HW 4

Joe Brew

For full code see https://github.com/joebrew/uf/tree/master/phc6194/hw4.

Part 1a

```
#################
## Setwd
if ( Sys.info()["sysname"] == "Linux" ){
  setwd("/home/joebrew/")
} else {
  setwd("C:/Users/BrewJR/")
}
```

Read txt file (converted from dbf file) for Orlando

```
orlando <- read.csv("~/Documents/uf/phc6194/hw4/ORLANDO.txt")
```

Geocode the address table with an address locator by writing a function geocode addresses using Google's API.

```
#### This script uses RCurl and RJSONIO to download data from Google's API:
#### Latitude, longitude, location type (see explanation at the end), formatted address
#### Notice ther is a limit of 2,500 calls per day

##### Note that the source of this code snippet is: http://stackoverflow.com/questions/3257441/geocodin

library(RCurl)
library(RJSONIO)
library(plyr)

url <- function(address, return.call = "json", sensor = "false") {
  root <- "http://maps.google.com/maps/api/geocode/"
  u <- paste(root, return.call, "?address=", address, "&sensor=", sensor, sep = "")
  return(URLencode(u))
}

geoCode <- function(address,verbose=FALSE) {
  if(verbose) cat(address,"\n")
  u <- url(address)
  doc <- getURL(u)
  x <- fromJSON(doc,simplify = FALSE)
  if(x$status=="OK") {
    lat <- x$results[[1]]$geometry$location$lat
    lng <- x$results[[1]]$geometry$location$lng
    location_type <- x$results[[1]]$geometry$location_type
    formatted_address <- x$results[[1]]$formatted_address
    return(c(lat, lng, location_type, formatted_address))
  } else {
```

```
    return(c(NA,NA,NA, NA))
  }
}
```

Perform the geocode.

```
# Use plyr to getgeocoding for a vector
address <- paste(orlando$FAC_ADDR,
                 orlando$FAC_CITY,
                 "Florida",
                 orlando$FAC_ZIP)
locations <- ldply(address, function(x) geoCode(x))
names(locations) <- c("lat","lon","location_type", "forAddress")

orlando <- cbind(orlando, locations)
```

Save the .rdata file of the geocoded addresses (so as to not have to repeat).

```
save.image("~/Documents/uf/phc6194/hw4/hw4.RData")
```

On subsequent runs, simply reload (rather than call the API again).

```
load("~/Documents/uf/phc6194/hw4/hw4.RData")
```

Clean up the dataframe a little bit.

```
# save lat and lon as numeric objects
library(dplyr)
orlando <-
  orlando %>%
  mutate(lat = as.numeric(lat),
         lon = as.numeric(lon))
```

Save a shapefile version or orlando.

```
library(sp)
library(rgdal)
```

Keep only the non-NA's.

```
orlando_sp <- orlando[which(is.finite(orlando$lat) &
                            is.finite(orlando$lon)),]
```

Convert to spatial points data frame projected in latitude and longitude.

```
orlando_sp <- SpatialPointsDataFrame(orlando_sp[,c("lon", "lat")], orlando_sp,
                                     proj4string = CRS("+init=epsg:4326"))
```

Write the shapefile.

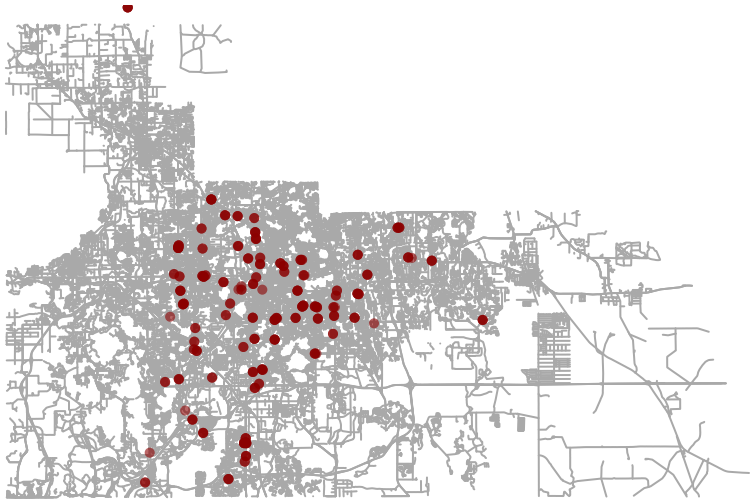
```
writeOGR(orlando_sp,
         dsn = "~/Documents/uf/phc6194/hw4/orlando",
         layer = "orlando",
         driver = "ESRI Shapefile")
```

Read in the Orange county shapefiles from http://www.census.gov/cgi-bin/geo/shapefiles2010/main.

```
# Read in orlando all lines (from http://www.census.gov/cgi-bin/geo/shapefiles2010/main)
#orlando_all <- readOGR("./Documents/uf/phc6194/hw4", "tl_2010_12095_edges")

# Read in orlando roads only
orlando_roads <- readOGR('/home/joebrew/Documents/uf/phc6194/hw4', layer = 'tl_2010_12095_roads', verbos
```

For the "screenshot", see the below plot:

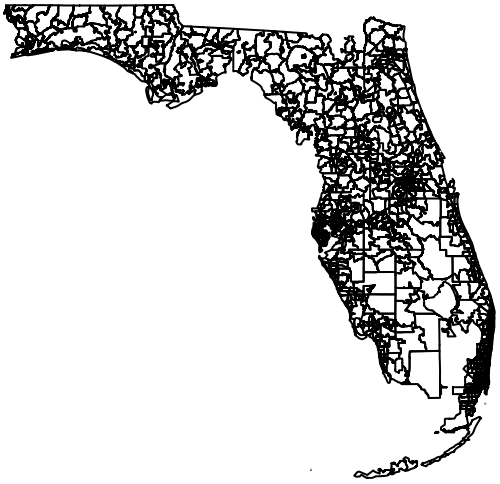
```
plot(orlando_roads,
     col = "darkgrey")
points(orlando_sp,
       col = adjustcolor("darkred", alpha.f=0.6),
       pch = 16,
       cex = 0.7)
#box("plot")
title(main = "Orlando dry-cleaning facilities")
```



Part 1b

1. Download the 2000 5-digit postal code shapefile for Florida from http://www.census.gov/geo/www/cob/z52000.html#shp (and read it into R).

```
zip <-  readOGR('/home/joebrew/Documents/uf/phc6194/hw4', layer = 'tl_2010_12_zcta510', verbose = FALSE
plot(zip)
```



2. Create a composite address locator using TIGER files from 1a and 5-digit postal code shapefile as reference datasets.

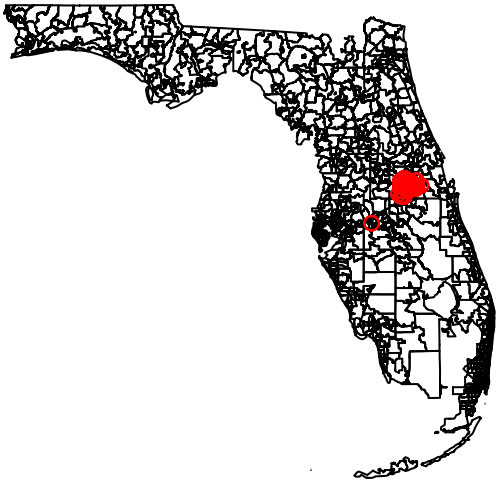
```
# This step is unecessary, given that I'll simply merging, binding and otherwise manipulating the raw d
```

3. Repeat the same steps in 1a and geocoding "Orlando.dbf".

```
# For the R method of this, I first create a zip column in both orlando and zip
orlando <- orlando %>%
  mutate(zip = FAC_ZIP)
zip$zip <- as.numeric(as.character(zip$ZCTA5CE10))

# In the zip polygons, I convert from factor to numeric
zip$INTPTLAT10 <- as.numeric(as.character(zip$INTPTLAT10))
zip$INTPTLON10 <- as.numeric(as.character(zip$INTPTLON10))

# Create a df version of zip
zip_df <- data.frame(zip)

# Finally, we bind the zip codes from orlando to their appropriate centroid points in zip
x <- merge(orlando, zip_df,
           by = "zip",
           all.x = TRUE,
           all.y = FALSE)

# The resulting dataframe contains the columns "INTPTLAT10" and "INTPTLON10", which are the latitutde a
```

4. Make a screen shot to copy the map from ArcGIS and paste it in a word document for grading

```
plot(zip)
points(x$INTPTLON10, x$INTPTLAT10, col = "red")
```



Question 2. Add x,y table and create a new shapefile.

The dataset of "FL_wel.xls" provides the X,Y coordinate system of the private wells in FL. The X,Y coordinate system is as the format of degrees, minutes, and seconds in a field with a delimiter of "-".

1. Please convert he vakyes to decimal degrees and store them in a numeric field

```
# First, read in the FL_wel.xls file
FL_well <- read.csv("~/Documents/uf/phc6194/hw4/FL_well.csv")

# Write a function to convert from degrees to decimals
DegreesToDecimals <- function(x){
  # split the string at the dash marks
  z <- do.call(rbind, strsplit(as.character(x), "-"))
  # perform the calculations to get to decimals
  zz <- as.numeric(z[,1]) +
    (as.numeric(z[,2]) + as.numeric(z[,3])/60 )/60
  # return the decimal converted object
  return(zz)
}

# Apply that function to latitude and longitude in order to get decimal coordinates
FL_well$lon <- DegreesToDecimals(FL_well$longitude)
FL_well$lat <- DegreesToDecimals(FL_well$latitude)

# Make longitude negative
FL_well$lon <- FL_well$lon * -1
```

2. Create the layer (i.e. shapefile) of "FL_PRIVATE_WELL" with a geographic coordinate system of "WGS 84" with the cleaned X,Y table using one of the following methods which we discussed in the class;

Using the Tools menu and click Add XY Data (i.e Method 1 in the handout); Using Method 2 in the handout (i.e "display X,Y data"); Or Using ArcCatalog to create a feature class from X, Y table.

```r
# Create a spatial object from FL_well
well_sp <- FL_well[which(is.finite(FL_well$lat) &
                           is.finite(FL_well$lon)),] # none removed


well_sp <- SpatialPointsDataFrame(well_sp[,c("lon", "lat")], well_sp,
                                  proj4string = CRS("+init=epsg:4326"))

# Convert projection string of well_sp to that of zip
proj4string(well_sp) <- proj4string(zip)
```

```
## Warning: A new CRS was assigned to an object with an existing CRS:
## +init=epsg:4326 +proj=longlat +datum=WGS84 +no_defs +ellps=WGS84 +towgs84=0,0,0
## without reprojecting.
## For reprojection, use function spTransform in package rgdal
```

```r
# Based on those points, interpolate which zip code they are in
my_points <- over(well_sp, polygons(zip))

# Get the zip code using my_points as the index
zip$id <- 1:nrow(zip)
well_sp$id <- my_points

zip_df <- data.frame(zip)
well_sp_df <- data.frame(well_sp)

well_sp_df <- merge(well_sp_df,
                    zip_df,
                    by = "id",
                    all.x = TRUE,
                    all.y = FALSE)

well_sp$zip <- well_sp_df$zip
```

Just for interest's sake, let's visualize the number of wells per zip code.

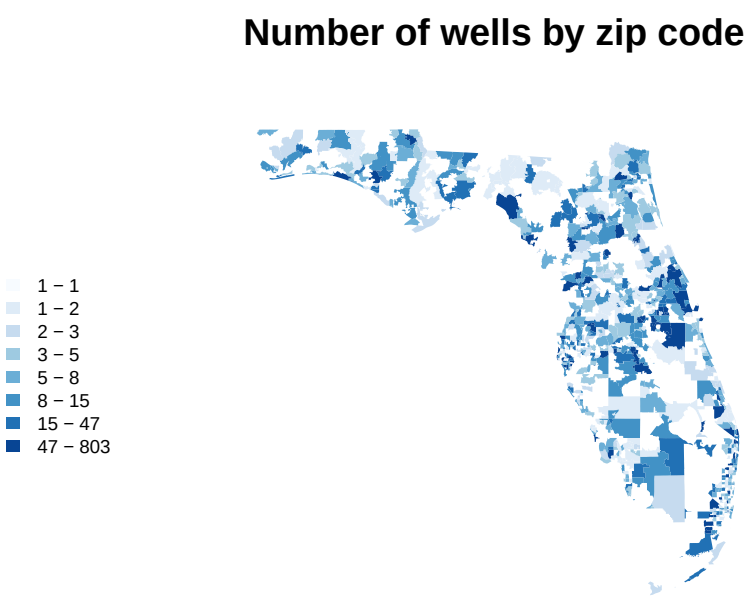
```r
# First, we need to calculate the number of wells in each zip code (we'll use dplyr)
well_sp_df$zip <- well_sp_df$ZCTA5CE10
zip_df$zip <- zip_df$ZCTA5CE10
well_by_zip <-
  well_sp_df %>%
  group_by(zip) %>%
  summarise(n = n()) %>%
  arrange(desc(n))

x <- merge(zip_df,
           well_by_zip,
           by = "zip",
           all.x = TRUE, all.y = FALSE)

# Now let's throw those numbers into our spatial object
zip$n <- x$n
```

Having gotten the numbers, let's plot a choropleth map.

```
# First, define a plotting function
library(RColorBrewer)
library(classInt)
WellFun <- function(var, color){
  plotvar <- var
  nclr <- 8
  plotclr <- brewer.pal(nclr, color)
  class <- classIntervals(plotvar, nclr, style = "quantile", dataPrecision=0) #use "quantile" instead
  #class <- classIntervals(0:100, nclr, style="equal")
  colcode <- findColours(class, plotclr)
  legcode <- paste0(gsub(",", " - ", gsub("[[]|[]]|[)]", "", names(attr(colcode, "table")))))
  plot(zip, border=NA, col=colcode)
  legend("left", # position
         legend = legcode, #names(attr(colcode, "table")),
         fill = attr(colcode, "palette"),
         cex = 0.6,
         border=NA,
         bty = "n")
}

# Now plot
WellFun(zip$n, "Blues")
```

```
## Warning: var has missing values, omitted in finding classes
```

```
title(main = "Number of wells by zip code")
```



3. Make a screen shot to copy the map from ArcGIS and paste it in a word document for grading..

```
plot(zip)
points(well_sp, col = adjustcolor("darkblue", alpha.f=0.5),
```

```
       pch = 15, cex = 0.3)
points(orlando_sp, col = adjustcolor("darkred", alpha.f=0.5), pch = 16,
       cex = 1)

legend("bottomleft",
       pch = c(16,15),
       col = adjustcolor(c("darkred", "darkblue"), alpha.f = 0.5),
       legend = c("Dry cleaners", "Wells"),
       bty = "n",
       pt.cex = c(1, 0.5))
title(main = "Florida Wells and Orlando dry cleaners")
```