Project: Visualization of California Water

June 10, 2015

Bit Bucket Repository: https://z109620@bitbucket.org/z109620/project.git

R code to access our Visualization: require(shiny); runGitHub(repo = "stat242project", username = "txzhou", subdir = "shinyApp")

Jacob Humber, Michael Levy and Tianxia Zhou

1 Introduction

Since the onset of 2012, California has been experiencing one of its worst droughts in over a thousand years[1]. For the year of 2014 alone the agricultural losses as a results of the drought totaled 2.2 billion dollars as well as a loss of 17,000 agricultural jobs [2]. In an effort to convey the magnitude of this persistent drought the United States Geological Survey (USGS) generated an interesting visualization[3]. While this visualization helps to illuminate the magnitude of the California drought, it focuses almost exclusively on surface water reservoir levels. Surface water reservoir levels only tell part of the story of the California drought. Clearly, an thorough understanding of California water consumption, stream flows, the depth of ground water wells are all important. The current project, therefore, will generate visualizations[4] which will depict each of these aforementioned water metrics and is therefore meant to compliment the existing USGS visualization.

All visualizations within the project are generated with the R package shiny. For a proficient R user utilizing the shiny package is relatively straightforward. At a minimum a shiny application consists of a user-interface

[1]http://www.nytimes.com/2015/05/03/opinion/sunday/the-end-of-california.html?_r=0

[2]http://www.wsj.com/articles/drought-will-cost-california-2-2-billion-in-losses-costs-this-year-1405452120

[3]The USGS visualization: http://cida.usgs.gov/ca_drought/

[4]Our visualization: To access the visualization the following lines of code must be executed: require(shiny); runGitHub(repo = "stat242project", username = "txzhou", subdir = "shinyApp"). Alternatively, the shiny app can be run by cloning the repository locally on your own machine.

(ui.R) script for the front-end and a server script (server.R) for the back-end. Each of these scripts are written in R. Consequently, the shiny packages provides some of the elegance of JavaScript without needing to know HTML, CSS or even JavaScript itself!

Three distinct visualization are generated within the project. The first depicts California water consumption by county and sector. The next depicts discharge rates of streams and rivers. Finally, the last visualization depicts the ground water depth of wells. In what follows, we will discuss each of these visualization separately within sections 2, 3 and 4. All R scripts are contained in the Appendix below.

2 Water Consumption

As water becomes scare, it is essential to understand both the industries as well as the areas within California that consume the most water. These sectors and areas are those most likely to bare the brunt of the hardship that is the California drought.

The USGS visualization does provides a pie chart of water consumption by industry. However, in the current project we expand this by depicting the water consumption by county by year for 2000, 2005 and 2010[5]. Unfortunately, the 2015 data is not currently available. Additionally upon a click of a county, a graph appears which displays the water consumption by sector for that county.

Since the data set for water consumption is small, we store all the data remotely on our Git Hub page. Intentionally we wanted to host our shiny application on shinyapps.io, however, the only way to host data on this website is to pay $30 a month, which is a lot of money for a group of graduate students.

The code to generate this visualization is contained the in R scripts plot.R and functions.R in the Appendix. The color.map function within functions.R generates the plot of California. This is essentially a wrapper of the map function from R's maps package. Upon clicking on a county, a graph of the consumption by sector for that county is generated with the ggplot wrapper gg.wrapper. Notice that when a users clicks the map, longitude and latitude coordinates are passed to server.R, however we don't want to pass longitude and latitude coordinates to the gg.wrapper, we want to pass a county name. Consequently, we must convert these longitude and latitude coordinates to a county. This is done with the latlong2county function within functions.R[6].

[5]Data source: http://water.usgs.gov/watuse/data/

[6]The latlong2county function draws heavily from the following StackOverflow thread: http://stackoverflow.com/questions/13316185

3 Stream flow

In the current context stream flow is measured by the discharge rate. The discharge rate is the volume of water moving down a stream or river per unit of time, in our context the discharge rate is measured in cubic feet per second.

The USGS visualization implicitly depicts stream flows insofar as the different drought categories within their visualization - No Drought, Abnormally Dri, Moderate Drought, Severe Drough, Extreme Drought and Exceptional Drought - are defined by varying stream flows. However, it is not possible to obtain, from their visualization, the actual stream flow from participial streams. Consequently, our visualization depicts the discharge rate read from each surface gauge monitored by the USGS

In this context we have utilized the leafet function in the R package of the same name to draw our map of California, see the R script server.R. Like D3, leafet is a JavaScript library. The leafet package, which is similar to shiny in this regard, introduces some of the functionally of JavaScript without the need to know how to write JavaScript code. The main reason for invoking leafet here is that this function makes it straightforward to create a map which the user can zoom in and out on. Given the numerous observation, to insure clarity, it was necessary that user have the ability to zoom. Another interesting feature of this visualization is that unlike the visualization of water consumption, the data on discharge rates are not stored remotely on Git Hub. This data is expansive and consequently we opted to load data upon a click. This is made possible by the R package dataRetrieval. This packages allows users to query the expansive hydrological data provided by USGS. Consequently, when a user clicks a site, the corresponding site number is passed to dataRetrieval. The data is in turn passed to plot wrapper functions in USGSplot.R.

4 Ground Water

Ground water wells are not currently depicted within the USGS visualization. Despite this fact, ground water is essential to California's water supply as it comprises 30% of the California water supply. Consequently, we provided a visualization of water well depth for all wells currently monitored by the USGS.

The approach used to generate the ground water visualization is very similar to the stream flow visualization. As above both, the leafet and dataRetrieval packages are utilized.

5 Conclusion

This project compliments the USGS visualization. It does so by visualizing water consumption, stream flows and ground water elevation. In this project we opt to use the R shiny package. Clearly, this assignment could have been done using JavaScript, however, shiny is capable to generate some very nice looking graphics and only requires a knowledge of R. That withstanding, a fun summer project would be do redo some of this project in JavaScript to understand the differences between these two approaches.

Appendix

ui.R

```
library(shiny)
library(maps)
library(leaflet)
library(mapdata)
library(maptools)
library(Hmisc)
library(ggplot2)
library(reshape2)
library(dataRetrieval)
library(data.table)
library(RColorBrewer)

# Define UI for application
shinyUI(
  navbarPage(
    "California Water Use",

    tabPanel(
      "Use by county",
      h1("Water use"),

      fluidRow(
        column(5,
               #Selection box
               selectInput("metric",
                           label = "Choose a Water Consumption Metric",
                           choices = list("Percent of California Consumption",
                                          "Per Capita Consumption")),
               selectInput("year",
                           label = "Choose a Year",
                           choices = list(2000,2005,2010)),

               helpText("Click on a county for consumption by
sector for that county in the selected year."),

               # Print the clickable map
               plotOutput("theMap", height = "400px", click = "plotclick"),

               textOutput("badCounty")  # This line has to be present for the
               # conditionalPanel()s to work. I have no idea why. Maybe badCounty has
               # to be evaluated before the JS test is called???



        ),

        column(6,
               helpText("California is a geographically diverse state with widely varying
                economies and population densities across its counties. Here you
                can explore how much water each county uses and what they use it for."),

               # offset = 1,
               # Show a plot of the generated distribution
               conditionalPanel("output.badCounty == 0",
                                plotOutput("useagePlot"))
        )
      )
    ),

    tabPanel(
      "Surface water flow",
    h1("Surface water"),

    fluidRow(
      column(5, leafletOutput("siteMap")),
      column(6,
             helpText("Surface water provides some of California's water supply. It is
             stored in massive manmade reservoirs on almost all the major rivers and
             in the natural reservoir of high alpine snowpack in the Sierra
             Nevada mountains. In the current drought, snowpack has been far
             less than normal, which means less runoff to fill the reservoirs,
             the levels of which have dropped dramatically."),
             br(),
             helpText("The map to left shows US Geological Survey surface water monitoring stations.
             Click on one to see how runoff changes seasonally and annually over
             the course of the drought. Try zooming in and looking at differences
             above and below major dams.")
      )
    ),

    #        column(6,
    #               offset = 1,
    conditionalPanel("!is.na(output.mapClick)",
                     plotOutput("sitePlot"))
    #        )
    #      )
  ),

  tabPanel(
    "Groundwater levels",
    h1("Groundwater"),

    fluidRow(
        column(5, leafletOutput("gwMap")),
        column(6,
               helpText("If surface water is California's first stop for water needs,
               groundwater is its backstop. Only a few municipalities still
               rely on groundwater (Davis being one of them); however, it is of critical
               importance for irrigation. When surface supplies are down,
               reliance on groundwater increases, and unlike surface water and
               unlike every other state in the US, groundwater use is unregulated
               in California. This has led to a furious groundwater grab during
               the current drought."),
               br(),
               helpText("The map to the left shows the groundwater wells that the the US
               Geological Survey uses to monitor groundwater elevation. Unfortunately,
               even public monitoring of groundwater levels is legally challenging
               in California, so data are sparser than hydrologists and water
               managers would like. However, here you can click on a series of wells
               to see how groundwater levels have changed over the course of the
               current drought.")
        )
      ),

      #         column(6,
      #                offset = 1,
      plotOutput("GWPlot")
      #         )

      #       textOutput("wellsInfo"),

      #       )
    )
  )
)
```

server.R

```r
#path.app <- "C:/Users/Athena/Desktop/project/shinyApp/"
#path.toapp <- "C:/Users/Athena/Desktop/project"

#setwd(path.app)

# source files ####
source(file = "functions.R")
source(file = "plot.R")  # Modified this so it brings df.long into the workspace
source(file = "readUSGSData.R")
source(file = "USGSplot.R")

shinyServer(function(input, output) {

  theCounty = reactive({
    if(is.null(input$plotclick))
      return("none")
    latlong2county(
      data.frame(x = input$plotclick$x, y = input$plotclick$y))
  })


  # Test whether theCounty() is valid, for startup and bad clicks, to not
  # display charts on ui side. Idea from https://gist.github.com/ptoche/8312791
  output$badCounty <- renderText({
    if(
      theCounty() == "none" |
        is.na(theCounty())   |
        !any( grepl(theCounty(), counties) )
    ) {
      return(1)
    } else {
```

```
    return(0)
  }
})

output$theMap <- renderPlot({
  # color.variable == 1L or 2L (can include more variables)
  # see functions.R for detail.
  select.box <- switch(input$metric,
                       "Percent of California Consumption" = 1L,
                       "Per Capita Consumption" = 2L)

  color.map(color.variable = select.box, year.map = as.numeric(input$year))
})

output$countyText <- renderPrint({
  cat("That's ", simpleCap(theCounty()), " county.")

})

output$useagePlot =
  renderPlot(
    gg.wrapper(county.name = theCounty(), year.gg = as.numeric(input$year))
  )

output$siteMap <- renderLeaflet({
  leaflet(data = goodSurfaceData) %>%
    addProviderTiles("Esri.WorldTopoMap") %>%
    addCircleMarkers(~long, ~lat, layerId = ~ siteNumber, radius = 2)
})

observe({
  if(!is.null(input$siteMap_marker_click$id))
```

```
    output$sitePlot = renderPlot({
      plot.discharge(siteNumber = input$siteMap_marker_click$id)
    })
})

output$gwMap <- renderLeaflet({
  leaflet(data = gwSites) %>%
    addProviderTiles("Esri.WorldTopoMap") %>%
    addCircleMarkers(~long, ~lat, layerId = ~ siteNumber,
                     color = "red", radius = 2)
})

observe({
  if(!is.null(input$gwMap_marker_click$id))
    output$GWPlot = renderPlot({
      gwPlot(input$gwMap_marker_click$id)
    })
})

if(FALSE){
  theGWSites = reactiveValues()
  theGWSites$sites = 374004122092106 # character(0)

  # On gw-map click, if well isn't in the vector to be plotted, add it.
  observe({
    if(!is.null(input$gwMap_marker_click$id) &
         !input$gwMap_marker_click$id %in% theGWSites$sites) {
      nextWell <- isolate(input$gwMap_marker_click$id)
      if(!nextWell %in% theGWSites$sites)
        isolate(theGWSites$sites <- c(theGWSites$sites, nextWell))
    }
  })
```

```
    observe({
      if(input$clear > 0) {
        theGWSites$sites = 374004122092106  # ""
      }
    })

    output$wellsInfo =
      renderPrint(cat("Plotting wells:", theGWSites$sites, sep = "\n"))

    output$GWPlot = renderPlot({
      gwPlot(theGWSites$sites)
    })

  }

})
```

functions.R

```
# install packages and load packages ####
packages.list = c("shiny",
                  "maps",
                  "mapdata",
                  "maptools",
                  "Hmisc",
                  "ggplot2",
                  "reshape2",
                  "dataRetrieval",
                  "data.table",
                  "RColorBrewer")

for (p in packages.list) {
```

```r
  if (!(p %in% rownames(installed.packages())))
    install.packages(pkgs = p)
}

if (!("leaflet" %in% rownames(installed.packages()))) {
  require("devtools")
  devtools::install_github("rstudio/leaflet")
}

library(shiny)
library(maps)
library(leaflet)
library(mapdata)
library(maptools)
library(Hmisc)
library(ggplot2)
library(reshape2)
library(dataRetrieval)
library(data.table)

latlong2county <- function(pointsDF, wantState = FALSE) {
  # Taken verbetim from http://stackoverflow.com/questions/13316185
  # Prepare SpatialPolygons object with one SpatialPolygon
  # per county
  counties <- map('county', fill=TRUE, col="transparent", plot=FALSE)
  IDs <- sapply(strsplit(counties$names, ":"), function(x) x[1])
  counties_sp <- map2SpatialPolygons(counties, IDs=IDs,
                                     proj4string=CRS("+proj=longlat +datum=wgs84"))

  # Convert pointsDF to a SpatialPoints object
  pointsSP <- SpatialPoints(pointsDF,
                            proj4string=CRS("+proj=longlat +datum=wgs84"))
  # Use 'over' to get _indices_ of the Polygons object containing each point
  indices <- over(pointsSP, counties_sp)

  # Return the county names of the Polygons object containing each point
  countyNames <- sapply(counties_sp@polygons, function(x) x@ID)

  county = strsplit(countyNames[indices], ",")[[1]]

  if(wantState) { county
  } else {
    county[2]
  }
}

color.map = function(color.variable = 1L, year.map) {
  # adapted from the example in the help doc of "map()" function.

  # manipulate the water dataset so that we can match it to the map data.
  df.water.cal = water.consum.data(long = FALSE, year= year.map)
  if(year.map == 2010){
  df.water.cal$cal.County = sapply(X = strsplit(x = levels(df.water.cal$County), split = ' County'),
                                   FUN = function(x) x[[1]])
  } else {
    df.water.cal$cal.County=df.water.cal$County
  }
  df.water.cal$polyname = paste0("california,", tolower(df.water.cal$cal.County))

  # define color buckets
  colors = c("#fee5d9",
             "#fcbba1",
             "#fc9272",
             "#fb6a4a",
              "#de2d26",
              "#a50f15")

if (color.variable == 1L) {
  df.water.cal$colorBuckets <- as.numeric(cut(df.water.cal$Percent, breaks = c(0, 0.01, 1:5*0.02)))
  leg.txt <- c("<1%", "1-2%", "2-4%", "4-6%", "6-8%", "8-10%")
  title.txt <- "County consumption:\n% of California total"
} else if (color.variable == 2L) {
  df.water.cal$colorBuckets <- as.numeric(cut(df.water.cal$Per.Cap, breaks = c(0, 1, 5, 10, 20, 40)))
  leg.txt <- c("<1", "1-5", "5-10", "10-20", ">20")
  title.txt <- "Per Capita Consumption \n (Mgal/day/1000 people)"
}



# align data with map definitions by (partial) matching state,county
# names, which include multiple polygons for some counties
colorsmatched <- df.water.cal$colorBuckets[
      match(map(database = "county", regions = "california", plot=FALSE)$names,df.water.cal$polyname)

# draw map
map(database = "county", regions = "california", col = colors[colorsmatched], fill = TRUE,
      resolution = 0, lty = 1)
# the following lines might be useful if we draw the map for the whole U.S.
#   map("state", col = "white", fill = FALSE, add = TRUE, lty = 1, lwd = 0.2,
#       projection="polyconic")

#INCLUDING CITIES
#data(us.cities)
#cities <- c("^San Francisco", "^West Sacramento", "^Los Angeles", "^San Diego", "^Fresno")
#city.index <- sapply(cities , function(x){grep(x, us.cities[ ,1])})
#map.cities(us.cities[city.index, ], country = "CA",
      #label = TRUE, pch = 16, col = "black", cex = 1.5, font = 2)
  title(title.txt)
  legend("topright", leg.txt, fill = colors)
}

simpleCap <- function(x) {
  # From toupper() help file.
  s <- strsplit(x, " ")[[1]]
  paste(toupper(substring(s, 1,1)), substring(s, 2),
        sep="", collapse=" ")
}
```

plot.R

```
#This Script will make the ggplot wrapper
# rm(list=ls())  # ML: It seems very dangerous to me to source files with this command.

# We want to build this data.frame once, not on each call to gg.wrapper.
water.consum.data <- function (select.variables = NULL,
                               long,


                               year=2010) {
  df <- read.csv(file = paste0("./clean_data/ca_",year,".csv"))



  if (!is.null(select.variables))
    df <- df[ , select.variables]

  #To use ggplot, I need to reshape the data from wide to long.
  if (long) {
    df.long <- melt(df, id = "County")
    names(df.long)[2:3] <- c("Source","Water")
    return(df.long)
  } else {
```

```r
    return (df)
  }
}

cc = rep("NULL", 20)
cc[3] = "character"
counties = tolower(read.csv("./clean_data//ca_2010.csv", colClasses = cc)$County)

# debugonce(gg.wrapper)
# gg.wrapper("imperial", "2000")
#plot
gg.wrapper <- function(county.name, year.gg){

  theDF <- water.consum.data(
    select.variables = c("County", "Public.Supply", "Domestic.Self", "Industry",
                         "Irrigation", "Livestock",
                         "Aquaculture", "Mining","Thermoelectric"),
    long = TRUE,
    year = year.gg)

  plotDF = theDF[grep(pattern = county.name, x = tolower(theDF$County)),]
  plotDF$Water = plotDF$Water * .325851 * 365.25 # MGal/day -> AF/year
#  plotDF = plotDF[plotDF$Water > 0, ]
  plotDF$lab = sprintf("%1.0f", plotDF$Water)
  plotDF$Source = factor(plotDF$Source,
                         levels = plotDF$Source[order(plotDF$Water)],
                         ordered = TRUE)
  #First I will subsample the data.  Some data is a double count.
  #For example Ir=Ir.C+Ir.G (i.e. Irrigation = Irrigation Crops + Irrigation Golf)
  col = brewer.pal(8, "Set1")
  names(col) = c("Public.Supply", "Domestic.Self", "Industry",
                 "Irrigation", "Livestock",
                 "Aquaculture", "Mining","Thermoelectric")


  plot.water <- ggplot(data = plotDF, aes(x=Source,y=Water, fill =Source ))+
    geom_bar(stat="identity")+
    theme_bw()+
#    scale_y_continuous("Total Fresh Water Withdrawn (Mgal/day)")+
    scale_x_discrete("") +
    scale_y_log10("Fresh Water Use (acre-feet/year)") +
#                 limits = c(1, max(theDF$Water) + .05 * max(theDF$Water))) +
  scale_fill_manual(values = col) +
  geom_text(aes(label = lab, y = ifelse(Water < 2, 1, Water / 2))) +
    coord_flip()+
    guides(fill=FALSE)+
    ggtitle(paste(simpleCap(county.name), "County"))+
    theme(plot.title = element_text(size=18, face="bold"),
                #Don't adjust text size. If you increase it will cut off San Luis Obispo County
          axis.text.y = element_text(size = 15),
          axis.text.x = element_text(size = 12),
          axis.title.x = element_text(size = 15))
  return(plot.water)
}
```

USGSplot.R

```
  plot = try(plot.discharge(siteNumber = siteNumber))


  require(ggplot2)
  null.plot = qplot(x = 0, y = 0)


  if ((class(plot)[[1]] != "try-error")) {
    return(list(0, plot))
  } else {
    return(list(1, null.plot))
```

```r
  }
}

plot.discharge.2 = function(siteNumber,
                            startDate = NULL,
                            endDate = NULL) {
  require(dataRetrieval)

  surfaceData = try(readNWISmeas(siteNumbers = siteNumber))

  stopifnot("measurement_dt" %in% names(surfaceData))
  stopifnot("discharge_va" %in% names(surfaceData))
  stopifnot("gage_height_va" %in% names(surfaceData))

  siteInfo = attr(surfaceData, "siteInfo")

  if (!is.null(startDate))
    surfaceData = surfaceData[surfaceData$measurement_dt >= as.POSIXct(startDate), ]

  if (!is.null(endDate))
    surfaceData = surfaceData[surfaceData$measurement_dt <= as.POSIXct(endDate), ]

  require(ggplot2)
  plot = ggplot(data = surfaceData, aes(x = measurement_dt, y = discharge_va)) +
    geom_point() +
    geom_line() +
    xlab("Date") +
    ylab("Discharge, cubic feet per second") +
    ggtitle(siteInfo$station_nm)

  return(plot)
}
```

```r
plot.discharge = function(siteNumber,
                          parameterCd = "00060") {
  require(dataRetrieval)

  data = readNWISdv(siteNumber, startDate = "2011-10-01",
                    parameterCd)

  data = renameNWISColumns(data)

  if(!"Flow" %in% names(data))
    stop("Sorry, USGS doesn't have flow data for that station.")



  variableInfo = attr(data, "variableInfo")
  siteInfo = attr(data, "siteInfo")

  require(ggplot2)
 ggplot(data = data, aes(x = Date, y = Flow)) +
    geom_line() +
    xlab("Date") +
    ylab(variableInfo$parameter_desc) +
    theme_bw() +
    scale_y_log10() +
    ggtitle(paste0("Daily warter discharge at: ", siteInfo$station_nm))

#  return(plot)
}

gwPlot = function(siteNum)
{
  ggplot(gwLevels[gwLevels$siteNumber == siteNum, ],
         aes(x = date,
#             y = log10(level))) +
             y = level)) +
             # color = siteNumber)) +
    scale_y_reverse() +
    scale_x_date(limits = c(as.Date("2011-10-01"), Sys.Date())) +
    geom_line() +
    geom_point() +
#    ylab("Depth, log10(feet)") +
    ylab("Depth (feet)") +
    xlab("Date") +
    theme_bw()
}
```

readUSGSData.R

```
sites = readRDS("clean_data/allSites.RDS")
goodSurfaceData = readRDS("clean_data/goodSurfaceSites.RDS")
gwLevels = readRDS("clean_data/gwLevels.RDS")
gwSites = sites[sites$siteNumber %in% gwLevels$siteNumber, ]
```