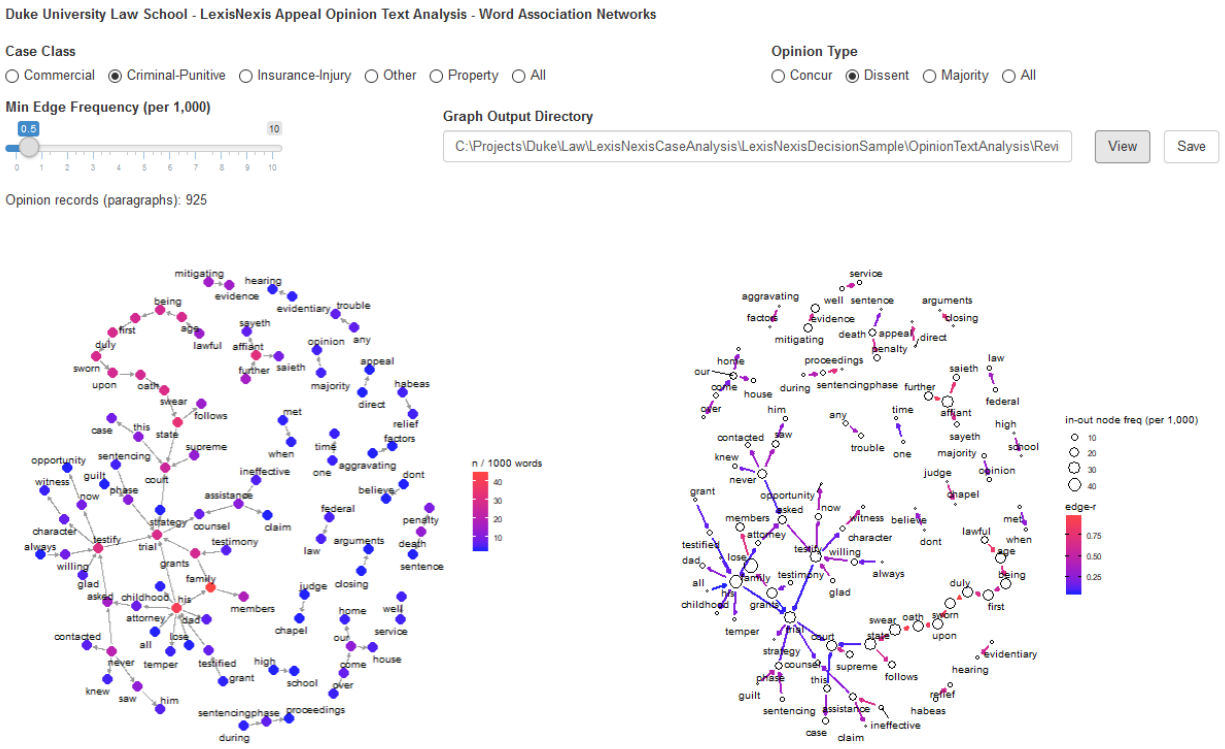

Figure 2: Shiny app developed to explore leading and trailing word pair correlation in case opinion text, using a graph with filtered edge density.

4.2 Public App

Alternatively, apps can be developed for public use, so that others can investigate your data and models and become better informed from your research. Figure 3 is a screen-shot from an app developed by the H2P2 (Hi-Host Phenome) Project at Duke. Researchers from around the world use it to explore associations between 150 cell pathogens and 15,000,000 chromosomal positions. The site url is http://h2p2.oit.duke.edu. Example Shiny scripts are available at https://github.com/tbalmat/Duke-Co-lab/tree/master/Examples/H2P2

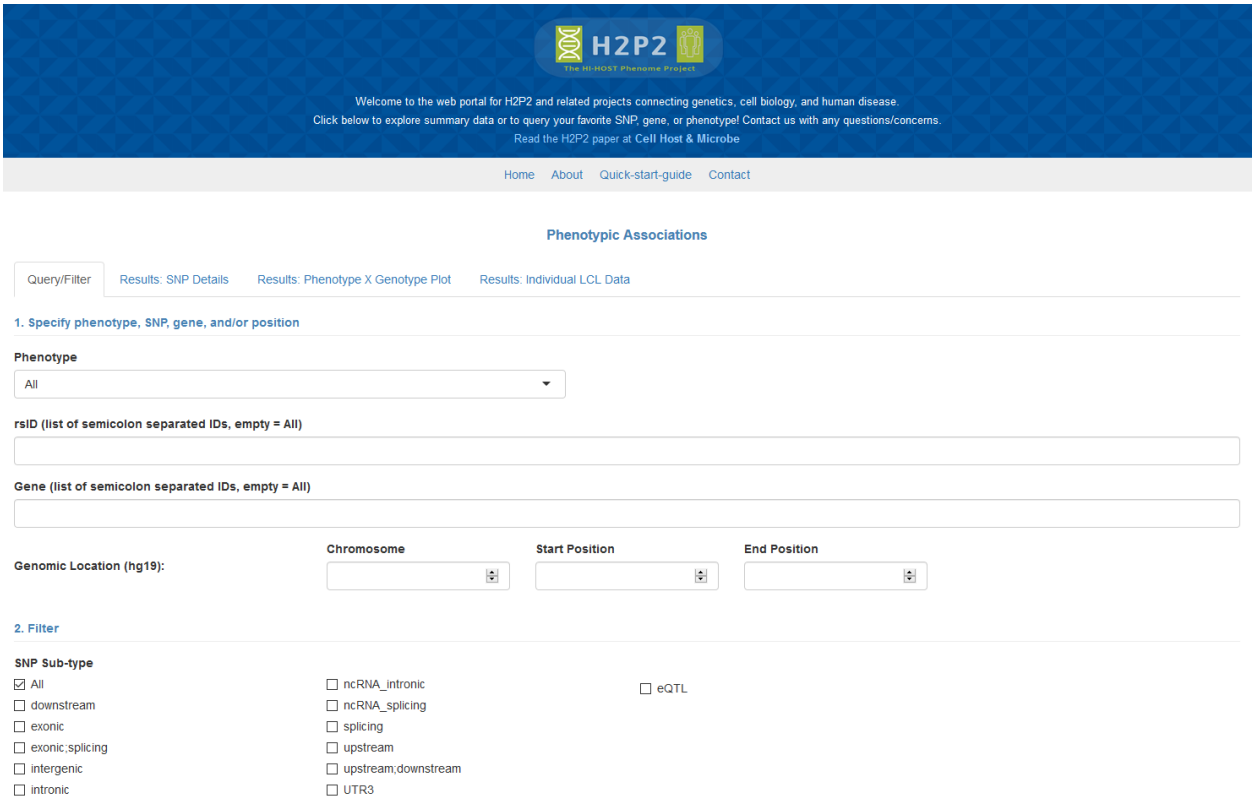

Figure 3: Shiny app developed to query and plot results from a Duke hosted genome wide association study.

5 Access Workshop Material

5.1 Execute Locally (copy workshop material from github repo)

- Copy scripts and data from https://github.com/tbalmat/Duke-Co-lab/tree/master/Session-1
- All workshop scripts should function locally

5.2 RStudio Cloud

- What is RStudio Cloud?
  - *We* [RStudio] *created RStudio Cloud to make it easy for professionals, hobbyists, trainers, teachers and students to do, share, teach and learn data science using R.*
- With RStudio Cloud
  - You do not need RStudio installed locally

  - Packages and data are available without installation and transfer

- Access workshop material
  - Create an Account: https://rstudio.cloud
  - Workshop project link: https://rstudio.cloud/project/580472

- All workshop scripts should function on RStudio Cloud, except OS shells that are used to automate execution

6 Anatomy of a Shiny App

A Shiny app is an R script executing in an active R environment that uses functions available in the Shiny package to interact with a web browser. The basic components of a Shiny script are

- `ui()` function
  - Contains your web page layout and screen objects for inputs (prompt fields) and outputs (graphs, tables, etc.)
  - Is specified in a combination of Shiny function calls and raw HTML
  - Defines variables that bind web objects to the execution portion of the app

- `server()` function
  - The execution portion of the app
  - Contains a combination of standard R statements and function calls, such as to `apply()`, `lm()`, `ggplot()`, etc., along with calls to functions from the Shiny package that enable reading of on-screen values and rendering of results

- `runApp()` function
  - Creates a process listening on a tcp port, launches a browser (optional), renders a screen by calling the specified `ui()` function, then executes the R commands in the specified server() function

7 First App - Histogram of Random, Normally Distributed Values

7.1 Version 1: Single Parameter Prompt, Immediate Histogram Rendering

Workshop file `App/NPDHist/NPDHist-1.r`. Features include

- Simple, "Hello World" example
- Prompts user for a number of random values to generate (`ui()` function)
- Generates the requested number of values, using a mean of 0 and standard deviation of 1 (`server()` function)
- Prepares a histogram of the generated values, using `ggplot()` (`server()` function)
- Displays the histogram in the user's browser (`ui()` function)
- The single on-screen adjustable object is a slider bar for selection of `n`, the number of values to generate
- Reference of `n` in a *reactive* context in `server()` causes immediate regeneration of random values and histogram rendering

Figure 4 is an example screen-shot of the app. The script that generated this app follows.

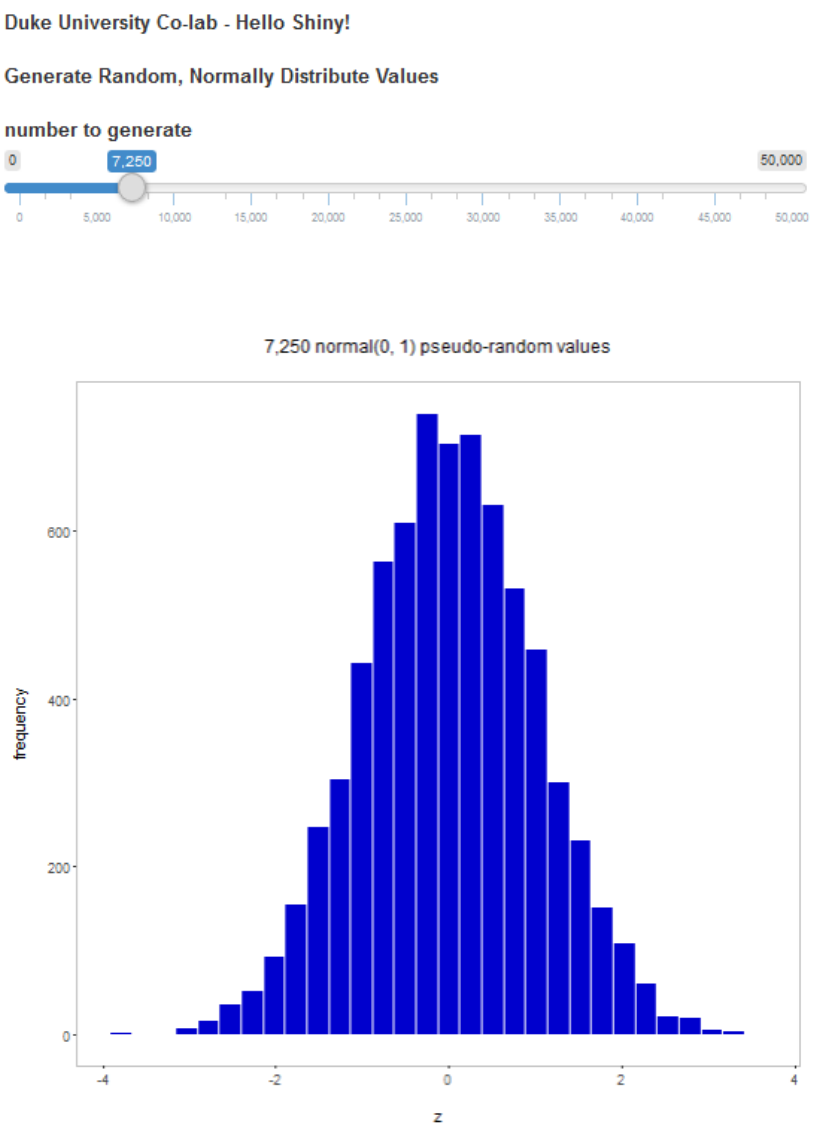

Figure 4: Screen-shot of first app. Normally distributed random values. Single parameter prompt.

```
# Shiny App
# Generate a histogram from random normal values
# Version 1, one reactive variables

options(max.print=1000)      # number of elements, not rows
options(stringsAsFactors=F)
options(scipen=999999)
#options(device="windows")

library(shiny)
library(ggplot2)

# A Shiny app consists of ui() and server() functions

# ui() can contain R statements (open a database and query it to populate selection lists, etc.), but its primary
# purpose is to format your web page (notice the explicit use of HTML tags)

# The HTML() function instructs Shiny to pass contained text to the browser verbatim, and is useful for formatting
# your page

# server() is a function containing R statements and function calls
```

```
# Any base function, functions declared in loaded packages (importantly, Shiny, here), or functions that you create
# in global memory can be called

# runApp() is a Shiny function that launches your default browser, renders a page based on the ui() function passed,
# then executes the server() function

ui <- function(req) {

    fluidPage(

    HTML("<br><b>Duke University Co-lab - Hello Shiny!<br><br>Generate Random, Normally Distributed Values</b><br><br>"),

    # Prompt
    fluidRow(width=12,
    column(width=5, sliderInput("n", "number to generate", min=0, max=50000, step=250, value=5000, width="90%"))
    ),

    HTML("<br><br><br>"),

    # Graph
    fluidRow(width=12,
    column(width=12, plotOutput("plot", width="600px", height="600px"))
    )

    )

}

server <- function(input, output, session) {

    # Use of cat() displays messages in R console, stderr() causes disply in red and writes to log (Shiny server)
    #cat("AAA", file=stderr())

    # Bind reactive variable(s)
    # They are referenced as functions in a reactive context (renderPlot, renderText, renderPlotly, renderTable, etc.)
    # Change in the value of reactive variables causes reactive function (renderPlot below) to be re-evaluated with new values
    n <- reactive(input$n)

    # Create and render plot
    # References to n() and w() cause re-execution of renderPlot() anytime input$n or input$w are modified
    # This gives the "instantaneous" or "fluid" appearance to graph updates in response to on-screen inputs
    output$plot <- renderPlot(
    ggplot() +
    geom_histogram(aes(x=rnorm(n())), color="white", fill="blue3") +
    scale_y_continuous(labels=function(x) format(x, big.mark=",")) +
    theme(plot.title=element_text(size=14, hjust=0.5),
    plot.subtitle=element_text(size=12, hjust=0.5),
    plot.caption=element_text(size=12, hjust=0.5),
    panel.background=element_blank(),
    panel.grid.major.x=element_blank(),
    panel.grid.major.y=element_blank(),
    panel.grid.minor=element_blank(),
    panel.border=element_rect(fill=NA, color="gray75"),
    panel.spacing.x=unit(0, "lines"),
    axis.title.x=element_text(size=12),
    axis.title.y=element_text(size=12),
    axis.text.x=element_text(size=10),
    axis.text.y=element_text(size=10),
    strip.text=element_text(size=10),
    strip.background=element_blank(),
    legend.position="bottom",
    legend.background=element_rect(color="gray"),
    legend.key=element_rect(fill="white"),
    legend.box="horizontal",
    legend.text=element_text(size=8),
    legend.title=element_text(size=8)) +
    labs(title=paste(format(n(), big.mark=","), " normal(0, 1) pseudo-random values\n", sep=""), x="\nz", y="frequency\n")
    )

}

# Execute
runApp(list("ui"=ui, "server"=server), launch.browser=T)}
```

Points to discuss

- Required libraries
- Instantiating `ui()` and `server()` functions

- Executing `runApp()`
- Terminating the app (esc key, stop sign)
- Residual browser effects
- `ui()` features
  - `fluidPage()`
  - `fluidRow()`
  - `column()`
  - Use of `HTML()`
  - Common error (`‘‘Warning: Error in tag: argument is missing, with no default"`) when delimiting comma missing between parameters of `fluidPage()`, `fluidRow`, `column()`, etc.
- `server()` features
  - `reactive(n)` declares `n` to be `reactive` causing reactive functions referencing `n()`, such as `renderPlot()`, to be executed when the on-screen parameter bound to `n` is updated
  - `renderPlot()` creates a *reactive* environment for `n()`)
  - Use of `ggplot()`
  - *Reactive* nature of `n`
- `runApp()` features
  - `launch.browser=T`
  - Alternate method(s) of launching a Shiny app (`ui.r`, `server.r`, `runApp(appDir=, host=, port=)`)

7.2 First App, NPD Version 2: Two Parameter Prompt

Workshop file `App/NPDHist/NPDHist-2.r`. Figure 5 is an example screen-shot. Features and discussion points include

- Modification of either on-screen parameter (`n` or `bar_width`) causes immediate regeneration of random values and histogram
- What consideration to problem size (number of observations, model computation time) should be given when multiple on-screen elements are reactive?

7.3 First App, NPD Version 3: Mean and Std Dev Parameters, Improved Script Structure, Use of Defined Functions, Parameter Validation, Side Bar Panel

Workshop file `App/NPDHist/NPDHist-3.r`. Figure 6 is an example screen-shot. Features and discussion points include

- Additional reactive variables for mean and standard deviation
- Model and plot parameter validation
- Use of defined function calls from within the `server()` function
- Further HTML formatting
- Implementation of `sidebarPanel()`

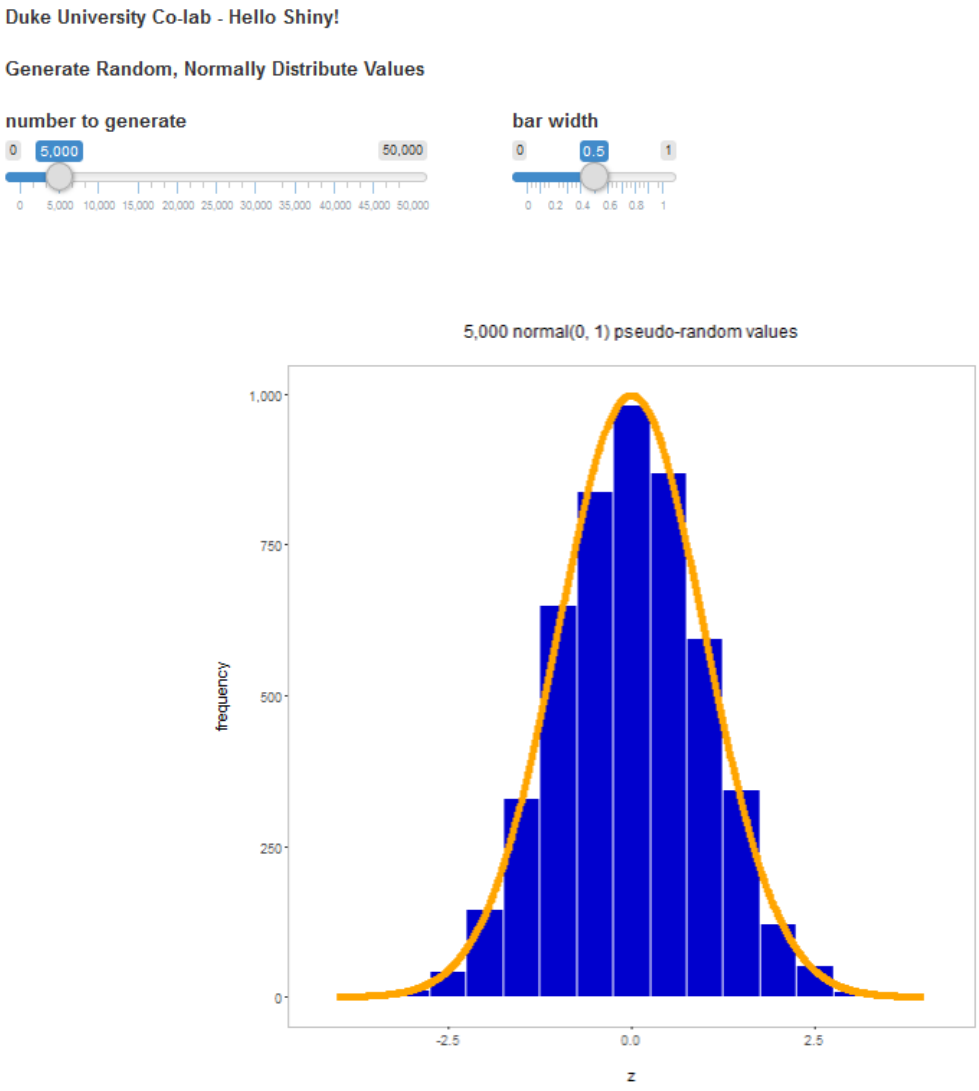

Figure 5: Screen-shot of first app, version 2. Normally distributed random values. Two parameter prompt.

7.4 First App, NPD Version 4: Curve Control Parameters, Further HTML Formatting, CSS, Incremental Plot Construction

Workshop file `App/NPDHist/NPDHist-4.r`. Figure 7 is an example screen-shot. Features and discussion points include

- Additional controls for bar color and continuous curve features
- Further HTML formatting (div for page and side bar margins)
- Use of cascading style sheet for HTML appearance
- Incremental construction of plot, based on requested features (web page controls)
- Suggestion: modify the list resulting from `ggplot()` to affect behavior and appearance
- Use of `input$x` within reactive function, as opposed to `x <- reactive(x)` followed by reference to `x()`

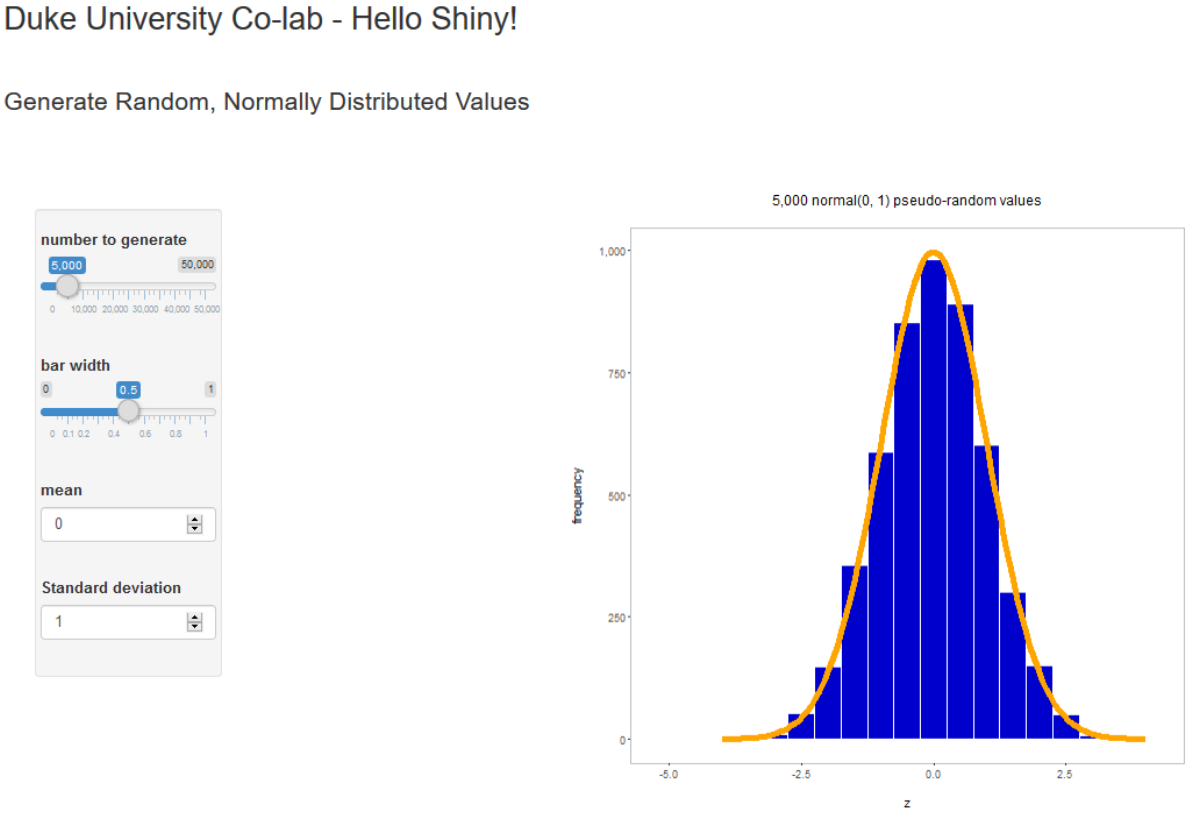

Figure 6: Screen-shot of first app, version 3. Normally distributed random values. Improved script, parameter validation, side bar panel.

- Is there any difference in the `input$x` and `x()` reference methods?

7.5 First App, NPD Version 5: Editing Script in an External IDE, Execution Using `server.r` and `ui.r`

Workshop files are in `App/NPDHist/ShellExecution`. Features and discussion points include

- Separate files for `ui()` and `server()` (improved isolation and maintainability)
- Option to use a preferred IDE for editing scripts
- Shell execution of R with `runApp()` targeting `ui.r` and `server.r` files, along with port specification implements a pseudo web server environment

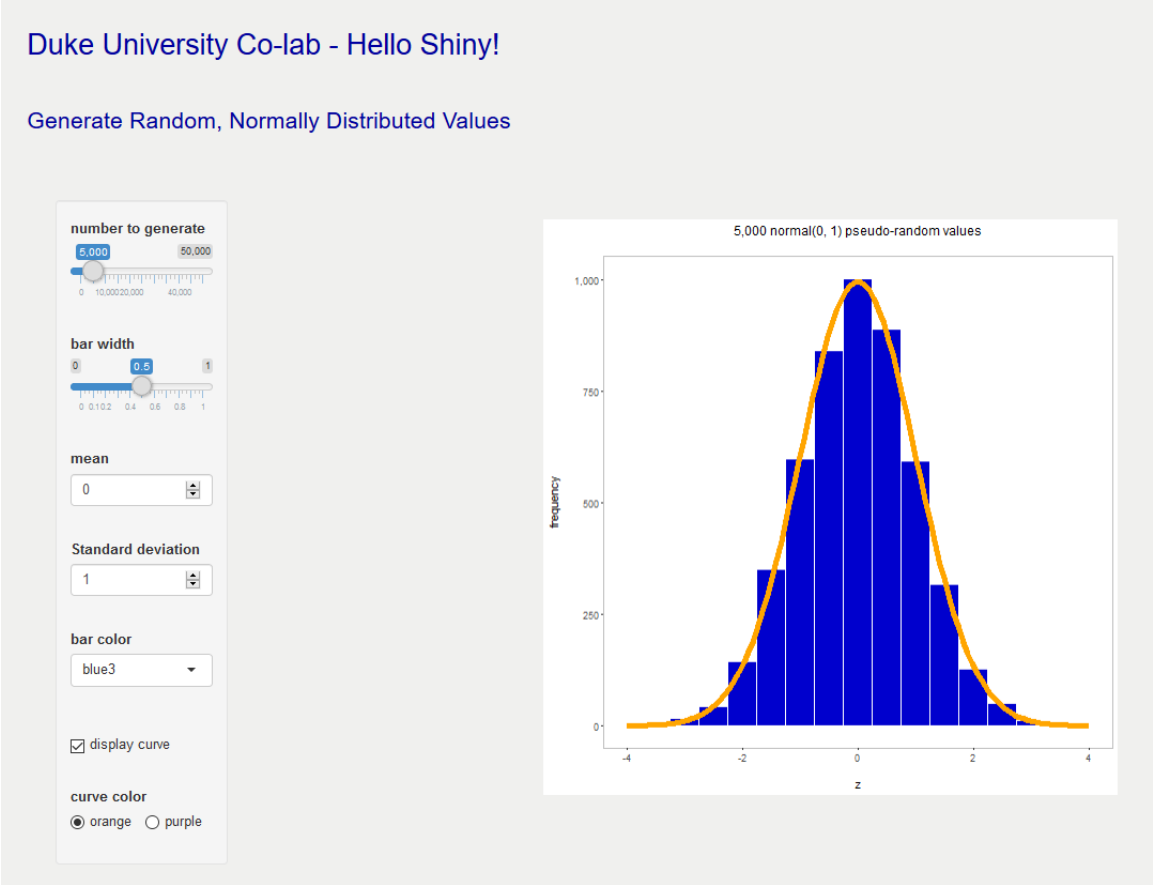

Figure 7: Screen-shot of first app, version 4. Normally distributed random values. Further HTML formatting, use of CSS, incremental plot construction.

8 Reactivity

Ideally, in the NPD app, changing bar color and enabling/disabling the continuous curve should not generate new random values, but simply regenerate the plot using the current values. However, The app does not distinguish between reactive parameter changes that are functionally important vs. merely aesthetic. Further, there does not seem to be a way of determining which reactive variable has triggered an event. Perhaps some info is available in the `reactiveValues()` list, https://shiny.rstudio.com/reference/shiny/latest/reactiveValues.html. The reactive log may be helpful, https://shiny.rstudio.com/reference/shiny/0.14/showReactLog.html. Considerations include

- Calls to `renderPlot(npdPlot())` have specified parameters in a reactive context (`input$n`)
- Can calls mix reactive and non-reactive variables?
- `isolate()` disables reactivity within a reactive environment
- Consider the example in `App/NPDHist/ShellExecution/isolate/server.r`
- The order of execution of reactive functions is not guaranteed, https://shiny.rstudio.com/articles/debugging.html
- Reactive functions (`observeEvent()`, `renderPlot()`) are executed during program initialization, since values transition from NULL to some initial value (`min` of a `sliderInput()`)
- Reactive variables (`input$x`) cannot be referenced outside of a reactive context
  Example: `if(input$x>0) output$plot <- renderPlot(... do something with input$x)`
  generates error message:
  `Error in .getReactiveEnvironment()$currentContext() : Operation not allowed without an active reactive context. (You tried to do something that can only be done from inside a reactive expression or observer.)`

More information on Shiny reactivity is available at https://shiny.rstudio.com/articles/reactivity-overview.html

9 HTML

Although we have, generally, used the `HTML()` function to inject tags for rendering, Shiny has a set of HTML builder and tag functions that accomplish similar results. For instance, the function calls `br()` and `HTML("<br>")` are equivalent. `br()` seems more compact, while `HTML("<br>")` seems more verbose. Additional information is available at https://shiny.rstudio.com/reference/shiny/latest/builder.html and https://shiny.rstudio.com/articles/html-tags.html

10 Debugging

It is important that you have a means of communicating with your app during execution. Unlike a typical R script, that can be executed one line at a time, with interactive review of variables, once a Shiny script launches, it executes without the console prompt. Upon termination, some global variables may be available for examination, but you may not have reliable information on when they were last updated. Error and warning messages are displayed in the console (and the terminal session when executed in a shell) and, fortunately, so are the results of `print()` and `cat()`. When executed in RStudio, Shiny offers sophisticated debugger features (more info at https://shiny.rstudio.com/articles/debugging.html). However, one of the simplest methods of communicating with your app during execution is to use `print()` (for a formatted or multi-element object, such as a data frame) or `cat(, file=stderr())` for "small" objects. The `file=stderr()` causes displayed items to appear in red. Output may also be written to an error log, depending on your OS. Considerations include

- Shiny reports line numbers in error messages relative to the related function (`ui()` or `server()`) and, although not always exact, reported lines are usually in the proximity of the one which was executed at the time of error
- `cat("your message here")` displays in RStudio console (generally, consider Shiny Server)
- `cat("your message here", file=stderr())` is treated as an error (red in console, logged by OS)
- Messages appear in RStudio console when Shiny app launched from within RStudio
- Messages appear in terminal window when Shiny app launched with the `rscript` command in shell
- There exists a "showcase" mode (`runApp(display.mode="showcase")`) that is intended to highlight each line of your script as it is executing
- The reactivity log may be helpful in debugging reactive sequencing issues (`options(shiny.reactlog=T)`, https://shiny.rstudio.com/reference/shiny/0.14/showReactLog.html It may be helpful to initially format an apps appearance with an empty `server()` function, then include executable statements once the screen objects are available and configured
- Although not strictly related to debugging, the use of `gc()` to clear defunct memory (from R's recycling) may reduce total memory in use at a given time

11 Second App - U.S. Office of Personnel Management Central Personnel Data Overview

The U.S. Office of Personnel Management maintains records on the careers of millions of current and past federal employees. The Human Capital and Synthetic Data projects at Duke have conducted various research using these data. Although the complete data set contains data elements that are private and not released to the public, OPM has released data sets with private elements omitted and with certain variables (age, for instance) induced with statistical noise. We will use a subset of results made available by Buzzfeed. The accuracy of the publicly available elements has been confirmed by comparison of data procured by Duke through FOIA requests. The data are highly aggregated, so that are analysis is limited to broad patterns, typically involving means of pay, age, education, etc. for large groups of employees. Additional information on the OPM data set, research at Duke, and Buzzfeed is available at

- The Office of Personnel Management: https://www.opm.gov/
- OPM Guide to Data Standards:
  - https://www.opm.gov/policy-data-oversight/data-analysis-documentation/data-policy-guidance/reporting-guidance/part-a-human-resources.pdf
  - https://github.com/tbalmat/Duke-Co-lab/blob/master/Docs/US-OPM-Guide-To-Data-Standards.pdf
- Duke Synthetic Data Project, Annals of Applied Statistics paper:
  - https://projecteuclid.org/euclid.aoas/1532743488
  - https://github.com/tbalmat/Duke-Co-lab/blob/master/Docs/AOAS1710-027R2A0.pdf
- U.S. Federal Grade Inflation supplement to Synthetic Data paper (section 9): https://github.com/tbalmat/Duke-Co-lab/blob/master/Docs/SynthDataValidationSupplement.pdf
- Buzzfeed OPM data: Buzzfeed(https://www.buzzfeednews.com/article/jsvine/sharing-hundreds-of-millions

Notes on the data set:

- Observations are limited to general schedule (GS) grades 1 through 15, fiscal years 1988 through 2011, full-time employees

- Columns included:
  - fy - U.S. federal government fiscal year
  - agency - federal agency employed (synthetically generated for workshop)
  - age - employee age (five year increments, noised induced by OPM)
  - grade - general schedule (GS) grade
  - occCat - occupational category
  - yearsEd - years of education
  - n - number of observations (employees) in fy, agency, age, grade, occCat, yearsEd combination
  - sumPay - sum of basic pay in fy, agency, age, grade, occCat, yearsEd combination (in 2011 $U.S.)
- There is one record for each unique combination of fy, agency, age, grade, occCat, yearsEd combination
- n and sumPay are aggregated within fy, agency, age, grade, occCat, yearsEd combinations

11.1 Second App, CPDF Analysis Development in R

Workshop file App/CPDF/CPDF-1.r. Two types of plots will be produced: an x-y plot to show relationships between two CPDF variables and a kernel density plot to show the distribution of observations for a given variable. Figure 8 is an example x-y plot and figure 9 is an example kernel density plot.

Discussion points:

- Computation of mean pay due to initial aggregation producing sum(pay) for each category
- Construction of common `ggTheme` (inspect and modify list results)
- Use of `aggregate()` to produce graphics data set
- Step-wise construction of plot (`g <- ggplot()`) (inspect list results)
- `ggplot()` features to be controlled in Shiny app: dependent var, independent var, faceting, color, alpha
- Use of `aes_string()`
- `geom_smooth()` with LOESS

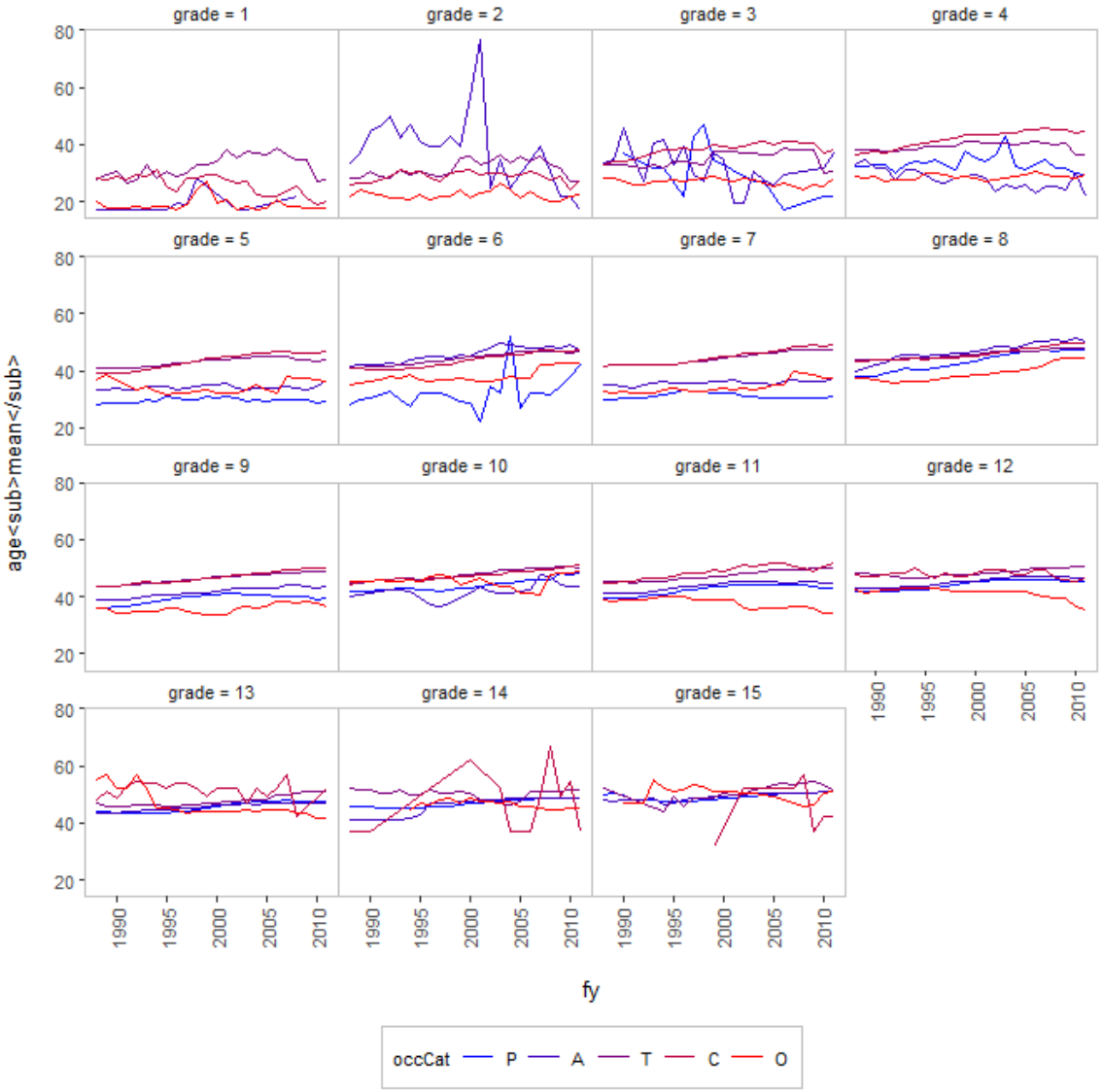

Figure 8: Example x-y plot from CPDF analysis development in R. Mean age vs. fiscal year.

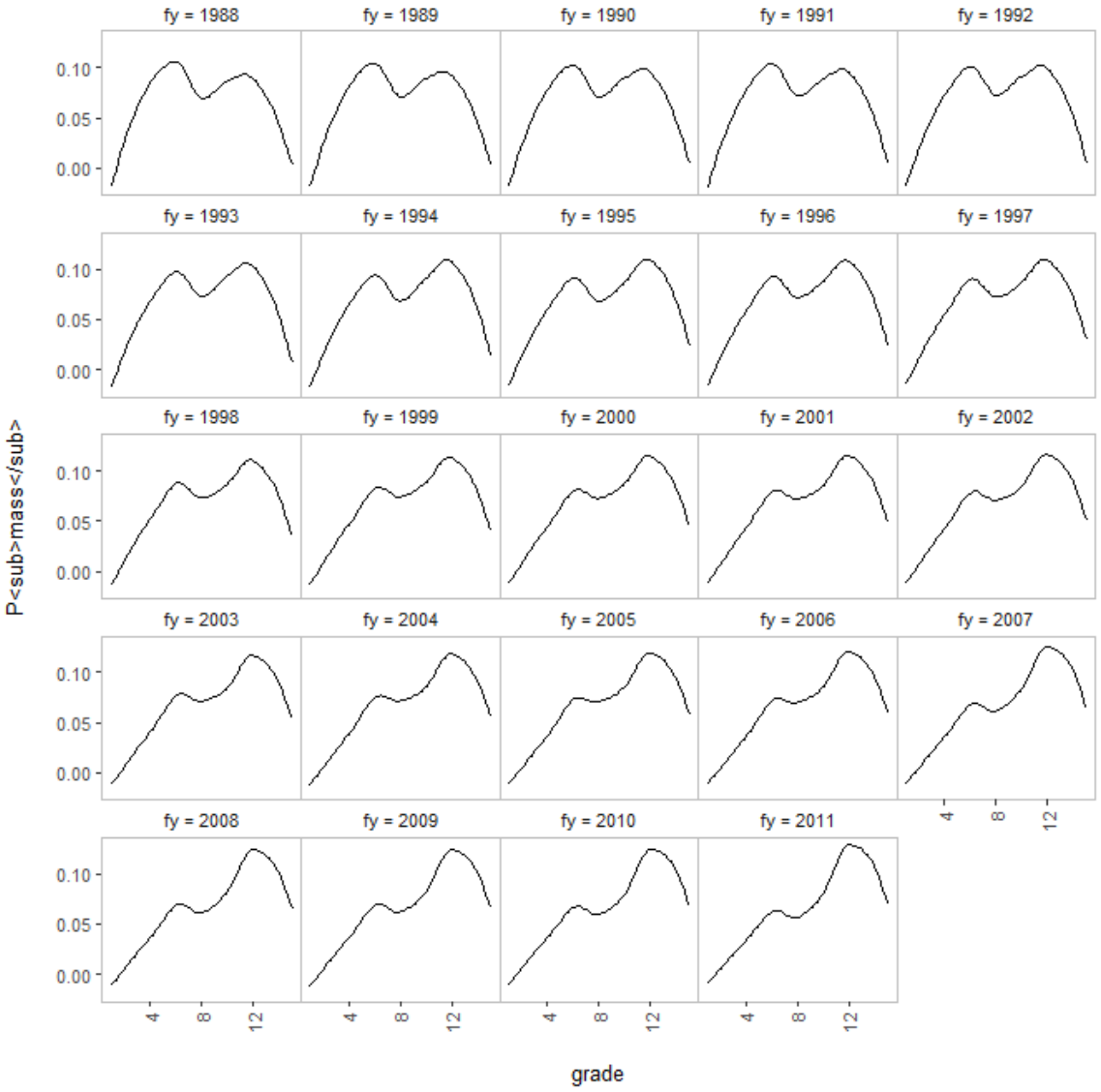

Figure 9: Example kernel density plot from CPDF analysis development in R. Distribution of grade by fiscal year.

11.2 Second App, CPDF Analysis in Shiny

Workshop files in App/CPDF/ShellExecution/CPDF-Shiny-1. These implement the CPDF R analysis in Shiny. Figure 10 is an example screen-shot of the x-y tab. Figure 11 is an example screen-shot of the distribution tab.
Features:

- CPDF observations are read into global memory to be available by both `ui()` and `server()`
- `tabsetPanel()`, one tab for x-y plots, one for distribution plots
- `sidebarPanel()` improves appearance and organization of screen prompts
- Reactivity limited to "plot" action button (note the assignment of reactive variables to local R variables in `server()` to decouple reactivity
- Data aggregation and plot generation is accomplished in functions outside of `renderPlot()` to improve program design, flow, and consistency through use of common procedure
- Ordering of occCat (PATCO) by creating a factor

11.3 Second App, Slider Bar for Plot Animation by a Selected Independent Variable

Workshop files in App/CPDF/ShellExecution/CPDF-FYSliderBar. This modification adds a slider bar for automated fiscal year scrolling. With it, fiscal year is incremented at a constant rate and a new plot is generated for the annual subset of observations. Longitudinal shifts and patterns become apparent as years advance. Figure 12 is an example screen-shot of this app.
Features:

- Use of conditional panel to display FY slider only when FY is not selected as the independent var
- Examine `sliderInput()` properties, https://shiny.rstudio.com/reference/shiny/0.14/sliderInput.html
- `plotly()` sliderInput appears more flexible (BD4RH example)
- Consideration: what is expected when fiscal year is specified as both the independent and panel variable?

Figure 10: CPDF x-y analysis in Shiny app. Mean pay by year, paneled by occupational category, colored by years of education.

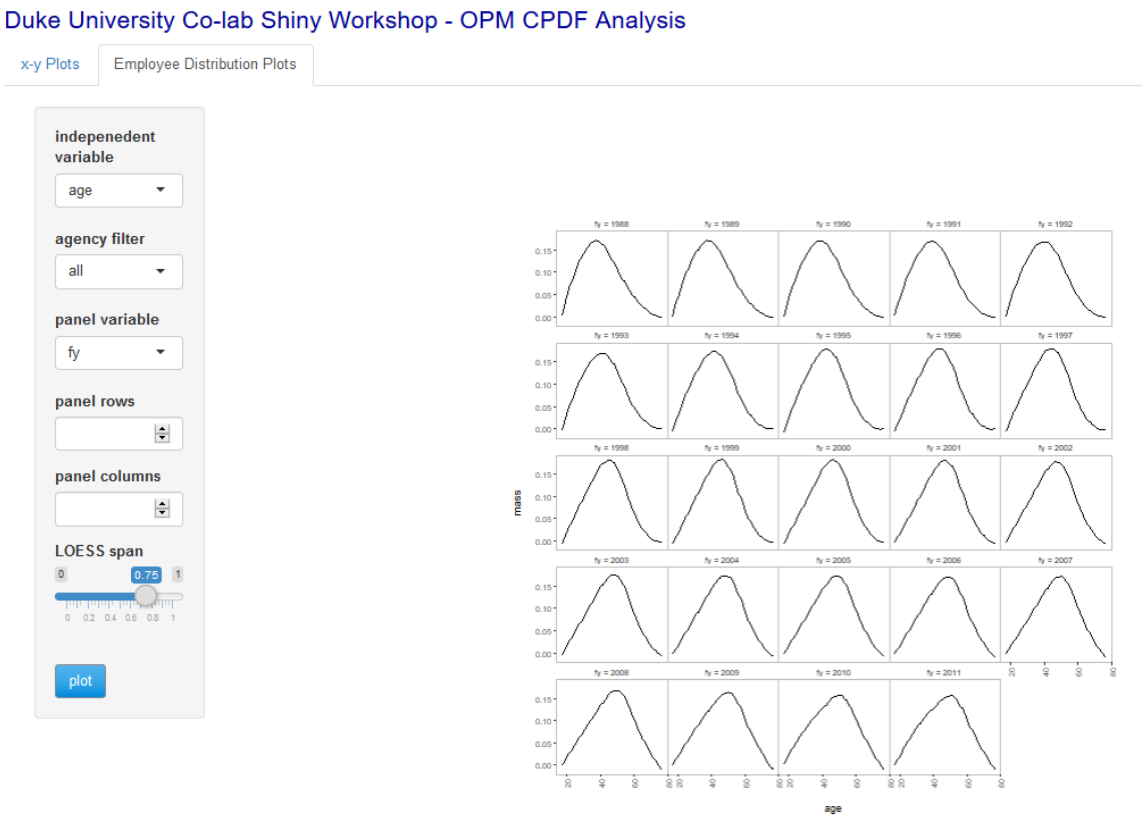

Figure 11: CPDF employee analysis in Shiny app. Distribution of age by year.

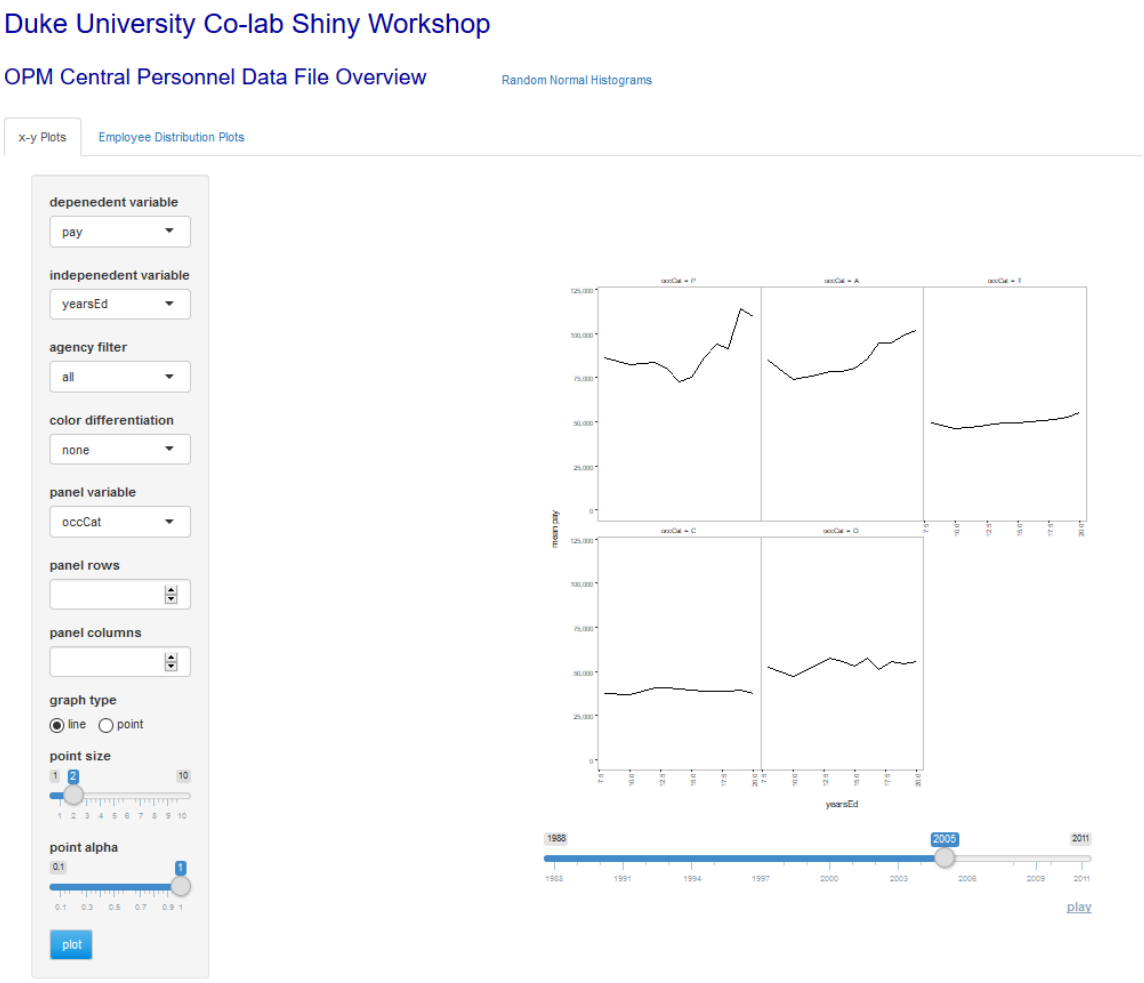

Figure 12: CPDF employee analysis in Shiny app. Slider bar for fiscal year animation.

12 Creating a Pseudo App Server Environment

12.1 Combine Several Apps Within a Single `navbarPage()`

- Advantages
  - Single file execution
  - Consistent, clean menu appearance (figure 13)
  - All R data objects shared between apps
  - State of all data objects retained when switching between apps
- Disadvantages
  - Scripts for all apps must be combined in a single pair of `ui()` and `server()` functions (`source()` may be useful in maintaining and loading individual files)
  - Global variables and functions with a common name must be renamed in order to maintain app independence

Figure 13: Example `navbarPage()` application that combines apps in a single R and menu environment.

12.2 Execute Apps Within Individual R Environments

Using the NPD histogram app of section 7.5 and the slider bar app of section 11.3, we note the following:

- Each is launched with a specific tcp port in `runApp(port=)`
- Each `ui.r` file contains an HTML anchor tag (`HTML("<a href=http://127.0.0.1:4292></a>")`) that references the other's port
- The anchor tags appear when each app is launched
- The clickable anchors serve as an entry to an external Shiny app
- Each time an anchor is clicked, the associated Shiny app is executed from it's beginning instruction, clearing memory, making it difficult to compare results in different apps difficult (although open-in-new-tab help)
- `runApp()` cannot be called from within `runApp()`, otherwise the scripts we have developed could be executed from within other scripts

12.3 Execute Apps in Parallel Within a Single R Environment

The following script (taken from workshop file App/ParallelShiny/ParallelShiny.r) demonstrates a method of loading individual apps in separate, parallel R processes.

```
# Launch two apps in parallel
# Note that each app specifies a unique tcp port for http requests
# The uir.r of each app has an anchor tag that targets the other app's listening port
# This way, one app can execute the other

library(parallel)
```

```
# Specify top level directory containing all ui.r and server.r files

# Local dir
setwd("C:\\Projects\\Duke\\Co-lab\\Shiny\\Session-1-NPDHist-CPDF\\App")

# Rstudio Cloud dir
#setwd("/cloud/project/Duke-Co-lab/Shiny/Session-1-NPDHist-CPDF/App"

# Define a function, to be called in parallel, that launches the specified app
execApp <- function(app) {

  library("shiny")

  if(app=="NPDHist") {
    appDir <- "NPDHist\\ShellExecution"
    tcpPort <- 4291
  } else if(app=="CPDF") {
    appDir <- "CPDF\\ShellExecution\\CPDF-FYSliderBar"
    tcpPort <- 4292
  }

  runApp(appDir=appDir,
  launch.browser=T,
  host = getOption("shiny.host", "127.0.0.1"),
  port=tcpPort,
  display.mode="normal")

}

# Create a two-core parallel cluster
cl <- makePSOCKcluster(rep("localhost", 2))

# Execute two apps
clusterApply(cl, c("NPDHist", "CPDF"), execApp)

# Stop cluster
stopCluster(cl)

# Clean up
rm(cl)
gc()
```

- Notes
  - The `execApp()` function executes a Shiny app, responding to a unique tcp port, as instructed by the value of its `app` parameter
  - The `makePSOCKcluster()` function of the `parallel` package (alternatively, `makeSOCKcluster()` of the `SNOW` package) is used to create a parallel cluster from local processors, one for each Shiny app to be executed
  - One core (processor) must be available for each Shiny app to be executed in parallel
  - clusterApply() calls `execApp()` once for the NPDHist app and once for the CPDF app, distributed to individual cores of the cluster
- Advantages
  - Individual app `ui.r` and `server.r` files are maintained, but all are launched with a single R script (once the `ui.r` and `server.r` files are composed, execution requires the above script only)
  - Each app functions as an independent http service (can be called from within your Shiny environment or independently from any browser)
  - Individual file sets for each app promotes hierarchical program structure and maintainability
- Disadvantages
  - Data states are lost when switching between apps