Co-lab Shiny Workshop

Interactive Data Tables and Plots

March 31, April 7, 2021

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

R provides a wide range of features and functions for transforming data, fitting models to data, and presenting results. However, to evaluate models and results under a set of possible scenarios, the analyst must iteratively adjust parameter values and manually execute R scripts. Shiny offers a means of presenting controls (text prompts, selection lists, radio button groups, etc.) on a web page for prompting user input and dynamically integrating the user's web environment with the analytical portion of an R script. Instead of altering the values of R variables, the user adjusts on-screen graphical controls and initiates an action (a button or immediate computation) then R scripts are executed and results presented, according to the particular Shiny features implemented in your app.

Shiny's input objects (`textInput`, `selectInput`, `sliderInput`, etc.) provide a means of presenting, to the user, placeholders for values to be used in an R script for querying data sets, subsetting observations, specifying model parameter values, configuring plot variables, and whatever else may be needed to conduct supported analyses. Additionally, using an appropriate update function (`updateSelectInput()`, `updateRadioButtons()`, etc.) the list of possible values presented to the user can be modified based on ranges of values observed in the data. Along with a fixed set of controls, we can provide a summarized view of the data, in tabular form, and use each row as a virtual menu entry that, when selected, will advance the user to another level of analysis, limited to data corresponding to that row.

In this session, we will develop a Shiny app to explore covariate relationships in a human capital data set by selecting subsets of observations indicated by rows of a summary aggregation table. Aggregation variables are specified by the user and corresponding data subsets are used to construct various plots based on further selection of dependent and independent variables. Values of on-screen Shiny controls are used in an R script to control table and graph construction. Design of the interaction between on-screen Shiny controls and internal R script variables using knowledge of the domain and data being studied is the key to producing an effective data exploration app. Development will begin with an R script (using no Shiny features) to demonstrate the types of analyses that will be converted into dynamic Shiny apps. A basic Shiny app will be developed from the analysis script and it will be extended in a series of apps that include increasingly useful Shiny and R features. The final version is representative of a professional, visually appealing app acceptable for published research. Important features include:

- Data table row selection
- Interactive data table formatting and search features (`DT` package)
- Interactive plot construction from data table subset (row) selection
- Control of `ggplot` parameters using on-screen Shiny controls
- Draggable panels for user control of screen layout
- Inspection and modification of themes for attractive app appearance
- User file search and upload capability
- Download link for review of sample input file format
- Aggregation table download capability
- Progress indicators
- Dynamic insertion of HTML windows containing supplemental project info and links
- Error handling to avoid simple crashes
- Modal windows for communication with user

1 Overview

- Preliminaries
  - What can Shiny, interactive data tables, and integrated plots do for you?
  - What are your expectations of this workshop?
- Examples
  - Example visualizations
  - Example Shiny apps
  - Example table and plot Shiny apps
- Resources
- Shiny Use Cases
- Anatomy of a Shiny App
- Reactivity
- Download Workshop Material and Configure R
- Example Data Set - U.S. Office of Personnel Management (OPM) Central Personnel Data File
  - Explore Increase in Mean Age with Fiscal Year
  - Explore Increase in Mean Age with Fiscal Year, Paneled by Grade
  - Explore Increase in Mean Age with Fiscal Year, Paneled by Grade, Differentiated by Occupational Category
  - Explore Increase in Proportion Employees by Grade and Fiscal Year
- Development of a Shiny App With Data Tables and Plots
  - Development of Analysis in R
  - Shiny App with Basic Table and Plot Features (V2)
  - Shiny App with Updated Appearance Using Navigation Bars and Themes (V3)
  - Shiny App with Draggable Panels (V4)
  - Shiny App for Public Deployment (V5)
- Advanced Data Tables Features
- Plot Animation - Fiscal Year Slider Bar
- Debugging

2 Examples

2.1 Visualizations

- `ggplot` gallery: `https://www.r-graph-gallery.com/all-graphs.html`
- `ggplot` extensions: `https://mode.com/blog/r-ggplot-extension-packages`

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 H2P2 project, *Host Pathogen Genome Wide Association Study*, http://h2p2.oit.duke.edu
- Duke iCPAGdb project, *Cross-Phenotype Analysis of GWAS*, http://cpag.oit.duke.edu
- Duke Nicholas School, *Health Effects of Airborne Pollutants*, http://shindellgroup.rc.duke.edu
- Shiny gallery: https://shiny.rstudio.com/gallery

2.3 Example Data Table apps

- Duke H2P2 GWAS phenotypic associations, http://h2p2.oit.duke.edu/PhenotypicAssociations. This app queries a large database of phenotypic and genotypic associations, produces a summary table of results in order of significance, and renders a boxplot of individual phenotypic response values corresponding to a single SNP (row) selected from the table. The plot and data subset can be further examined using on screen controls.
- Duke iCPAGdb review tab, http://cpag.oit.duke.edu. This app presents a table with each row describing a set of precomputed cross-phenotype analysis results. Computation time required to produce a selected analysis prohibits effective browsing of results. Further, presenting descriptions in a convenient, selectable format draws attention to important findings. Complete data used in a selected analysis are displayed in tables and heatmap plots.
- Shiny gallery, basic data table, https://shiny.rstudio.com/gallery/basic-datatable.html
- Review your data set on-line, https://shiny.rstudio.com/gallery/file-upload.html
- Shinyapps.io example, https://yihui.shinyapps.io/DT-rows/. This app highlights the plotted point corresponding to a selected data table row. Note the distinction between client and server side tables. The documentation for `renderDataTable()` (https://shiny.rstudio.com/reference/shiny/0.14/renderDataTable.html) states that only server side tables are implemented.

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
  - `?shiny` from the R command line
  - Click `shiny` in the `Packages` tab of RStudio

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

- dataTables
  - ?DT from the R command line
  - Click DT in the Packages tab of RStudio
  - https://cran.r-project.org/web/packages/DT/DT.pdf
  - https://rstudio.github.io/DT/
  - java-centric: https://datatables.net/reference/option/

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

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

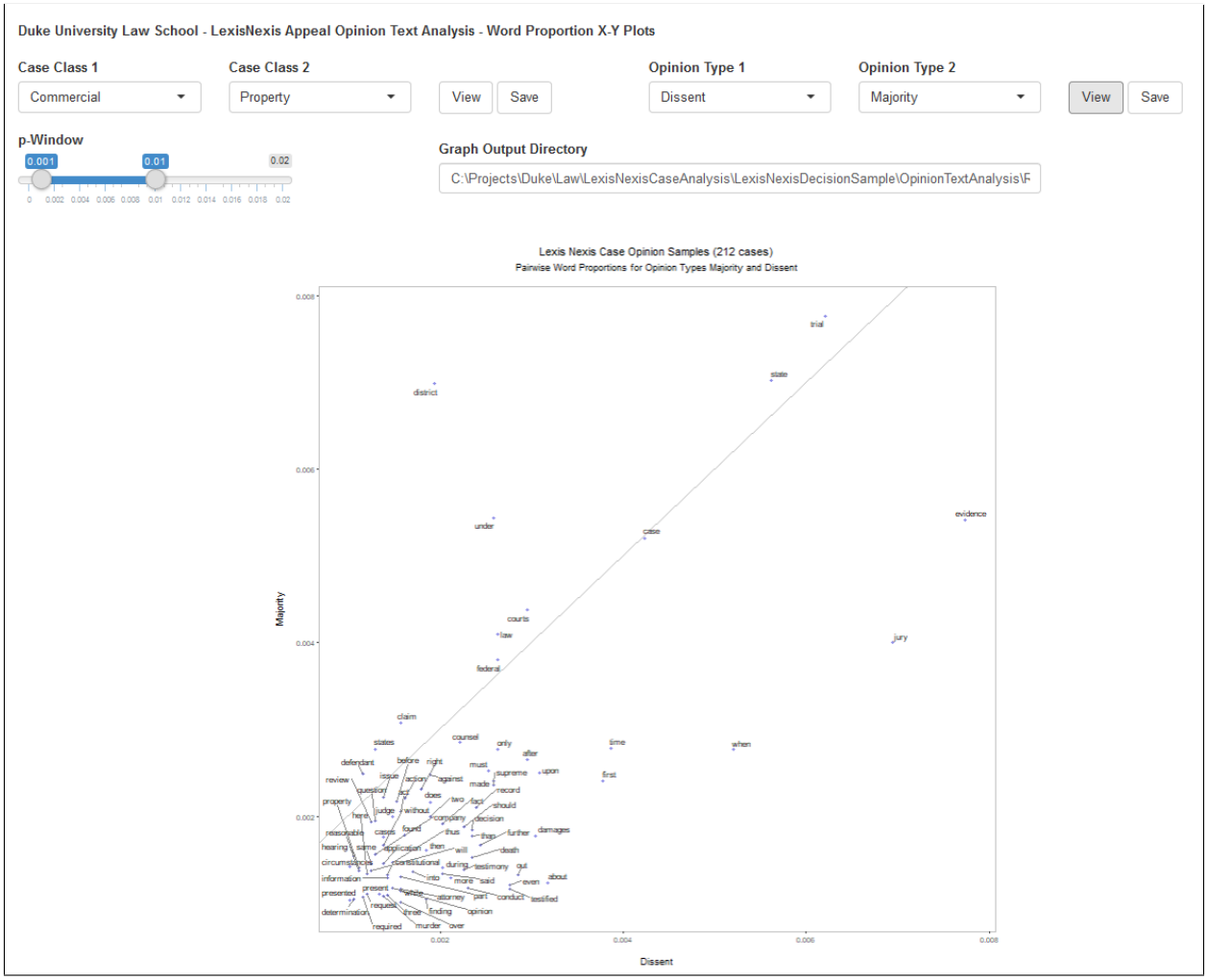


Figure 1: Shiny app developed to explore two-way incidence proportions of words in legal text.

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

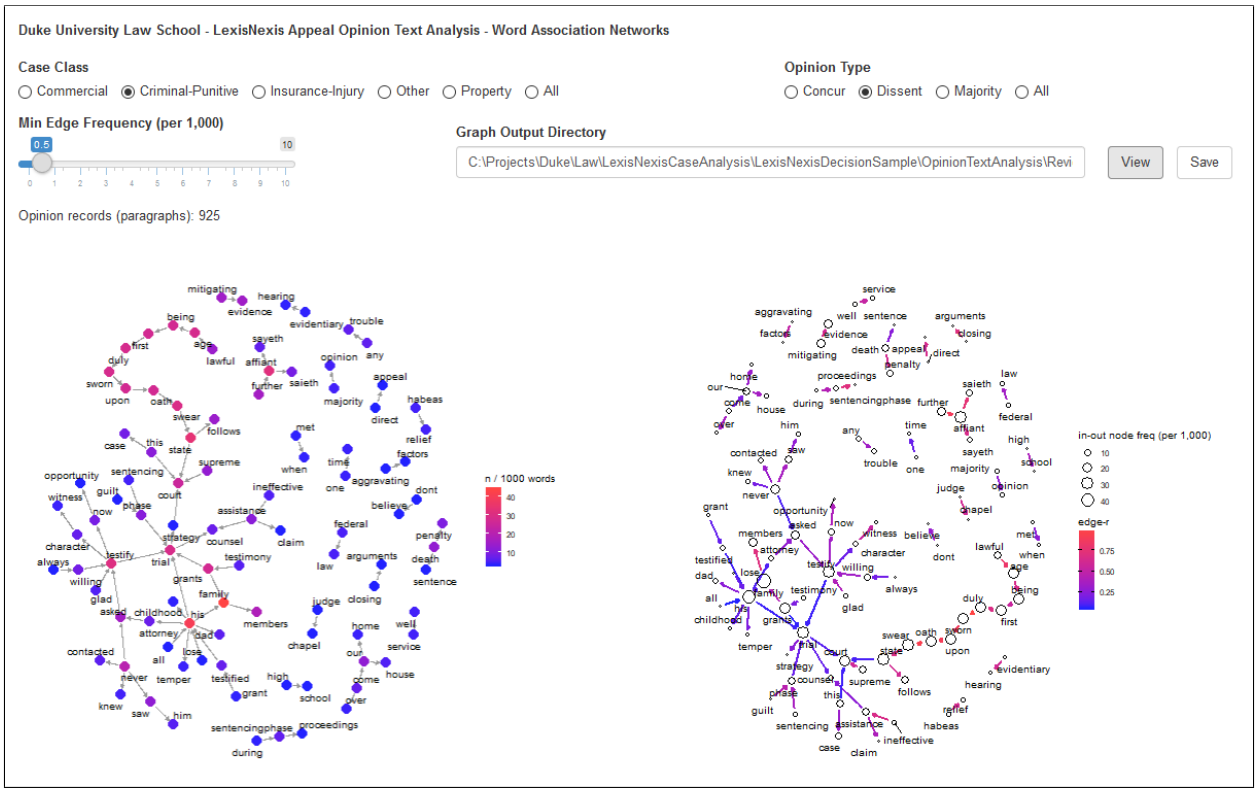


Figure 2: Shiny app developed to explore leading and trailing word pair correlation in case opinion text, using a graph with filtered edge density.

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 of the Upload and Compute tab of the iCPAGdb Shiny app developed by the H2P2 (Hi-Host Phenome) Project at Duke. Researchers from around the world are able to upload their data sets and execute sophisticated, comparative statistical analyses to identify possible genetic associations between disparate phenotypic studies, using the algrithms developed by the Duke team. In addition to the Compute tab presented in figure 3, the app has:

- a Review tab for exploring pre-computed results of significant findings from studies curated, or conducted, by the Duke team
- an About tab that explains the motivation behind the project, links to related research, information on app development, and contact links
- a Quick Start tab that is intended as a primer, to guide new visitors through use of the site

- a Bibliography tab that provides a consolidated list of links to related research

The site url is http://cpag.oit.duke.edu. Shiny scripts are available at https://github.com/tbalmat/iCPAGdb

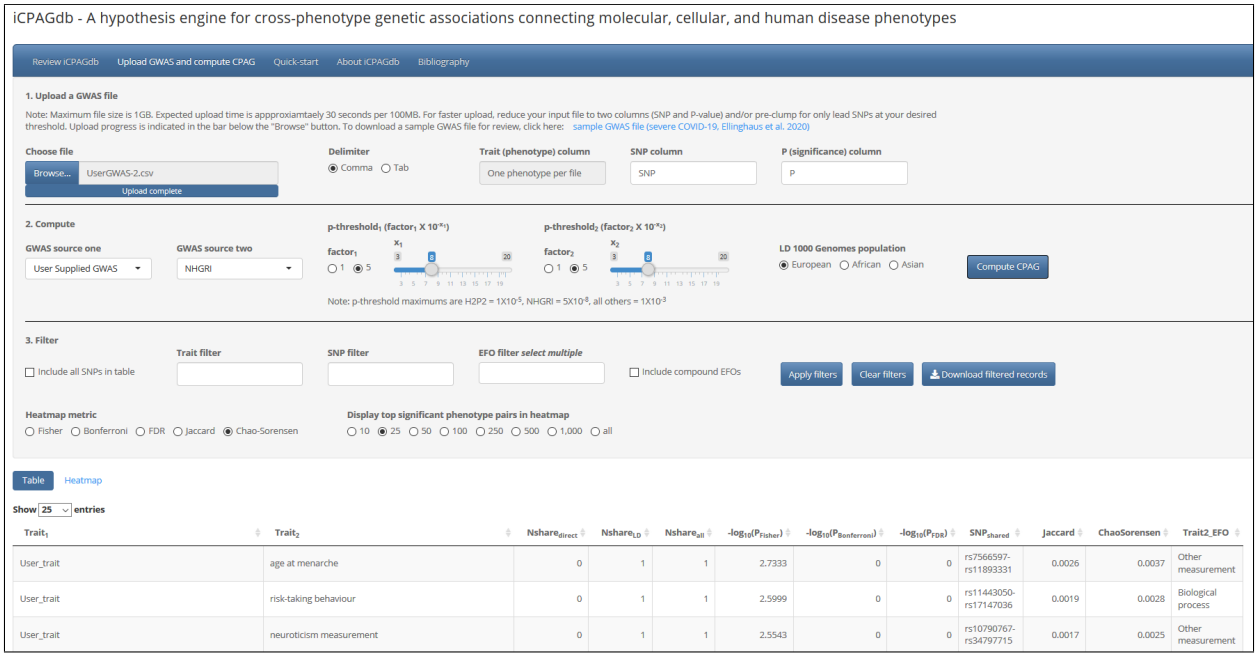


Figure 3: Shiny app developed by Duke team to allow researchers to upload data and execute cross-phenotype analyses of GWAS algorithms

5 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

6 Reactivity

Reactivity is the single most important feature that Shiny offers. Variables are defined in your `ui()` function with an `input$` prefix and when these variables appear in `observe()` functions within in your `server()` function, execution events are triggered by on-screen changes to the corresponding `ui()` variables. In addition to referencing `input$` variables `observe()` functions include standard R commands, including those supported by any valid R package, so that the reactive variables become parameters to your R functions, enabling dynamic analysis of data. Output is rendered in the app by targeting `ui()` variables defined with an `output$` prefix. A simple example follows. It has a single, numeric input (`x`) and one plot output (`plot`). Changes in `x` cause the `observeEvent()` to be executed. The `observeEvent()` generates a histogram of `x` random, normal values. The histogram is a suitable input value to `renderPlot()`. Assignment of the `renderPlot()` result to `output$plot` causes the histogram to be displayed as defined in `ui()`. Notice how a Shiny input variable (`input$x`) is used as a parameter to an R function (`rnorm()`) and the result of an R function (`plot()`) is used as a parameter to a Shiny function (`renderPlot()`). Note that modifying `x` to its current value does not cause execution of the `observeEvent()` (try it).

```
library(shiny)

# Define UI
ui <- function(req) {
  numericInput(inputId="x", label="x"),
  plotOutput(outputId="plot")
}

# Define server function
server <- function function(input, output, session) {
  observeEvent(input$x, {
    output$plot <- renderPlot(hist(rnorm(input$x)))
  })
}

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

7 Download Workshop Material and Configure R

- Copy course outline, scripts, and data from https://github.com/tbalmat/Duke-Co-lab/tree/master/Spring-2021/Session-1-2-DataTables-Plots
  - App.zip
  - Co-lab-Session-1-2-DataTables-Plots.pdf
  - Data.zip
  - SupplementalMaterial.zip
- Expand zip files (one subdirectory per file)
- Launch RStudio
- Install packages:
  - Shiny: `install.packages("shiny")`
  - Shiny: `install.packages("shinythemes")`
  - Shiny: `install.packages("shinyjs")`
  - ggplot: `install.packages("ggplot2")`
  - Data Tables: `install.packages("DT")`

8 Example Data Set - OPM CPDF

The U.S. Office of Personnel Management (OPM) maintains records on the careers of millions of current and past federal employees in what is called the Central Personnel Data File (CPDF). 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 review 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 used here are highly aggregated, so that 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: https://www.buzzfeednews.com/article/jsvine/sharing-hundreds-of-millions-of-federal-payro

Notes on the data set:

- Observations are limited to:
  - Fiscal years 1988 through 2011
  - General schedule (GS) grades 01 through 15
  - Full-time employees
  - Grade between 01 and 15
  - Occupational category in P, A, T, C, O
  - Education level between 01 and 22
  - Adjusted basic pay > 10 (thousands per year)
  - Top five agencies (left two positions) by observation frequency
- Columns include:
  - 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

For motivation, we will use a Shiny app to expose a few trends in the data. Important features of federal employee human capital include a general increase in age, education, pay grade, and pay throughout the study period, along with a decrease in proportion persons occupying clerical positions (occupational category "C") and an increase in proportion persons occupying professional and administrative positions (occupational categories "P" and "A"). These trends should be clearly revealed in the output produced by our app.

Executing the app:

- From the `V9-CPDF-FYSliderBar` directory (downloaded in section 7)
  - Edit `ui.r` and modify the statement

    ```
    setwd("C:/Projects/Duke/Co-lab/Shiny-Spring-2021/Session-1-2-DataTables-Plots/App/V9-CPDF-FYSliderBar")
    ```

    to reference the `V9-CPDF-FYSliderBar` directory on your computer
  - Edit `CPDF-FYSliderBar.r` and modify the statement

    ```
    ad <- "C:/Projects/Duke/Co-lab/Shiny-Spring-2021/Session-1-2-DataTables-Plots/App/V9-CPDF-FYSliderBar"
    ```

    to reference the `V9-CPDF-FYSliderBar` directory on your computer
- Load `CPDF-FYSliderBar.r` into RStudio
- Execute the loaded script

8.1 Motivation Graph 1, Increase in Mean Age with Fiscal Year

On the x-y Plots tab of the app, select `age` for dependent variable and `fy` for independent variable then click the `plot` button. As seen in figure 4, mean aggregate age does increase with fiscal year throughout the study period.

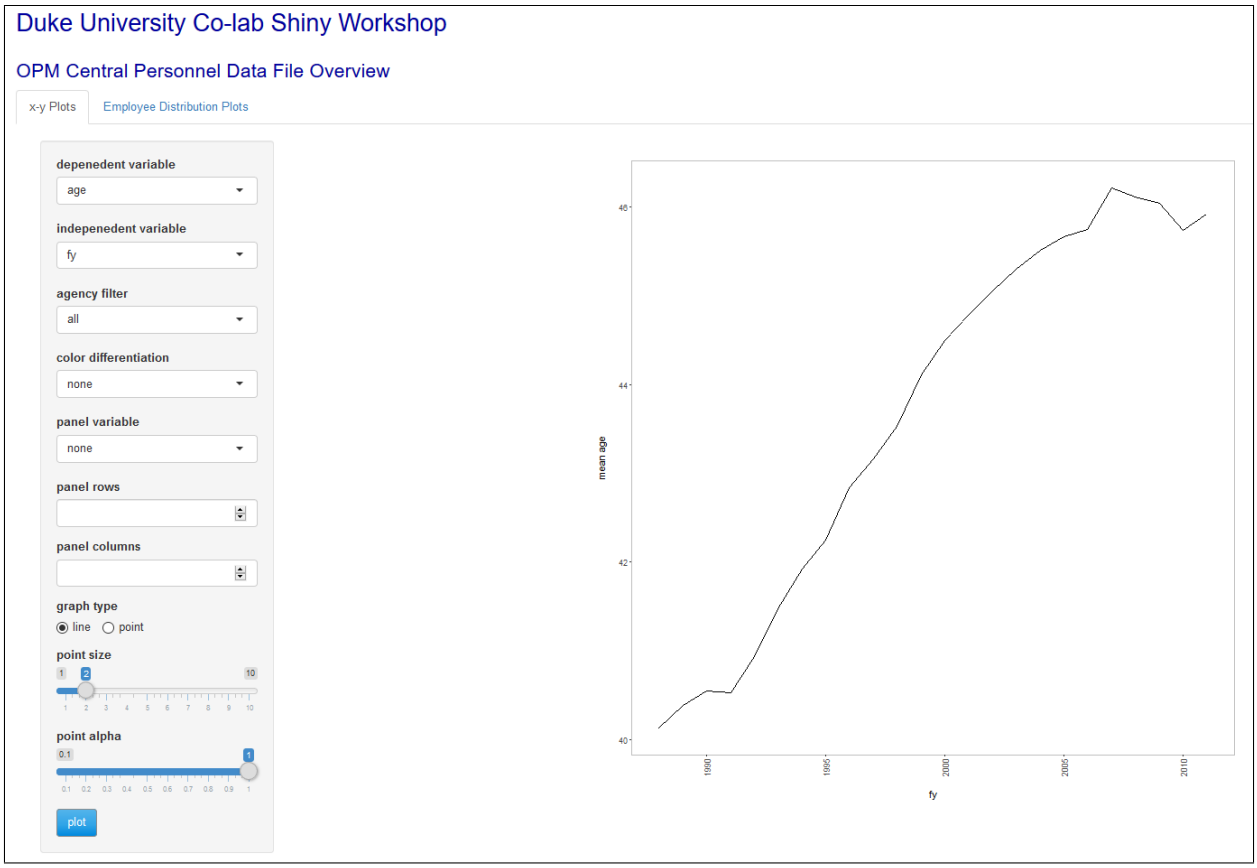


Figure 4: OPM CPDF graph of mean employee age vs. fiscal year

8.2 Motivation Graph 2, Increase in Mean Age with Fiscal Year, Paneled by Grade

Selecting `age` for dependent variable, `fy` for independent variable, and `grade` for panel variable produces figure 5. This panel reveals differences in mean age by grade (generally, greater age with grade) and slight variation in the increase of mean age by fiscal year for various grades.

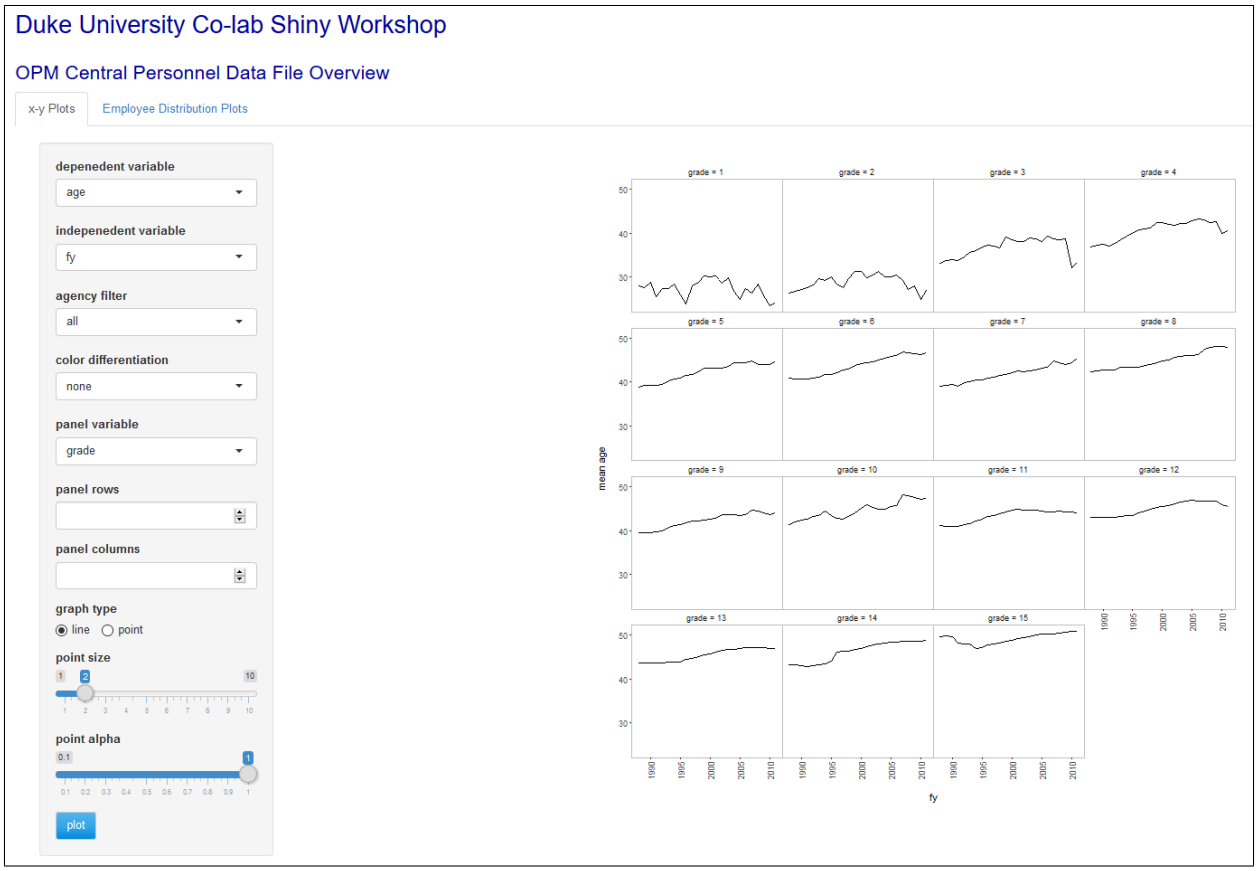


Figure 5: OPM CPDF graph of mean employee age vs. fiscal year, paneled by grade

8.3 Motivation Graph 3, Increase in Mean Age with Fiscal Year, Paneled by Grade, Differentiated by Occupational Category

Selecting `age` for dependent variable, `fy` for independent variable, `grade` for panel variable, and `occCat` for color differentiation produces figure 6. This panel reveals further differences in the rate of change in mean age by fiscal year for various grades and occupational categories.

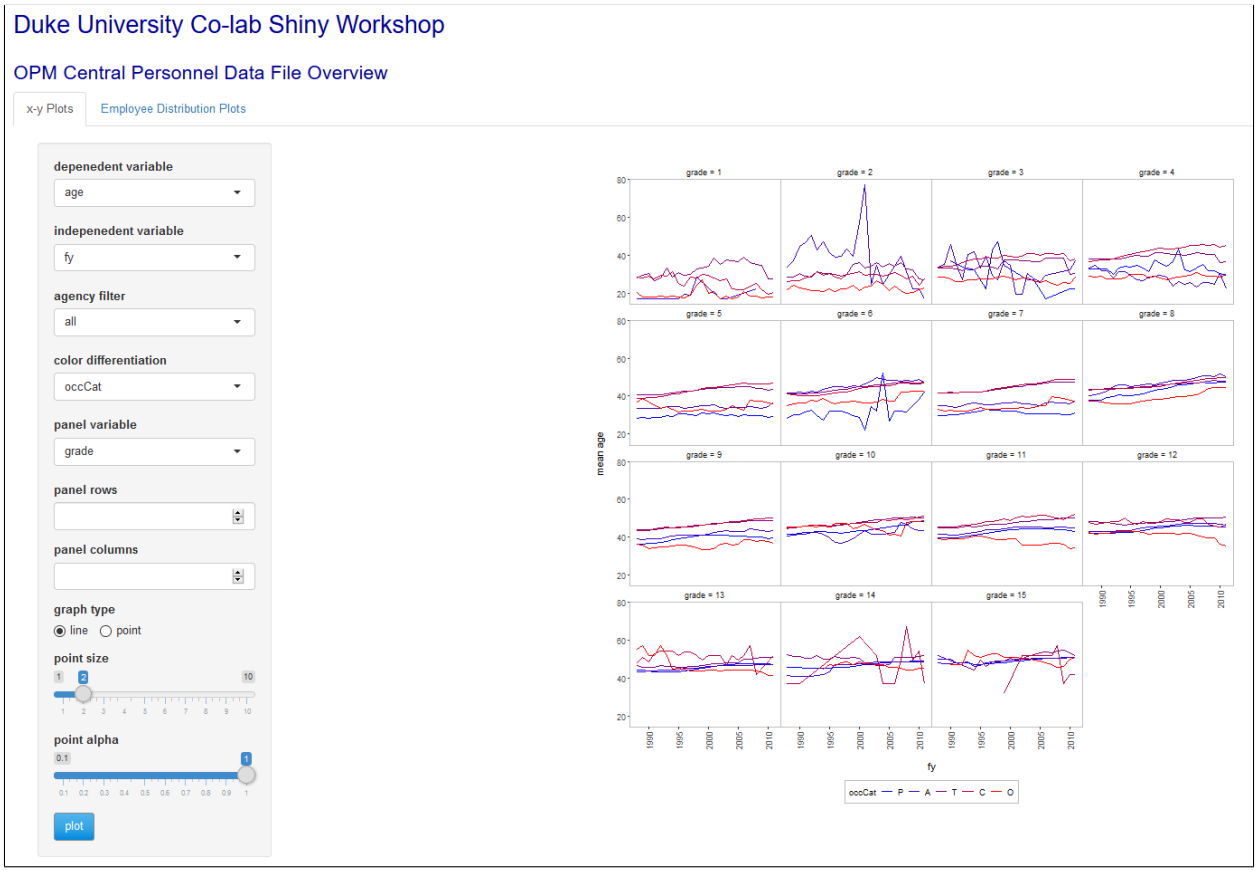


Figure 6: OPM CPDF graph of mean employee age vs. fiscal year, paneled by grade, differentiated by occupational category

8.4 Motivation Graph 4, Increase in Proportion Employees by Grade and Fiscal Year

On the Employee Distribution Plots tab of the app, select `grade` for independent variable and `fy` for panel variable then click the `plot` button. As seen in figure 7, the proportion of employees assigned to higher grades increases with fiscal year.

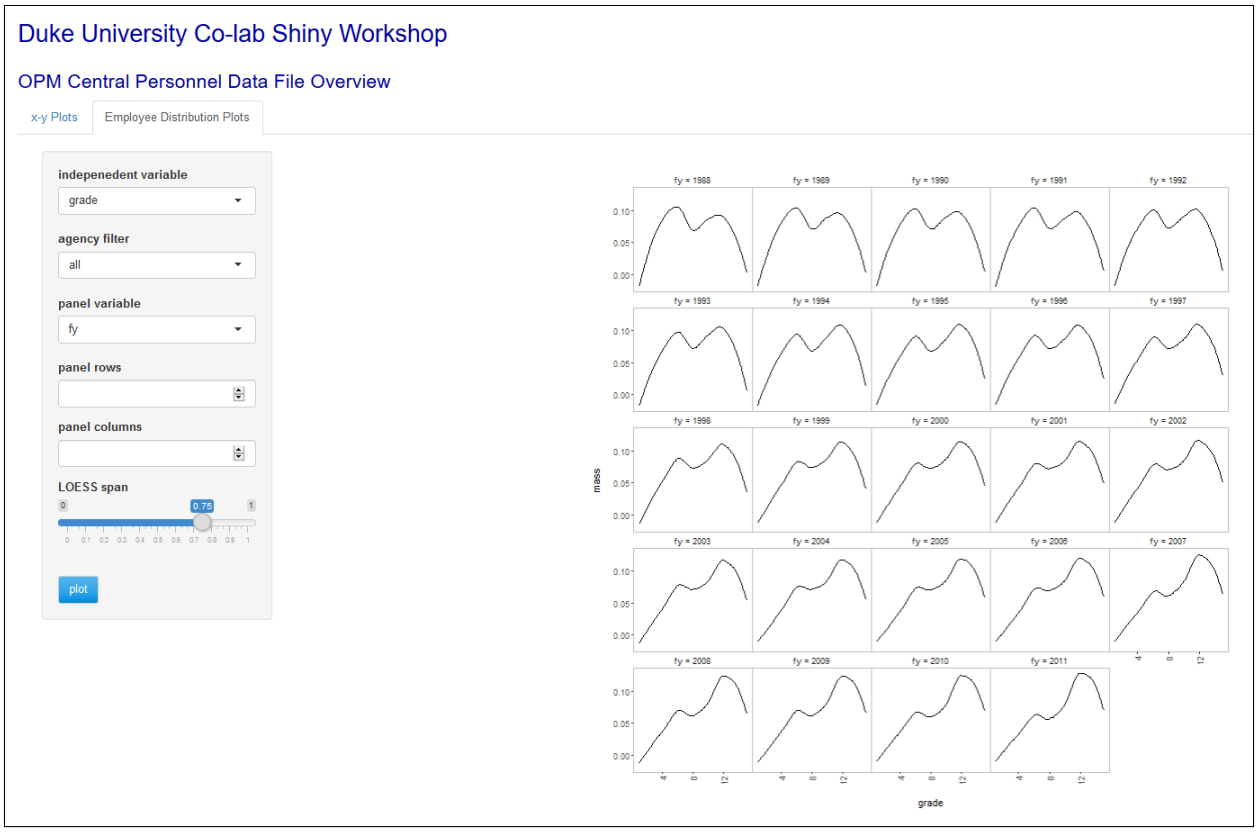


Figure 7: OPM CPDF graph of mean employee age vs. fiscal year

Figures 4, 5, 6, and 7 demonstrate simple, yet effective, exploration of the CPDF data. With little effort, known historical human capital patterns were verified. Experimentation with the on-screen Shiny controls may give the analyst new insight into unknown trends or covariate relationships within an area being studied. The opportunities presented through dynamic data exploration using Shiny and `ggplot` should be apparent from these examples.

9 Development of a Shiny App With Data Tables and Plots

To analyze CPDF observations, we might aggregate a dependent variable by joint levels of a combination of independent variables and produce a summary table with one row per joint level of the independent variables. If done within a Shiny app, using the `DT` (data tables) package, clicking a table row can signal an event such that the app, with identifying row information, can conduct additional, detailed analysis on the subset of observations associated with the row. For example, a table of median pay by agency and occupation can be presented with one row per agency, occupation combination. When a row is selected, we can advance the user to a plot configuration screen where various relationships between pay, fiscal year, grade, education, and age can be viewed for the selected agency, occupation subset. This approach separates the analysis into two phases: a summary phase to present the structure of the data and an investigation phase to explore relationships within key data subsets identified in the summary phase.

9.1 Analysis Script in R (V1)

Before developing a Shiny app for this analysis, we will examine a basic R script that aggregates observations for fixed independent and dependent variables and plots covariate relationships for a fixed set of additional variables. Once an R script is designed, implemented, and tested, converting it a Shiny app requires little modification to the R instructions. The challenge in developing an effective Shiny app is in superimposing the Shiny structures to transform static R variables into dynamic, reactive variables that reflect a variety of user inputs and analysis requirements. For this script, and subsequent Shiny apps, we will use a random sample of observations from the Buzzfeed data, as opposed to aggregated values.[1] Table 1 lists the variables included in the data.[2]

Table 1: Buzzfeed OPM data set

Column	Description
PseudoID	unique (OPM randomly assigned) employee ID
FY	U.S. federal government fiscal year
Agency	federal agency employed (synthetically generated for workshop)
Grade	general schedule (GS) grade
OccupationalCategory	occupational category
Occupation	occupation
Age	employee age (five year increments, noised induced by OPM)
EducationYears	years of education
BasicPay	adjusted basic pay, in 2011 $U.S.

Restrictions on observations used in our analysis are:

- FY between 1988 and 2011
- WorkSchedule=F
- PayPlan=GS
- Grade between 01 and 15
- OccupationCategory in P, A, T, C, O
- EducationLevel between 01 and 22
- AdjustedBasicPay $>$ 10 (thousands per year)
- Limited to top five agencies (left two positions) by observation frequency

[1] For information on Buzzfeed and their hosting of these data, see https://www.buzzfeednews.com/article/jsvine/sharing-hundreds-of-millions-of-federal-payroll-records

[2] Additional information on CPDF data elements is available in the 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

Script location: App/V1/CPDF-Tables-1.r (App was downloaded in section 7)

Features and considerations include (line numbers are approximate):

- (lines 43-60) One of two sample CPDF files is read, either a random sample of a quarter of the entire data set or a random sample of 100,000 records (the smaller 100,000 record sample is useful for script development)
- (lines 64-89) A common `ggplot` theme is defined once and specified on all `ggplot` calls (this avoids redundant specification of themes)
- (lines 95-99) Dependent and independent aggregation variables are specified
- (lines 101-121) A table (data frame labeled `aggdat`) of aggregated mean and quartiles is constructed using specified dependent and independent variables
- (lines 124-163) An alternative aggregation method truncates `agency` and `occupation` to two positions (to achieve a high level of aggregation)
- (line 171) One row is selected from `aggdat`
- (line 174) An index vector is constructed that subsets the disaggregated data (all observations) corresponding to the selected `aggdat` row
- (line 177) An independent (x-axis) variable for the graph is specified (in `gindepVar`) that is different from any used in the aggregation step
- (lines 182-185) A data frame (`gdat`) is prepared to be used as the data source of `ggplot()`
- (lines 183, 185) `gindepVar` is coerced to a factor to avoid the problem of `ggplot()` producing a single element x-axis when x (`gindepVar`) is continuous
- (line 185) Occupational category, if specified as a graphical independent variable, is coerced to a factor with levels specified in the standard order P, A, T, C, O, for proper alphabetic appearance
- (lines 188-195) The plot (`g`) is constructed in a step-wise manner, as a template for applying further conditional geoms and appearance features
- (line 197) The plot is rendered
- (lines 199-276) A more developed plot is prepared that is closer to what is needed for the Shiny app. Features include:
  - Faceting based on a specified panel variable (`panelVar`)
  - Specification of the number of panel rows or columns to arrange (`panelRows`, `panelCols`)
  - Ability to display points for individual observations (`pointDisplay`)
  - Ability to specify point transparency (`pointAlpha`) to diminish the effect of point overlay
  - Ability to specify a point coloration variable (`diffVar`) to aid in distinguishing categories of observations
  - (line 250) Jitter (`geom_jitter()`) is added in the x-dimension to diminish the effect of point overlay
  - (line 256) The point color aesthetic is implemented by direct editing of the list produced by `ggplot()`, but only if `diffVar` is non-empty. This is an example of conditional geom modification.
  - (line 257) Actual point category colors are generated using a `colorRampPalette` from blue to red
  - (line 262) Default display of outlier points is suppressed in `geom_boxplot()`
  - (line 263) Error bars (`stat_boxplot()`) are included to indicate the inter-quartile range of each independent variable level

  - (lines 266-268) A custom function is defined to label facet panels
  - (line 272) The previously prepared theme (`ggTheme`) is used to control overall plot appearance

- (line 279) Examining the structure of a composed `ggplot()` list reveals the inner structure of a plot and which list elements affect particular plot features, along with an idea of the behavior achieved by using certain parameter values (examination of `g` revealed the element `g[["layers"]][[length(g[["layers"]])]][["mapping"]][["colour"]]`, used on line 256 to apply the color aesthetic for `diffVar`)

Figure 8 is an example plot from this analysis, showing the distribution of grade by joint combinations of education and fiscal year, for employees in agency Health and Human Services with two-position occupation code 02 (agency and occupation were selected from the initial aggregation data frame - there is no data table yet from which to review and select rows - Shiny is needed for that). It reveals an expected general increase in employee grade vs. education and a general increase in grade throughout the study period, regardless of education (although this may confounded with the visible reduction in employees in clerical and other occupational categories).

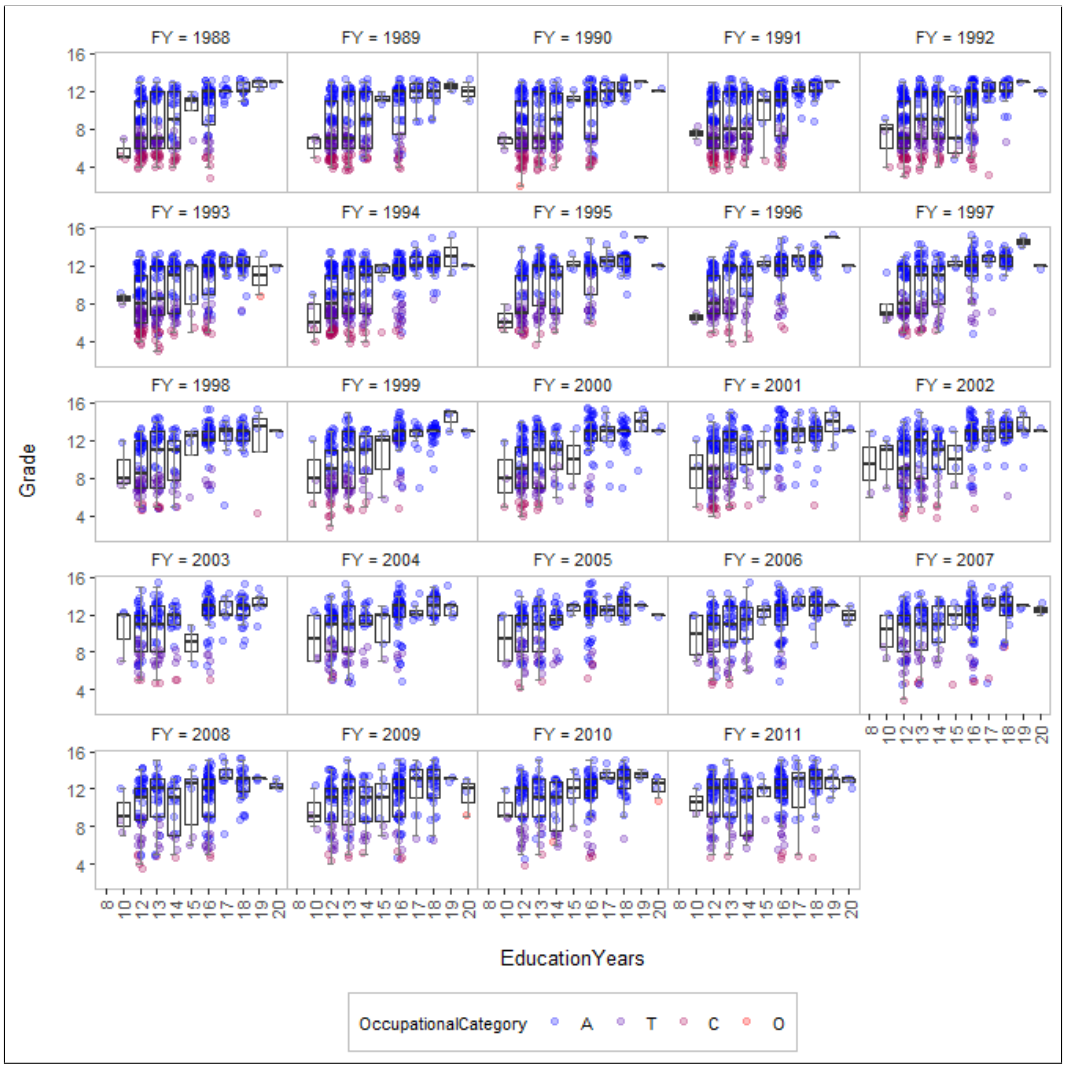
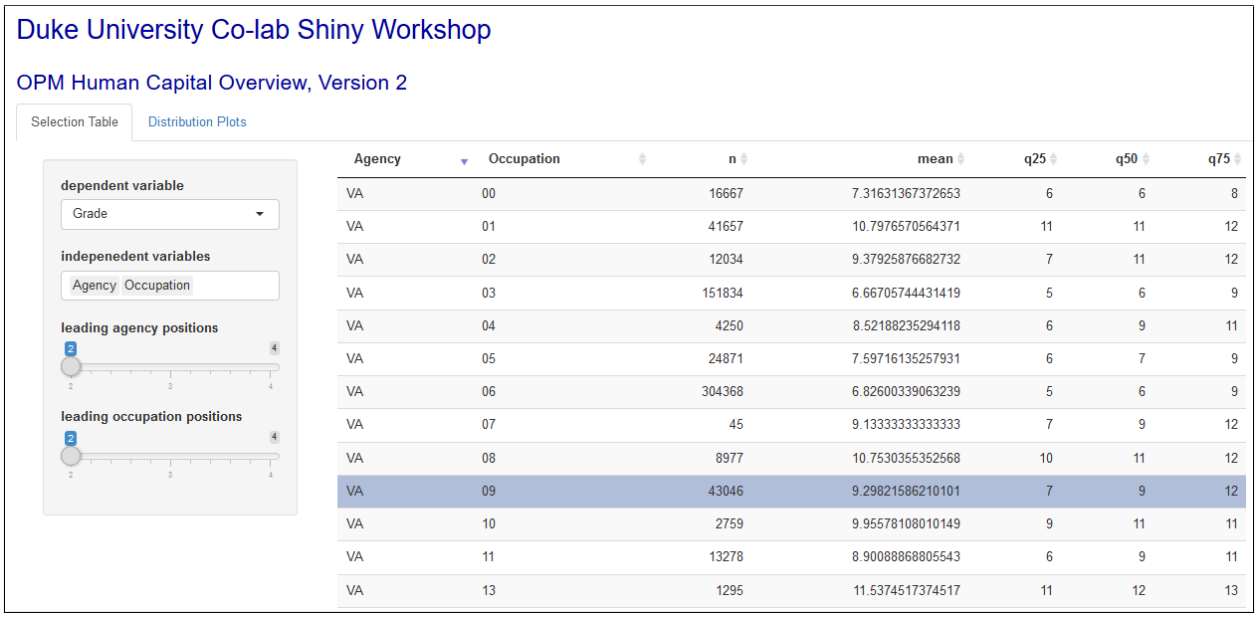


Figure 8: Change in employee grade, given education, along with change in distribution of occupational category, Health and Human Services, occupation code 02, FYs 1988-2011

9.2 Shiny App with Basic Table and Plot Features (V2)

The analysis of section 9.1, implemented in R, reveals important features of federal employee human capital. To expedite and broaden the scope analysis of these data, we will develop a Shiny app that produces a summary table for user selected dependent and independent variables. From the table, a row can be selected to subset observations for detailed plotting. Plot configuration includes controls to specify variables for the graph independent (x-axis) variable, paneling, and point category coloration. Additional controls are included for panel layout (rows/columns), point size, point jitter, and point transparency. Figure 9 is an example screen-shot of the aggregated summary table tab of the V2 app. Row highlighting indicates an observation subset (agency VA and occupation 09 here) selected for plotting. Figure 10 is an example screen-shot of the plotting tab, with a plot composed from the observation subset corresponding to the row selected in figure 9, using education as independent variable, fiscal year as paneling variable, and display of points with color differentiating occupational category. Features of this app include:

- Use of a tab panel with individual tabs for aggregation and plot controls[3]
- Use of cascading style sheet for HTML formatting[4]
- Immediate reactive controls for preparing the aggregation table (which is efficient)
- An action button for plot rendering (plotting time may be lengthy and, with immediate reactivity, rendering would be repeated for each change in a reactive control)
- Multiple selection of aggregation variables
- Functions for aggregating data and preparing plots
- Table row selection for subsetting source data to be plotted[5]



Agency	Occupation	n	mean	q25	q50	q75
VA	00	16667	7.31631367372653	6	6	8
VA	01	41657	10.7976570564371	11	11	12
VA	02	12034	9.37925876682732	7	11	12
VA	03	151834	6.66705744431419	5	6	9
VA	04	4250	8.52188235294118	6	9	11
VA	05	24871	7.59716135257931	6	7	9
VA	06	304368	6.82600339063239	5	6	9
VA	07	45	9.13333333333333	7	9	12
VA	08	8977	10.7530355352568	10	11	12
VA	09	43046	9.29821586210101	7	9	12
VA	10	2759	9.95578108010149	9	11	11
VA	11	13278	8.90088868805543	6	9	11
VA	13	1295	11.5374517374517	11	12	13

Figure 9: Screen-shot of V2 CPDF Shiny app, aggregated summary table tab

[3]For additional information on tab panels, see https://shiny.rstudio.com/reference/shiny/0.14/tabsetPanel.html
[4]For more information on using file-based CSS, see https://shiny.rstudio.com/articles/css.html
[5]For additional information on Data Tables and row selection, see https://rstudio.github.io/DT/shiny.html

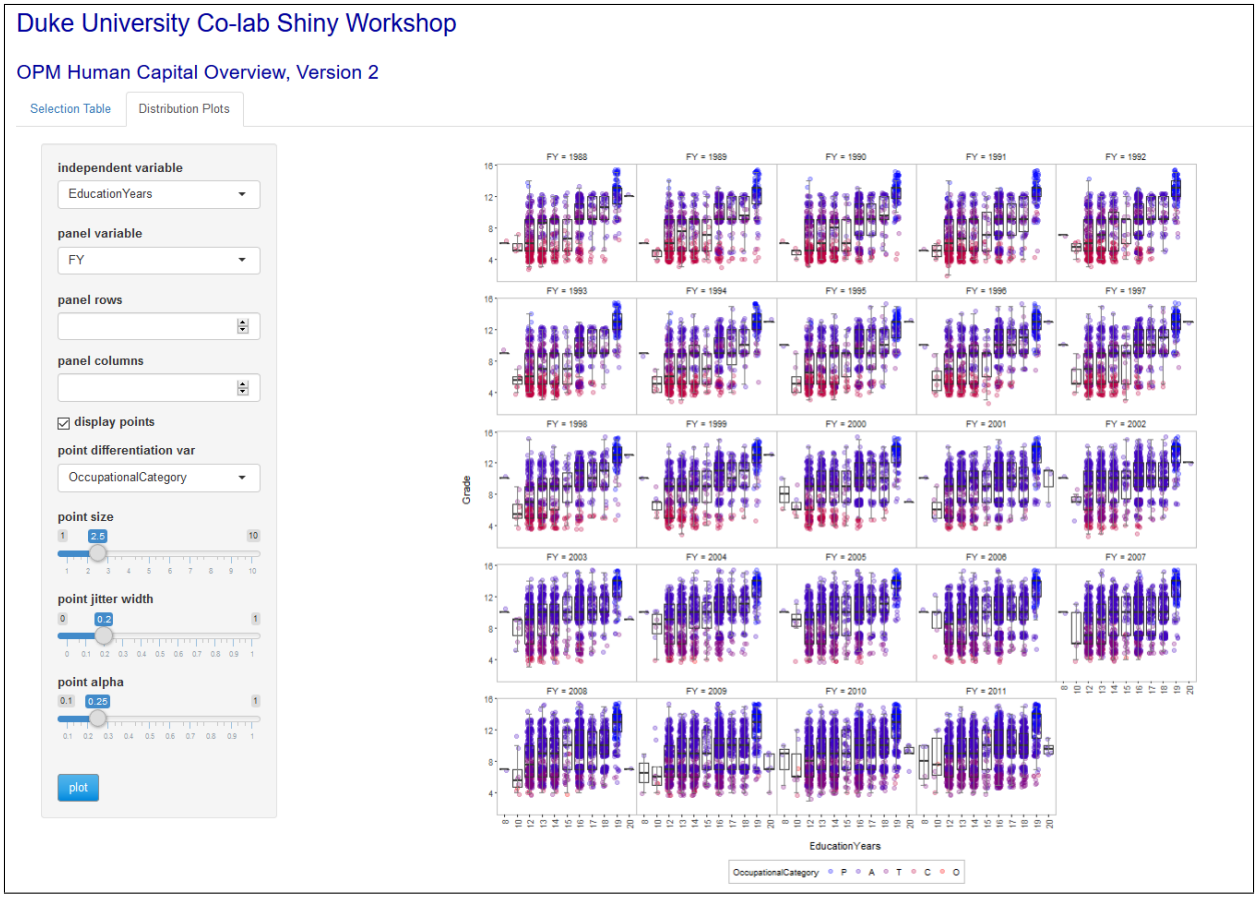


Figure 10: Screen-shot of V2 CPDF Shiny app, plot configuration tab

Script location: App/V2/CPDF-Tables-2.r (App was downloaded in section 7)

Feature implementation and considerations include (line numbers are approximate):

- User interface (file `ui.r`)
  - (lines 59-72) A subset of observations is assembled corresponding to approximately one fourth of the employees represented in the data
  - (lines 74-78) Record cleaning is accomplished
  - (lines 82, 151) An agency selection list is constructed to be used in an agency selection input field (agency selection is disabled in this version)
  - (line 116) A `tabsetPanel` is configured for two tabs: one for the aggregation table, one for plotting
  - (lines 118-140) The aggregation and selection tab (`t1`) is defined
  - (lines 121, 137) Two columns are defined: one of width 2 and one of width 10. There are twelve columns available in the UI.
  - (line 123) A `sidebarPanel` is defined to contain the user input controls. It is defined within a column of width 12, the entire width of the column in which it is defined.
  - (line 127) Multiple independent variables are simultaneously selectable (`t1IndepVar`, `multiple=T`). Note that the selection list is static. It can also be composed from the column names of the input data (as with `t2AgencyFilter` and `agencyList`).

- (line 138) The data table object (`t1Table`) is defined (selecting a row will activate the plot tab)
- (lines 42-196) The plot configuration and review tab (`t2`) is defined
- (line 171) An action button (`t2ActionPlot`) is defined for initiating plot construction
- (line 182) The plot object (`t2Plot`) is defined (this is where plots will be displayed)
- (line 190) A text output line (`t2Msg`) is defined for messaging

- Server (file `server.r`)
  - (lines 28-52) A `ggplot` theme is defined to control plot appearance
  - (lines 54-133) A function is defined to compose and render the aggregation table
  - (line 60) It is confirmed that a dependent and at least one independent variable are selected. If violated, then (lines 128-129) the table is set to NULL (it disappears) and a message is displayed).
  - (lines 65-77) Observation subset indices are composed for each combination of levels of the independent variable(s) specified on the table tab (`t1IndepVar`). The result (`iagg`) is a list with one element per independent var level combination and is saved as a global variable to be made available outside of current function. Note the use of `lapply()` for the `by()` clause of `aggregate()`. A list is required here and it will contain one element per independent variable. This is responsible for observation subsetting.
  - (line 79) Subset index validation is accomplished
  - (lines 83-94) Observation count, mean, and quartiles are aggregated for each index subset.
  - `apply()` proceeds through each set of indices in `iagg` and produces a list of data frames that are then flattened into a single data frame (`aggdat`).
  - (line 96) `aggdat` validation is accomplished
  - (lines 100-110) A data table is constructed and rendered (in `t1Table`), using `aggdat` results. Features include (this might be a good time to review the `dataTables` and `renderDataTable()` documentation listed in section 3):
    - Fixed page length (100) with no page length adjustment object (`bLengthChange=F`)
    - The global table search tool is disabled (bFilter=F)
    - Single row selection (multiple rows are possible, yielding a list that grows with selection, until it is reset)
    - Automatic column width assignment (based on max width of contained data)
    - The order of the second column is set to ascending (note that col IDs are 0-based)
  - (lines 112-129) Messages are displayed to the user if no observations exist corresponding to the input subset parameter values
  - (lines 135-232) A function is defined to generate a plot based on a selected `aggdat` table row and configuration parameter values taken from the plot configuration tab (`t2`). Features include:
    - (lines 142-146) Dependent, independent, and aggregation variables are validated (specification of a single variable as having multiple roles indicates an invalid plot structure). Invalidation causes a NULL plot to be returned, along with display of an error message (lines 220-232)
    - (line 149) The observation subset indices corresponding to the selected row are retrieved
    - (lines 151-174) A plot source data frame (`gdat`) is composed from the subset observations. Variables are converted to factors as needed and occupation category is order according to standard.
    - (lines 177-213) A composite box plot with overlayed points (for individual observation sin the data subset) is composed
    - (lines 181-182) Points are displayed with x-jitter to aid in distinguishing multiple points at shared coordinates. Note the use of `aes_string()`, since the x and y coordinates are contained in `gdat` indicated by `indepVar` and `depVar` (use of `aes()` would cause a search of `"indepVar"` and `"depVar"` in the column names of `gdat`).

    - (lines 183-192) The color differentiation aesthetic is assigned by direct editing of the `ggplot()` result list. This is an alternative to complex use of various `aes()` calls when aesthetics are conditional (two conditional aesthetics, one for color and one for size, would require four separate `aes()` calls embedded in if-then-else statements; three aesthetics would require eight statements, etc.).
    - (line 195) The box plot is generated with outliers suppressed (perhaps this should be done only when points are requested)
    - (line 196) error bars are included (although specified with fixed color and width, these can be made adjustable with additional on-screen controls)
    - (lines 198-208) Panels are produced, one for each level of a specified variable. A custom labeler function is specified.
    - (lines 210-213) Y-axis labels are formatted (thousands) and a previously prepared theme is applied
    - (line 216) The plot is rendered (displayed in `t2Plot`)
  - (lines 234-246) An action event is defined to respond to changes in any of the tab 1 input variables. This renders an aggregation table based on current tab 1 input values
  - (lines 248-270) An action event is defined for `t1Table` row selection
  - (line 257) The selected row ID is saved in a global variable (`t1SelectedRow`) so that other functions may reference the currently selected row
  - (line 261) A call to `t2RenderPlot` is made to compose and display a plot using the selected table row ID and the current values of on screen objects. Note that the values of Shiny variables (preceded by `input$`) are passed as parameters instead of directly referencing Shiny variables within `t2RenderPlot` to avoid any possible attempt to create a reactive environment, where `t2RenderPlot` might be called when a Shiny variable is modified (this is not supposed to occur, but it does)
  - (line 267) The plot tab (`t2`) is made active, so that the user can view and configure the resulting plot
  - (lines 272-289) An action event is defined for the `t2ActionPlot` button. This renders a plot using the previously specified table row (`t1SelectedRow`) and current tab 2 input values
  - (lines 245, 269, 288) `ignoreInit=T` instructs that an observe is not to be triggered during script startup, when reactive values change from NULL to their default values

Observations from review of the aggregation table:

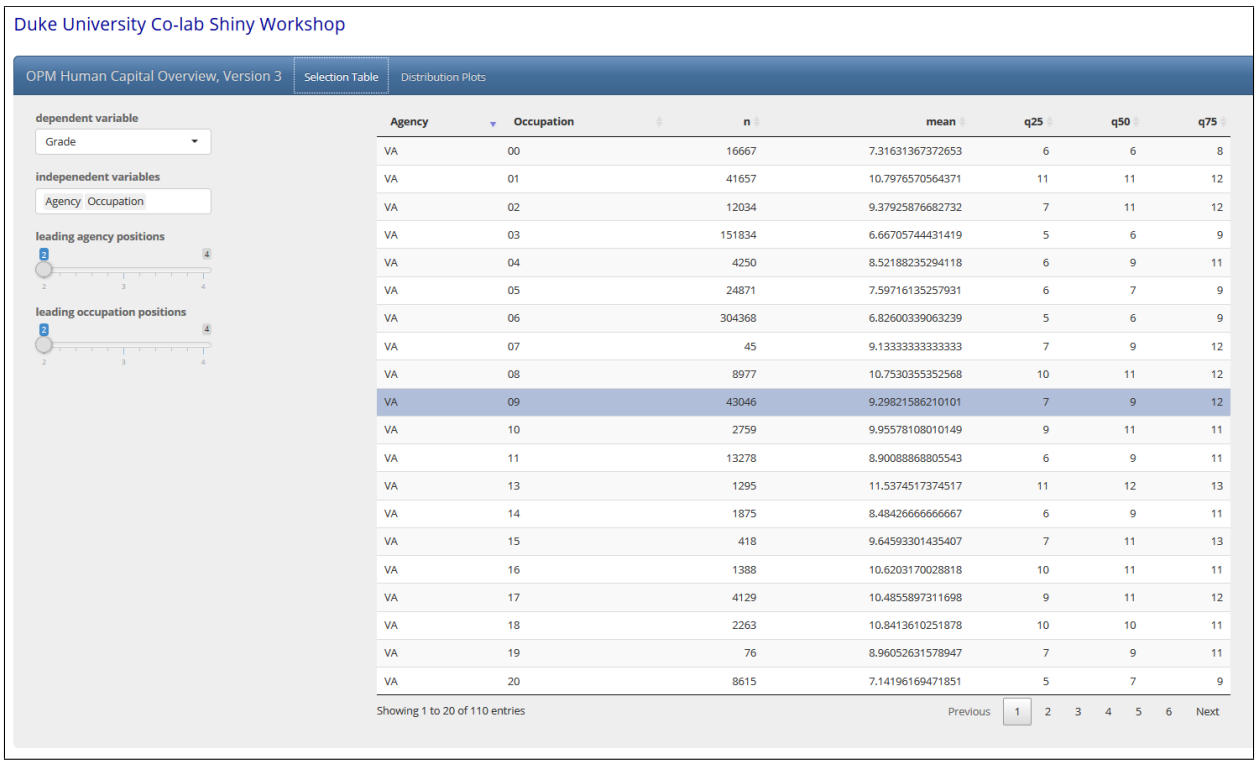
- Aggregation of grade by fiscal year reveals a pattern of increased grade by year, with median grade exhibiting greater increase than mean grade (does this indicate a widening of grade distribution with a tail forming for upper grades?)
- Aggregation of age by fiscal year reveals a pattern of increased age by year, indicating an older (more experienced?) workforce at the end of the study period
- Aggregation of education by fiscal year indicates an increase in mean and median education over the study period
- Aggregation of grade by agency and occupational category indicates that VA employees tend to have lower grade than employees of similar category in other agencies

How can the plot tab be used to investigate the above observations to reveal differences between agencies, occupational categories, fiscal years, and so forth? Note that it would be interesting to explore a data table of grade aggregated by occupational category, then plot with fiscal year as the independent variable and agency as a panel variable. However, although our app allows truncation of agency on the aggregation tab, it does not on the plot tab. Therefore, panels are constructed of individual plots using four position agency codes, giving a somewhat detailed view. Yet, individual agencies with general increase in grade over the period are easily identified for each selected occupational category, which is informative.

9.3 Shiny App with Updated Appearance Using Navigation Bars and Themes (V3)

To improve the appearance and messaging capability of our app, we will replace the basic `tabsetPanel` with a `navbarPage`, employ themes, and implement modal dialog windows. V2 and V3 app functions (`server.r`) are very similar. The primary difference in these versions is their appearance. Figures 11 and 12 are screen-shots of the Selection Table and Distribution Plots tabs of the V3 app. Features implemented in this version include:

- `navbarPage` layout[6]
- Themes and the theme selector[7] [8]
- In-line CSS styles[9]
- `showModal()` and `modalDialog()`[10]



Agency	Occupation	n	mean	q25	q50	q75
VA	00	16667	7.31631367372653	6	6	8
VA	01	41657	10.7976570564371	11	11	12
VA	02	12034	9.37925876682732	7	11	12
VA	03	151834	6.66705744431419	5	6	9
VA	04	4250	8.52188235294118	6	9	11
VA	05	24871	7.59716135257931	6	7	9
VA	06	304368	6.82600339063239	5	6	9
VA	07	45	9.13333333333333	7	9	12
VA	08	8977	10.7530355352568	10	11	12
VA	09	43046	9.29821586210101	7	9	12
VA	10	2759	9.95578108010149	9	11	11
VA	11	13278	8.90088868805543	6	9	11
VA	13	1295	11.5374517374517	11	12	13
VA	14	1875	8.48426666666667	6	9	11
VA	15	418	9.64593301435407	7	11	13
VA	16	1388	10.6203170028818	10	11	11
VA	17	4129	10.4855897311698	9	11	12
VA	18	2263	10.8413610251878	10	10	11
VA	19	76	8.96052631578947	7	9	11
VA	20	8615	7.14196169471851	5	7	9

Figure 11: Screen-shot of V3 CPDF Shiny app, aggregated summary table tab

[6] For more information on the `navbarPage`, see https://shiny.rstudio.com/reference/shiny/1.0.4/navbarPage.html

[7] For more information on themes, see https://cran.r-project.org/web/packages/shinythemes/shinythemes.pdf

[8] For more information on the theme selector, see https://rdrr.io/cran/shinythemes/man/themeSelector.html and https://shiny.rstudio.com/gallery/shiny-theme-selector.html

[9] For more information on in-line CSS and particular elements modified in the script, see https://cran.r-project.org/web/packages/shinythemes/shinythemes.pdf and https://bootswatch.com/spacelab/

[10] For more information on modal and dialog windows, see https://shiny.rstudio.com/reference/shiny/1.0.2/showModal.html and https://shiny.rstudio.com/reference/shiny/0.14/modalDialog.html

Figure 12: Screen-shot of V3 CPDF Shiny app, plot configuration tab

Script location: App/V3/CPDF-Tables-3.r (App was downloaded in section 7)

Implementation considerations are discussed in the following sections by feature. Line numbers preceded by "u" correspond to lines in `ui.r`, those preceded by "s" correspond to lines in `server.r` (all line numbers are approximate):

9.3.1 `navbarPage`

- (u104) The `navbarPage()` function replaces `tabsetPanel()` used in V2
- (u105) Text placed in the title parameter ("OPM Human Capital Overview, Version 3") appears beside the tab titles (see u220, below)
- (u105) Inverse=T instructs to display light tab titles on a dark background
- (u123, 158) `tabPanel` specifications are constructed as in the V2 app
- (u154, 206) The style for the div containing tabs is modified to eliminate space between the title bar and main content (top margin), round corners, and change the background colors (experiment with these)
- (u212-244) A `tabPanel` is defined within a hidden section, so that the tab title and any contained elements are not visible

- (u99, 248) The `navbarPage` lacks a width parameter, so that width (actually of the entire page) is specified for the div in which it is defined (experiment with this!)
- (s112) Table page length is configured in the `renderDataTable()` call and is fixed at 20 rows (experiment with this!)
- (s112) User specification of table length and filtering are disabled (create reactive variables to adjust these!)

9.3.2 Themes and the theme selector

- (u118) A theme is specified for the entire page (all elements)
- (u95) Calling the `themeSelector()` function of the `shinythemes` package caused the theme selector to be displayed in the user's browser. Figure 13 is an example theme selector. Selecting a theme from the pull-down list causes the theme to be immediately applied to all elements, regardless of individual element theme and style specifications (experiment with this!).

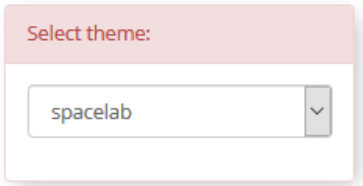


Figure 13: Shiny theme selector

- (u121) A theme is specified for the `navbarPage`, overriding the initial theme for certain elements within (experiment with this!)

9.3.3 In-line CSS styles

- (u214-245) A hidden panel contains in-line style specifications that override those from the specified, current theme. In many cases, styles can be specified outside of a container element (near the initial theme specification seems like an appropriate location), but with a `navbarPage`, experience has shown that, in order to be recognized, they must be specified within the element and after any theme specification.
- (u215-239) Styles are specified for various on-screen elements, including the navbar (where titles and tabs appear), buttons, and modal windows. Each style specification modifies the appearance of associated elements according to the CSS parameters modified. How do we know which parameters to modify for a particular element? Our browser can give us important clues. Say we want to modify the radius of the top corners of the navbar. In your browser (Firefox here) , while executing the app, hover over the navbar and, from the mouse right-button options, select Inspect Element. A sub-window as appears in figure 14 should appear. Hover over the element definitions in this window until the navbar is highlighted (figure 15). Click on the definition being hovered over (16). In the details section of the inspection window (on the right), identify style parameters that appear to indicate radius ((17). Several radius related style parameters appear with values that can be altered and names that can be copied into our script style commands (experiment with these!).

Figure 14: Firefox element inspection window

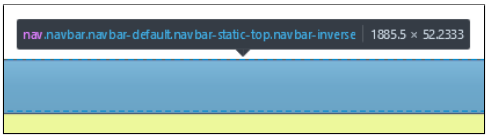


Figure 15: Hovering over browser element definitions to highlight a desired element

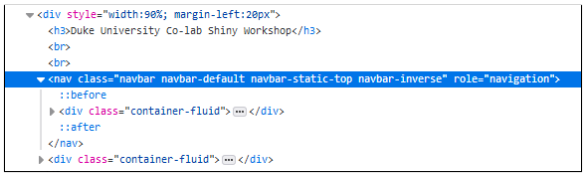


Figure 16: Selecting definition of desired element

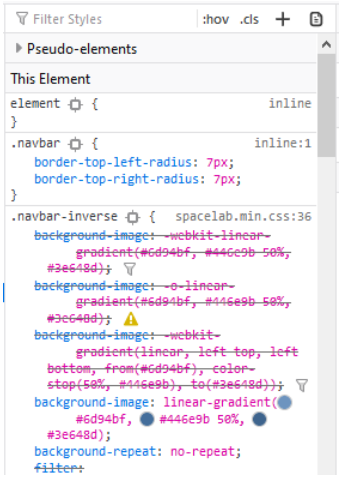


Figure 17: Style details of selected element

- (u188) A style can be specified for certain objects (a button here) that overrides values for specified style parameters (experiment!)

9.3.4 `showModal()` and `modalDialog()`

- (s58-61) A messaging function is defined such that, when it is called, a modal window is displayed. An example modal window appears in figure 18. The window remains on screen until dismissed by the user.

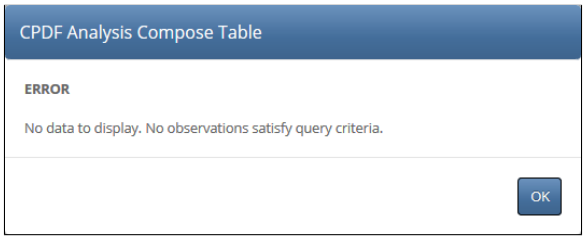


Figure 18: Modal window used for messaging

- (s123, 130, 224, 231) The message window function is called for specific data integrity conditions that are tested
- Note how errors reported after applying intentional data verification differs from unhandled program errors. For an example unhandled error, start the app, enable the Distribution Plots tab and click Plot. The browser window should turn gray and your RStudio console should display a message as in figure 19. Error handling will be treated in more detail in a later version of the app.

```
Warning: Error in t2RenderPlot: object 'iagg' not found
  73: t2RenderPlot [C:\Projects\Duke\Co-lab\Shiny-Spring-2021\Session-1-2-DataTab
les-Plots\App\V3/server.R#152]
  72: observeEventHandler [C:\Projects\Duke\Co-lab\Shiny-Spring-2021\Session-1-2-
DataTables-Plots\App\V3/server.R#288]
   1: runApp
```

Figure 19: Unhandled error in Shiny script

9.4 Shiny App with Draggable Panels (V4)

One deficiency of the `tabsetPanel` and `navbarPage` versions of the app is that the user is able to see only one tab at a time. Observations are queried and an aggregation table composed on one tab then additional variables are chosen and a plot is configured on a separate tab. While configuring a plot, the user must either remember which subset (row) was selected from the table or refer back to the table tab for that information. An alternative design is to use draggable panels, such that all are visible simultaneously and able to be repositioned as desired by the user. Figure 20 is an example screen-shot of the V4 app, which employs draggable panels. Individual panels are identical to those of V3, but all now appear in a single window. Panels are repositioned by clicking and holding the title bar, dragging to a desired position, then releasing the mouse button. V4 reactive variables and functions are identical to those of V3 so that, although being different in appearance, the two versions operate uniformly. This is an example of slightly different `ui.r` configurations giving different user experiences, but with a common `server.r` script.

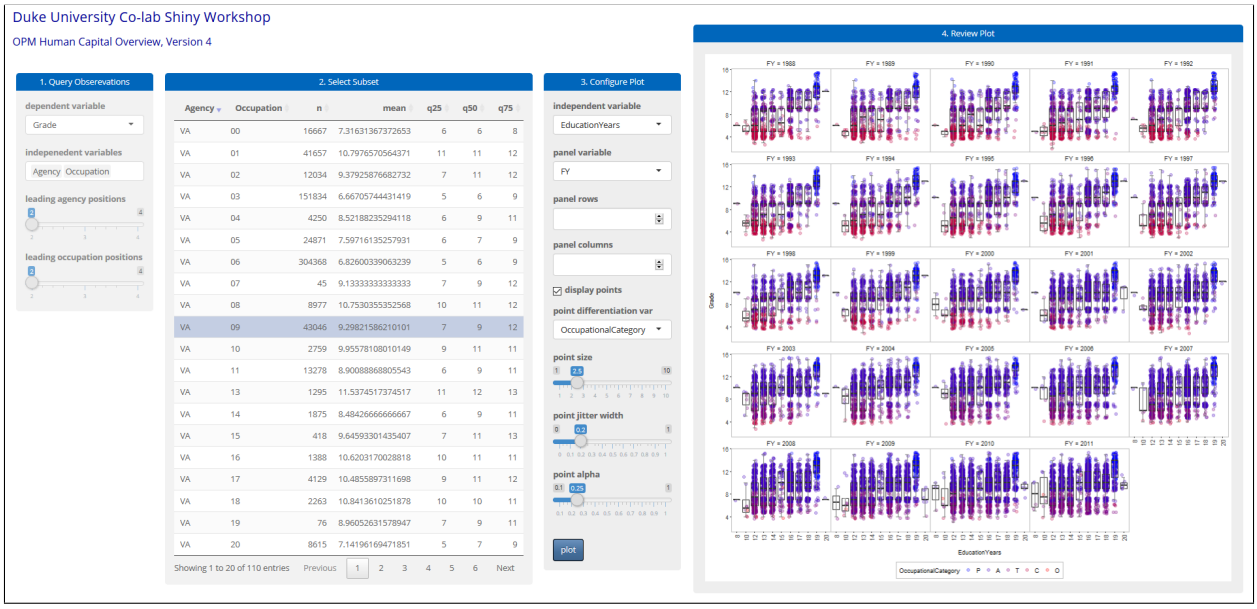


Figure 20: Screen-shot of V4 Shiny CPDF Shiny app, independent draggable panels

Script location: App/V4/CPDF-Tables-4.r (App was downloaded in section 7)

Feature implementation and considerations include. Line numbers preceded by "u" correspond to lines in `ui.r`, those preceded by "s" correspond to lines in `server.r` (all line numbers are approximate):

- (u109) Since all panels are intended to be repositioned, there are no margins (style) specified for the primary div
- (u115, 139, 151, 189) Each panels is defined within an independent draggable `fixedPanel` construct. All panels share a common top margin and each has a unique left margin to avoid initial overlap.

9.5 Shiny App for Public Deployment (V5)

The previous versions of the CPDF app demonstrate functioning solutions that utilize important Shiny features. However, they might be classified as those for personal use, as defined in section 4. In V5, we will develop the app to include features that improve user interaction with various data sets, explain site operation, and provide information on related research, making the app appropriate for public use. Figure 21 is a screen-shot of the V5 app before any selections are made. In addition to the V4 panels, this version includes panels for Quick Start, About, and File Upload. Implemented features include the following:

- File upload capability
- A download link for review of sample input file format
- Aggregation table download capability
- Progress indicators
- Dynamic insertion of panels with supplemental project info and links
- Import of HTML for additional project info and links
- Dynamically created reactive elements for prompting user action and control
- Error handling

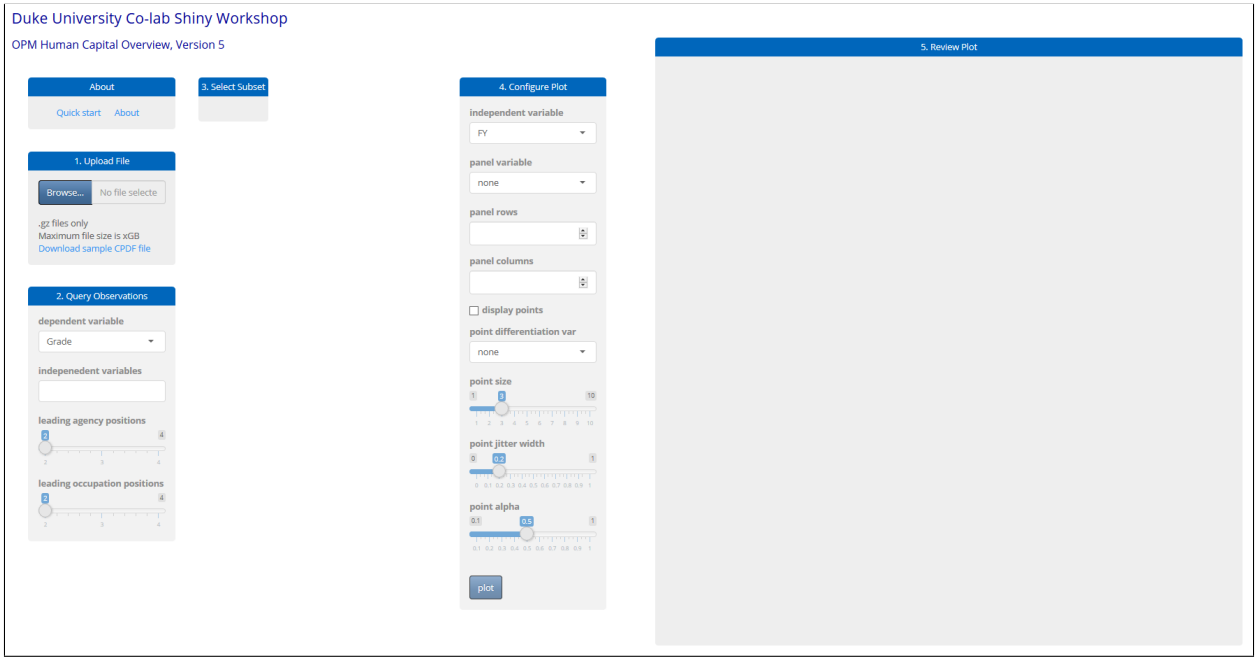


Figure 21: Screen-shot of V5 Shiny CPDF app, with additional panels for project info, file upload, and sample file download

Script location: App/V5/CPDF-Tables-5.r (App was downloaded in section 7)

Implementation considerations are discussed in the following sections by feature. Line numbers preceded by "u" correspond to lines in `ui.r`, those preceded by "s" correspond to lines in `server.r` (all line numbers are approximate):

9.5.1 File Upload

In previous versions of the app, we have loaded a static data set with the following R instructions:

```
# Specify source data file
fn <- c("Data/CPDFSampleDataBuzzfeed-Qtr-Tot.csv.gz", "Data/CPDFSampleDataBuzzfeed-100k.csv.gz")[1]

# Read observations into global environment so that data are available in server module
cpdf <<- read.table(gzfile(fn), header=T, sep=",", strip.white=T)
```

How do we allow the user to examine and process multiple files from a data set? One option is to construct a table of file names (either from a static list or from the output of `dir()`) with selectable rows. Another option is to allow the user to upload a file from a local computer or other available repository. Figures 22 through 25 are screen-shots of the local file upload method implemented in V5. Important operations include:

- (u146, figure 22) The user clicks Browse, presenting the local operating system file browser
- (s420-459, figure 23) A file is selected, causing a change in `input$browseFile` and execution of the associate observe event (note that `input$browseFile` is a single element character vector containing the name of the selected file)
- (s426-427) Five records from the selected file are read
- (s430-443, figure 24) A dialog window is presented with the five initial records and two action buttons: one to accept the file, one to ignore it (the OK button abandons upload, so it might be better labeled "Ignore")
- (s385-414) The user clicks Accept, which executes the observe event defined for `input$fileInputAccept`
- (s395) The entire file (named in `input$browseFile`) is read (into the global environment, so that it is available outside of the observe event)
- (figure 25) The user is informed that the file has been uploaded

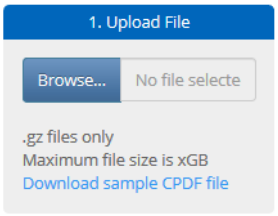


Figure 22: Screen-shot of V5 File Upload panel, prior to file selection

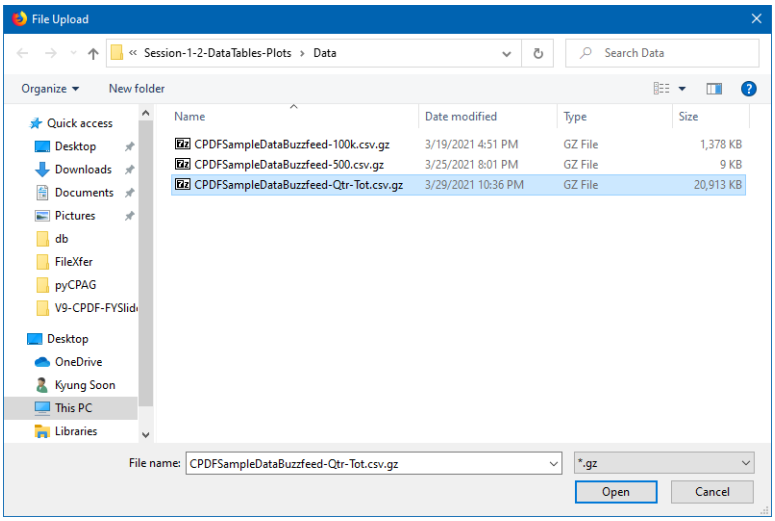


Figure 23: Screen-shot of V5 file selection window provided by the user's local operating system

Figure 24: Screen-shot of V5 record sample review after file selection

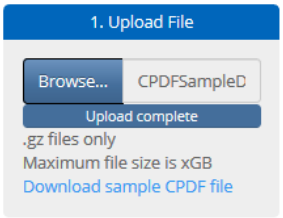


Figure 25: Screen-shot of V5 File Upload panel, after file retrieved

9.5.2 Download Link for Sample Upload File

In order to effectively upload files, visitors to your site must know the file formats that are accepted. One solution is to simply provide a sample file for review. The V5 steps for sample file download are:

- (u149, s-361-378, figure 26) The user clicks the `sampleCPDFdownload` download link, causing the observe function defined for it to be executed
- (s362) A file name, to be used by the local operating system is defined
- (s364) The OS file name is passed as a single element character vector to the download handler function parameter value
- (s368) The sample file is read

- (s371) The sample data are written to the file named in the `file` parameter
- (figure 27) When the download handler function completes, the user is presented with the OS file download window, initialized with the file name specified on s362, and chooses to either save or open the file
- (figure 28) Results of selecting "Open with" from the OS download dialog

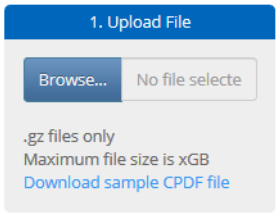


Figure 26: Screen-shot of V5 sample file download link

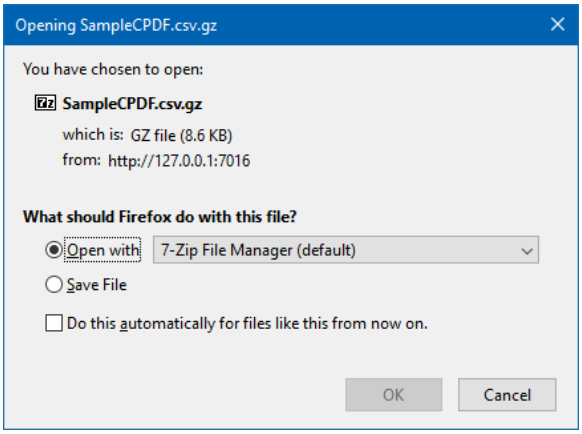


Figure 27: Screen-shot of V5 sample file download, options provided by user's local operating system

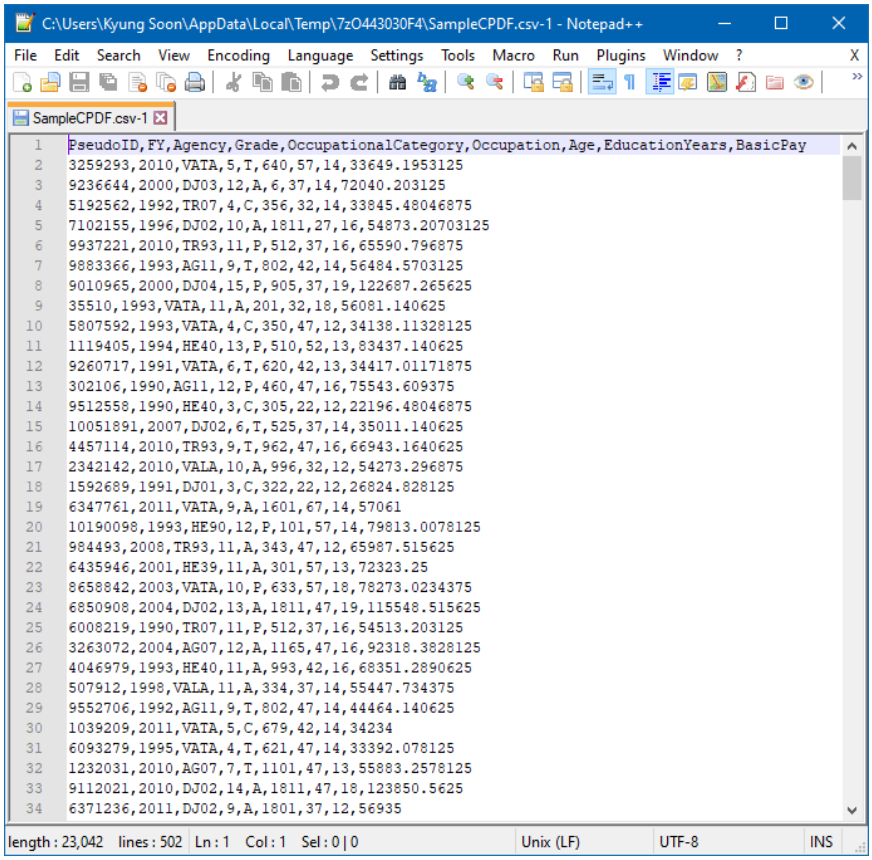


Figure 28: Screen-shot of V5 sample data

9.5.3 Aggregation Table Download

You may want to share source data or semi-processed results with colleagues and reviewers. In the V5 app, we offer download of the table appearing on the aggregation panel. Since the table is composed from a data frame, if we retain that data frame after table construction, then it can be used for additional operations, such as download. The steps for aggregation table download are:

- (u188-207, figure 29) The aggregation and selection table panel is defined
- (u195-202) A download button (`downloadAggregationTable`, s200) is defined within a conditional panel, so that it is visible only when `input.displayTableDownloadButton` is `TRUE` (note that `input.displayTableDown` is a java variable and corresponds to the Shiny variable `input$displayTableDownloadButton`)
- (u184-186) `input$displayTableDownloadButton` is defined within a conditional panel that is never visible (the Shiny variable must be created so that we have a java variable in which to reference)
- (s123) `input$displayTableDownloadButton` is set to `TRUE` when an aggregation table is successfully constructed, otherwise `FALSE`
- (s127) The data frame from which the aggregation table is composed (`aggdat`) is saved in the global environment to be made available for download, later
- (s326-338, figure 30 The user clicks the download button and is given options for viewing or saving the `aggdat` data frame. Note that `write.table()` controls the downloaded file format. Other valid R file output functions can be used for this operation.

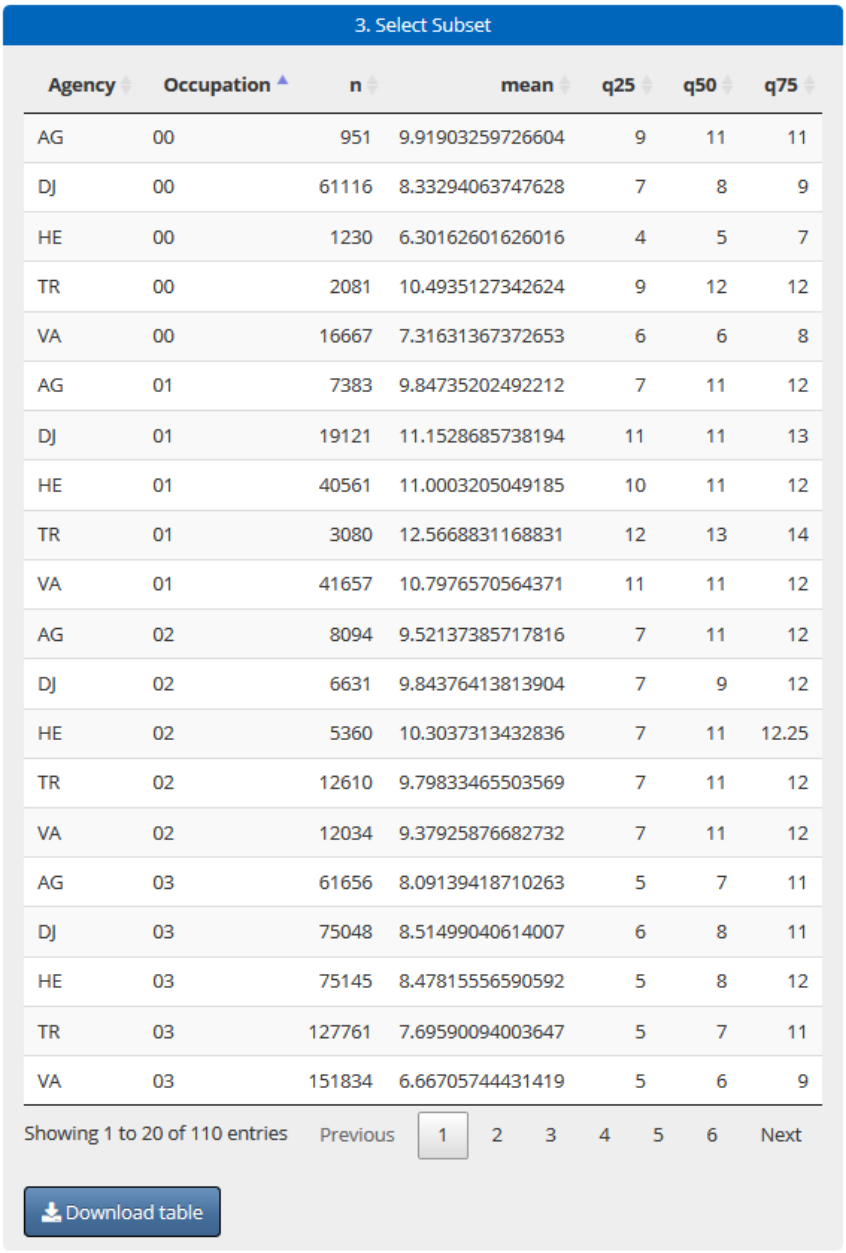


Figure 29: Screen-shot of V5 aggregation table with download button

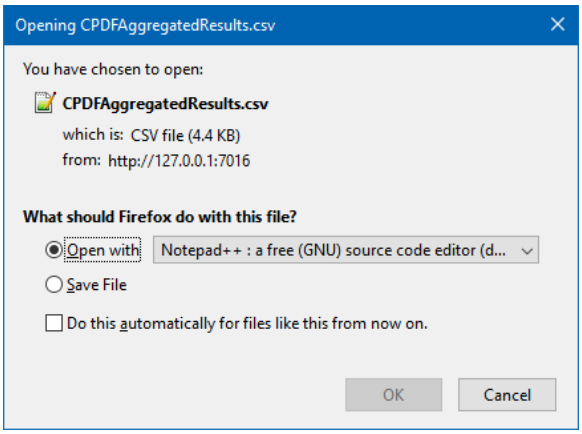


Figure 30: Screen-shot of V5 aggregation table download, options provided by user's operating system

9.5.4 Progress Indicators

Presenting an indication of progress during lengthy operations can aid the user in estimating time to completion and inform that computation is in-progress, as opposed to being complete with indeterminate output. V5 includes progress indicators as provided by the `Progress` object of Shiny. Figure 31 shows an example progress bar. Considerations include:

- (s101) A progress object is created
- (s102) The progress object is given text to be presented and the proportion of completed progress to be indicated by the bar
- (s138) The progress object is closed and removed from the screen
- It can be a challenge to provide an accurate statement of proportion completed. For instance, in a call to `apply()`, the index vector is iterated over and, without knowledge of the vector length and proportional computation time required for each, the progress bar cannot be updated with an accurate proportion complete.
- Because many functions in R packages are executed asynchronously, the progress object may be closed before a function call completes. For instance, composing the list to configure a `ggplot` object is generally very efficient, but rendering the actual plot on a device (R graphics device or Shiny web page) may not be. In V5 a progress value is displayed (s255) and a call to `renderPlot()` is made (s256), which is executed asynchronously, so that the Shiny script continues with closing the progress object (s257) before the plot is visible in the user's browser. Often, the time between progress bar closure and appearance of the plot is more than ten seconds.

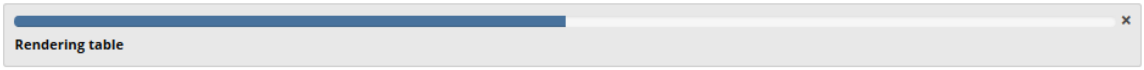


Figure 31: Screen-shot of example V5 progress indicator

9.5.5 Dynamic Creation of Panels with Imported HTML for Supplemental Project Info

Often, we want to provide information on data sources, explanations of analyses, links to published papers, and a primer on how to use our app. Since such information is generally static, we can compose web pages in external HTML files, import them into our app, and display content related to a particular request. Since the output of a Shiny app is, effectively, a stream of HTML, introducing a customized web page, including images, links, tables, and pdfs is fairly straightforward. Figures 33 and 34 are example information panels

that are generated as requested by the user (content was borrowed from the iCPAGdb app described in section 2.3. Following are the steps used by the V5 app to import an external HTML file and render its contents in Quick start and About dynamic panels:

- (u127, u129, figure 32) Action links (IDs `quickStartLink` and `aboutLink`) are defined for Quick start and About pages
- (s456-479, s499-522) Observe events are defined for `quickStartLink` and `aboutLink`
- (s459-477, s502-520) `insertUI()` dynamically appends a UI, composed from the design defined by its parameters, to the current UI
- (s461-462, s504-505) The UI insertion point (`selector`) is `supplementalText`, which is the last item in the UI (u259) and the `where` parameter is set to "afterEnd." This superimposes the dynamic panels above (on top of) all other content, so that they are guaranteed to be visible.
- (s463, s506) `immediate=TRUE` causes immediate rendering of panels
- (s465-476, s508-519) Draggable panels are defined as containers for the HTML. IDs (`quickStartText` and `aboutText`) are assigned so that the panels can be referred to in subsequent operations (removal).
- (s469, s512, figures 33 and 34) `includeHTML()` is used to import HTML from the QuickStart.html and About.html files in the SupplementalMaterial subdirectory of the current working directory. Following is an excerpt of the contents of QuickStart.html.

```
<h4>Quick-start guide</h4>

<p style="margin-top:20px">
<b>Review iCPAGdb:</b> Explore pre-calculated iCPAGdb results
<p>

<ol>
  <li style="margin-top:5px">
    Using the selection table (in section 1) click the row with the pair of GWAS datasets that you want to
    compare (combinations of NHGRI-EBI GWAS catalog, metabolomics, or cellular host-pathogen traits) at the
    specified p-value thresholds and with the specified population used in calculating LD.  Results appear
    in the "Table" and "Heatmap" sections below the query controls.
    <ul>
      <li style="margin-top:5px">
        Table: Each pairwise trait combination is listed (Trait<sub>1</sub>, Trait<sub>2</sub>) along with
        the number of SNPs that overlap directly (Nshare<sub>direct</sub>), the number of SNPs that overlap
        based on LD-proxy (Nshare<sub>LD</sub>), and the number of SNPs that overlap overall
        (SNPshare<sub>all</sub>), -log<sub>10</sub>(p-values) calculated by Fishers exact test
        (-log<sub>10</sub>(PFisher)) and corrected by either Benjamini-Hochsberg
        (-log<sub>10</sub>(Padjust<sub>FDR</sub>) or Bonferroni (-log<sub>10</sub>(P<sub>Bonferroni</sub>)),
        the shared SNPs (SNP<sub>shared</sub>), similarity indices (Jaccard and Chao-Sorenson), and ontology
        information based on EFO codes in NHGRI-EBI GWAS catalog. Results can be sorted by any column or
        restricted by text of any column.
      </li>
      <li style="margin-top:5px">
        Heatmap: By default, the heatmap is based on the Chao-Sorenson similarity index, but options for
        Jaccard similarity or -log<sub>10</sub>(P<sub>Fisher</sub>) are also available. Hover over cells
        to inspect the plotted value for pairs of phenotypes.  Click and drag to zoom in on a particular
        region of the heatmap.
      </li>
    </ul>
  </li>
```

- (s471, s517) Action buttons are included in the UI. With corresponding observe events, these become reactive and a mechanism for programmatically closing associated dynamic panels. Implementation of dynamically created reactive elements (buttons) is described in section **??**.

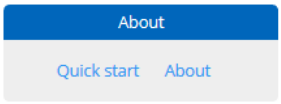


Figure 32: Screen-shot of V5 About panel

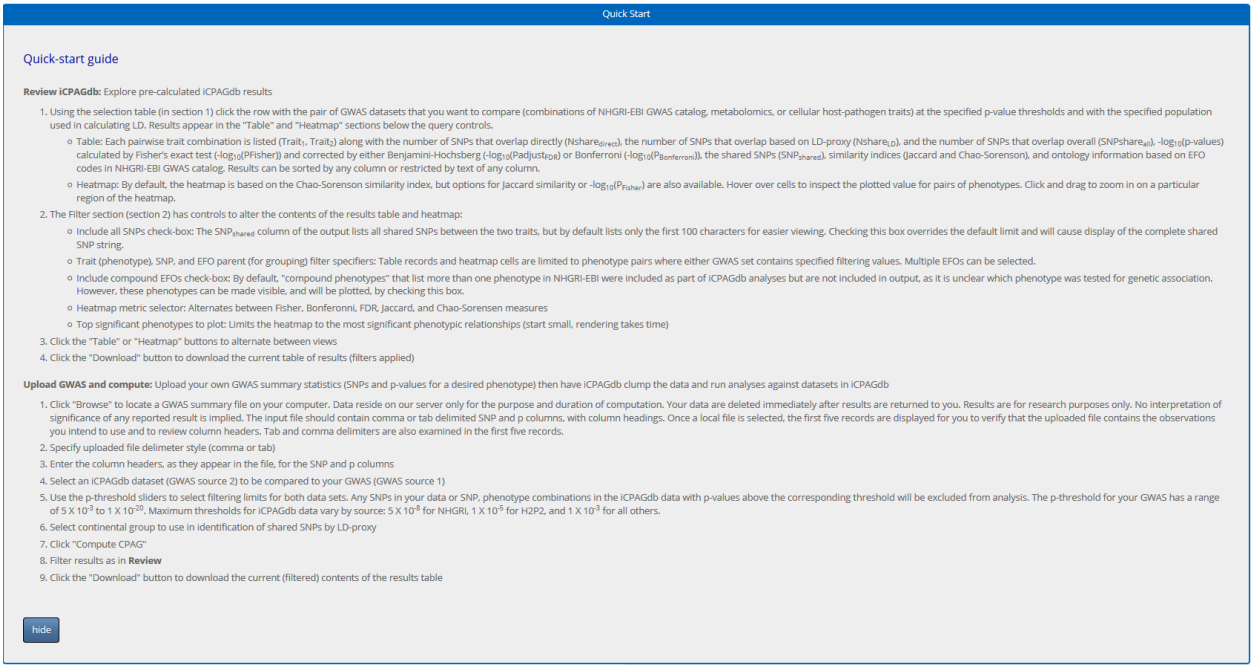


Figure 33: Screen-shot of V5 Quick Start dynamic panel

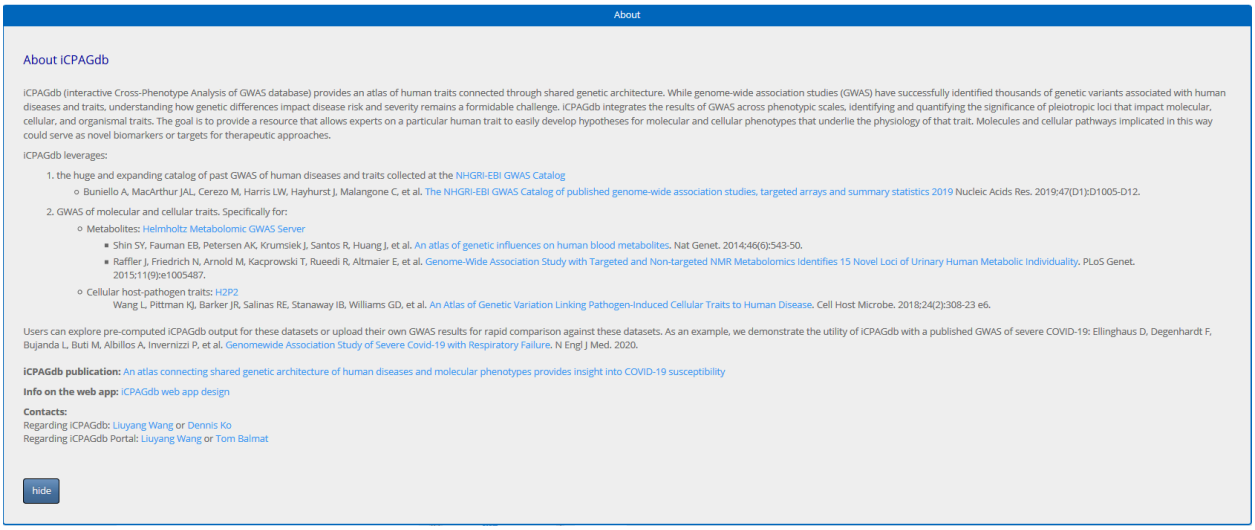


Figure 34: Screen-shot of V5 About dynamic panel

9.5.6 Dynamically Created Reactive Elements

Creation of dynamic panels was presented in section 9.5.5 and it is assumed that we also desire to dismiss such panels on user demand. Each panel UI contained an action button but, to be reactive (have some effect), an element must have an observe event defined for it. These events are no different from those created for statically defined elements (in `ui.r`). We simply create an observe function (in `server.r`) with name corresponding to the dynamic element to be created later in the call to `insertUI()`. Following are the steps used by the V5 app to create a reactive element for dismissing a dynamically created information window:

- (s471, s517) Action buttons are included in the UI. With corresponding observe events, these become reactive and a mechanism for executing further instructions
- (s485-493, s528-536) Observe events are defined for the dynamically created action buttons
- (s487, s530) The observe events each have a single instruction - `removeUI()`, which closes the corresponding draggable panel (note the requirement to prefix the panel ID with #)

9.5.7 Error Handling

Typically, a Shiny script will terminate if an error occurs during execution, causing the user's browser session to "gray" out. Error messages are displayed in the R console. To avoid this, program flow can be intercepted when an error occurs, a message can be given to the user, and instructions to remedy the error can be executed. A trycatch() block is one way to accomplish this. Considerations:

- Basically, with `tryCatch()`, execution of a block of instructions is attempted and, if an error or warning occurs, a user defined handler function is called. Following is an example `tryCatch()` block within an observe event handler.

```
observeEvent(input$var, {

  tryCatch(
  {
    # instructions
  },
  warning=function(err) msgWindow("WARNING", "Title", err),
  error=function(err) msgWindow("ERROR", "Title", err)
  )

}, ignoreInit=T)
```

- The warning and error functions call the `msgWindow()` function, which is defined in lines 66-69 of `server.r`. `msgWindow()` displays a modal window with the error text as the message (err passed as the msg parameter). A `modalButton` is displayed in the window footer, which dismisses the window. Once the modal window is dismissed, processing continues since the Shiny script is not terminated.
- (s77-163, s174-278, s365-377, s387-412, s422-448) Important `tryCatch()` blocks implemented in the V5 app
- (s163, s164) To test error handling, `stop(‘‘message")` can be inserted into the try section of a `tryCatch()` block

10 Advanced Data Table Features

Each of the CPDF apps presented have employed rather basic tables. The Data Tables package includes many options for improving user interaction with a table, including:

- A rows per page selector
- General and column-wise search

- Column formatting, justification, and ordering
- Editable cells
- A table caption
- Download capability

The file App/AdvancedDataTableFeatures/AdvancedDataTableFeatures.r contains an alternative `t1ComposeTable()` function that implements a number of advanced features (App was downloaded in section 7). To utilize this function, replace lines 75-165 of the V5 `server.r` with the following instruction:

```
source("App/AdvancedDataTableFeatures/AdvancedDataTableFeatures.r", local=T)
```

A discussion of implemented features will follow.

11 Plot Animation - Fiscal Year Slider Bar

Source directory: App/V9-CPDF-FYSliderBar (App was downloaded in section 7).

Discussion points:

- (u111-114) A `sliderInput()` is defined within a conditional panel, such that it is visible only when the plot independent variable is not fiscal year (why do we not want FY animated when it is the independent variable?)
- (u111) Animation executes an observe event once for each successive value of a vector (a range of integers from 1988-2011 here)
- (s262-311) The observe event defined for the `sliderInput()`. It is executed once for each year in the fiscal year range, rendering a succession of plots by year.

Experiment!

12 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