**Shiny Workshop, VT lab may 2019**

**Intro:**

* Shiny translate R to HTML, CSS, JavaScript to build interactive applications
* Built on functions and concept of reactivity
  + if you don’t use functions now, you will, changes coding style
  + Reactivity= Connection, What happens on user side, processed on server side, back to user, complete loop happens quickly, on user input or start of app
  + Every object/component of app must be unique
  + Troubleshooting key, frequently run/deploy apps to test
* Great for sharing data products with people who don’t know R/code
* How we use at DEQ:
  + TMDL meetings/pipeline review/interactive maps, present complex datasets to public/share data between disparate parts of our agency
  + Front end for tools- BCG apps, Benthic TMDL stressor analysis, water permit background metals, water quality assessments rivers& lakes, station creation tools
  + Where we are going: Database entry/querying tool, dashboards
* Questions before starting
* Demo TMDL app, basis for example modules

**1\_appLayout**

* look at basic structure of shiny app
* **STEP 1: MAKE NEW PROJECT FOR EVERY APP**
  + establishes working directory, file locations key for shiny app
  + each app must be in unique folder, file structure critical for shiny apps
* why separate ui and server
* what is global.R, data, www folders
* Run app, then try to switch up css (cascading style sheet) to others provided in www folder
  + also where images are stored
* drag size of window and see how shiny handles it; try run in window/viewer pane/external
* introduce headers, columns, span to change in text styling, br() vs hr()
* comma at end of every line, practice deleting one and look at error message
* Forced to save every time you run app, why github is important

**2\_basicApp\_widgets**

* open project in new RStudio session, see general architecture of shiny app, look at global, ui, server files, final app scripts
* run through app layout with app open, basic reactivity included at the start
* Note selectInput(‘dropdown’) choices are only Ecoregions, use levels(cdfdata$Subpopulation)
  + will fix that later
  + levels(cdfdata$Indicator) all radiobutton options are there but not matching data
  + Complete Step 1- fix radiobutton and subset data based on user input
* Step 2- add table output from cdfsubset()
  + explain why it needs to be called as a function- reactive element, if wasn’t reactive then the app would not update on user input
* Step 3- add a plot of cdf data with the option to turn conf int on/off
  + cdfplotFunction() already in global.R, run through what it is doing
* Step 4- add download button, go over naming the file different things
  + go over using run app external here for download to actually do something
* Step 5- add all dat$Subpopulation options to selectInput()
  + selectInput() good for a few things you want to define but setting up a renderUI() statement is the best if you have a long list of things you can call from a dataset
  + renderUI makes the user interface reactive
  + nest our selectInput() in renderUI() on the server side of things so it can process all the options we give it (based on input data)
  + show what happens if you don’t fix output$dropdown in cdfsubset()
  + demonstrate req() trick to avoid error messages

Widgets, reactive elements, renderUI statements, using programmer defined functions in an app, downloading data from app, troubleshooting basics

**3\_leafletMapApp**

* Go over preprocessData.R, always save this way when converting files from original spreadsheets. **Open up data and look at it!**
  + RDS for pointfiles, saved two types for this app, one for map and one for data viewing, maps work best with long format data
* Go over global.R, identify colors and breaks in data, bring in preprocessed RDS
* ui.R- css, navbarPage looks different than other apps, have to include bootstrap.css for leaflet maps to look good. bootstrap page essentially allows you to take up whole screen for map.
  + use tabPanel to organize app
  + map tabPanel explanation of each section
* server.R- output$myMap and renderLeaflet call
  + go to link with leaflet basemaps
  + explain API key issue
* Step 1- Hook up leaflet map to our data- zooms in to our data extent
* Step 2- Add an option to change the basemap in the absolute panel
  + change addProviderTiles() in output$myMap
* Step 3- Plot Probmon data on Map
  + ui- talk about columns, wellPanel, checkboxInput named checkbox 1 bc going to add a few of these and a button to remove all at once, named all the same we can remove all in a loop
  + server- talk about unique group names, I add \_ after object name just to keep it unique but not introduce too much to keep straight
    - matching up long/lat names
    - groups important for ELSE remove statement
    - popup call
* Step 4- Plot shapefiles on map
  + how to call in geospatial data, file locations, DEPLOYMENT ISSUES/LINUX
    - WGS84 projection for leaflet
  + placement of readOGR(), not in server call itself(would repeat) not in global either
  + plotting order depends on order checkboxes clicked, except points go on top
* Step 5- Make action button to remove all layers at once
  + note it is in observeEvent call, basically benefit is that you don’t have to handle the NULL option, prevents action buttons from bombing out
  + change n checkboxes to the total n checkboxes in app
* Step 6- Click point on map and get lat/long
  + talk through verbatimTextOutput for troubleshooting
  + observe() statement, waits for something in app to happen then executes
    - \_marker\_click great leaflet tip, can do brush as well and maybe hover
    - only works when point data is available to app, prob data must be turned on bc it is marker\_click
* Step 7- Tell server to subset original dataset based on click results
  + change the verbatimTextOutput to stationID to test if subset is working
* Step 8- Display clicked data
  + finally populate Table tabPanel
  + why output$table must be in observe statement (it only exists in observe statement, we did not save it as a separate object/function, we could but would take more steps
    - if(is.null()) prevents Table tab from throwing error message if you click on tab before selecting data
* Step 9- DT package extensions: buttons
  + MUST RUN EXTERNAL TO SEE ALL OPTIONS
  + modify datatable call to include button options
  + dom controls the elements around the table
  + B=Buttons, f= filter, r=processing, t =table, i=table information, p= pagination
  + dom= ‘Bt’ buttons and table
  + formatStyle based on definitions in global.R
* Step 10 - Show difference between tableOutput and DT::dataTableOutput
  + adds tabPanel inside tabPanel

All markers are a lot, change to limit markers so users can see spatial distributions in data

* Step 11- Change Markers plotted and color based on selectInput
  + 11a) add selectInput, specify what user sees bc acronyms in dataset
  + 11a) colOptions and pal defined to correspond to factor levels in data
  + 11b) Connect changes on server side
    - filteredData is reactive function that filters whole dataset based on user
    - don’t have to delete checkbox1 on server/ui but could at this point
    - addCircleMarkers in observe() statement bc we want it to update as we change parameters

Practice deploying app