STAT 231: Problem Set 6A

Jack Dove

due by 5 PM on Monday, October 5

In order to most effectively digest the textbook chapter readings – and the new R commands each presents – series A homework assignments are designed to encourage you to read the textbook chapters actively and in line with the textbook's Prop Tip of page 33:

"**Pro Tip**: If you want to learn how to use a particular command, we highly recommend running the example code on your own"

A more thorough reading and light practice of the textbook chapter prior to class allows us to dive quicker and deeper into the topics and commands during class. Furthermore, learning a programming lanugage is like learning any other language – practice, practice, practice is the key to fluency. By having two assignments each week, I hope to encourage practice throughout the week. A little coding each day will take you a long way!

Series A assignments are intended to be completed individually. While most of our work in this class will be collaborative, it is important each individual completes the active readings. The problems should be straightforward based on the textbook readings, but if you have any questions, feel free to ask me!

Steps to proceed:

1. In RStudio, go to File > Open Project, navigate to the folder with the course-content repo, select the course-content project (course-content.Rproj), and click "Open"
2. Pull the course-content repo (e.g. using the blue-ish down arrow in the Git tab in upper right window)
3. Copy ps6A.Rmd from the course repo to your repo (see page 6 of the GitHub Classroom Guide for Stat231 if needed)
4. Close the course-content repo project in RStudio
5. Open YOUR repo project in RStudio
6. In the ps6A.Rmd file in YOUR repo, replace "YOUR NAME HERE" with your name
7. Add in your responses, committing and pushing to YOUR repo in appropriate places along the way
8. Run "Knit PDF"
9. Upload the pdf to Gradescope. Don't forget to select which of your pages are associated with each problem. *You will not get credit for work on unassigned pages (e.g., if you only selected the first page but your solution spans two pages, you would lose points for any part on the second page that the grader can't see).*

1. PUG Shiny Project

A reminder that the wrangled dataset checkpoint is this Thursday (October 8). For this checkpoint, you'll be submitting *your code* written to acquire, ingest, and wrangle the data in a file titled "data_wrangling" (e.g, `data_wrangling.R` or `data_wrangling.Rmd`). The last lines in your data wrangling file should be code that outputs your wrangled dataset as a permanent file (e.g., an R dataframe or csv file). The purpose of this is to keep your wrangling code separate from your Shiny app code. In your Shiny app code, you can then load the wrangled dataset that you saved and jump right into (or more quickly into) defining the ui and the server functions.

I will be running the "data_wrangling" code, so make sure your repository is organized (so I can easily find the data_wrangling file!) and make sure the code is reproducible. The most common reproducibility error I've encountered with student work is that students have an object saved in their local environment that is not defined in the code. A good way to check the reproducibility of your code is to run it from a completely clean environment.

Further details can be found in the project guidelines document ("PUG_Shiny_Project.pdf") on Moodle. I've also added the document to your GitHub Shiny repositories.

To make the most of Tuesday's class time, your team should coordinate some work (e.g., acquiring and wrangling the data) to be done before class on Tuesday. That way, I can help address questions that may arise as you're working, and you can check in with each other on your progress.

If you haven't already, please get in touch with your team members to coordinate your plan. Briefly describe your group's plan (e.g., "I'll be working to get X done before Tuesday's class, and the others will be working on Y and Z." – or – "We're planning to meet on X day at Y time via Zoom to do Z together").

> ANSWER: We've divided the project into three segments: NFL results, betting odds, and predictions. John is wrangling the year by year historical NFL score data, I'm wrangling team by team betting performance data, and Michael is working to wrangle data for a logistic regression model predicting win probability.

2. Working with spatial data

ggmap

Unfortunately, the `ggmap` package now requires registering with Google Cloud Platform (GCP) and setting up an API key. In the past, the process to do this was smooth for a few students, but quite bumpy for many others, and – even in the smoothest cases – it does take some time to set up. Given that we only have one class day to explore working with spatial data in R, I did not think it was worth the effort during this semester to have everyone set up a GCP account and API keys.

As such, you will not be able to code along with much of the code in Chapter 14. I still encourage you to read the text around the code, but in terms of what to focus on in this Chapter, Section 14.2 (introducing shapefiles) and Sections 14.3.4 and 14.4.5 (using dynamic maps with leaflet) will be most important.

a. shapefiles

Section 14.2 introduces shapefiles, and includes an example of working with a shapefile to re-create Snow's cholera map. Run the code below (line-by-line to understand what each part is doing). Confirm that you get a figure similar to that of Figure 14.2 in the textbook.

Note that the `rgdal` package is loaded in the set-up code chunk above.

```
# insert the path to where you want to save the data
# a good choice would be your problem sets folder within your git repo
my_path <- "~/Desktop/Data Science/Stat231JackDove/Homeworks"

# downloads a zip file from the internet website
download.file("http://rtwilson.com/downloads/SnowGIS_SHP.zip"
              , destfile=paste0(my_path,"/SnowGIS_SHP.zip"))

# unzips the file
# if you look in the location of "my_path", you should now see
# a number of .shp, .shx, .dbf files, etc.
unzip(zipfile = paste0(my_path,"/SnowGIS_SHP.zip")
      , exdir = my_path)

# the readOGR function is from the rgdal package
CholeraDeaths <- readOGR(
  dsn=paste0(my_path,"/SnowGIS_SHP"),
  layer="Cholera_Deaths")
```

```
## OGR data source with driver: ESRI Shapefile
## Source: "/Users/jackdove/Desktop/Data Science/Stat231JackDove/Homeworks/SnowGIS_SHP", layer: "Cholera
## with 250 features
## It has 2 fields
```

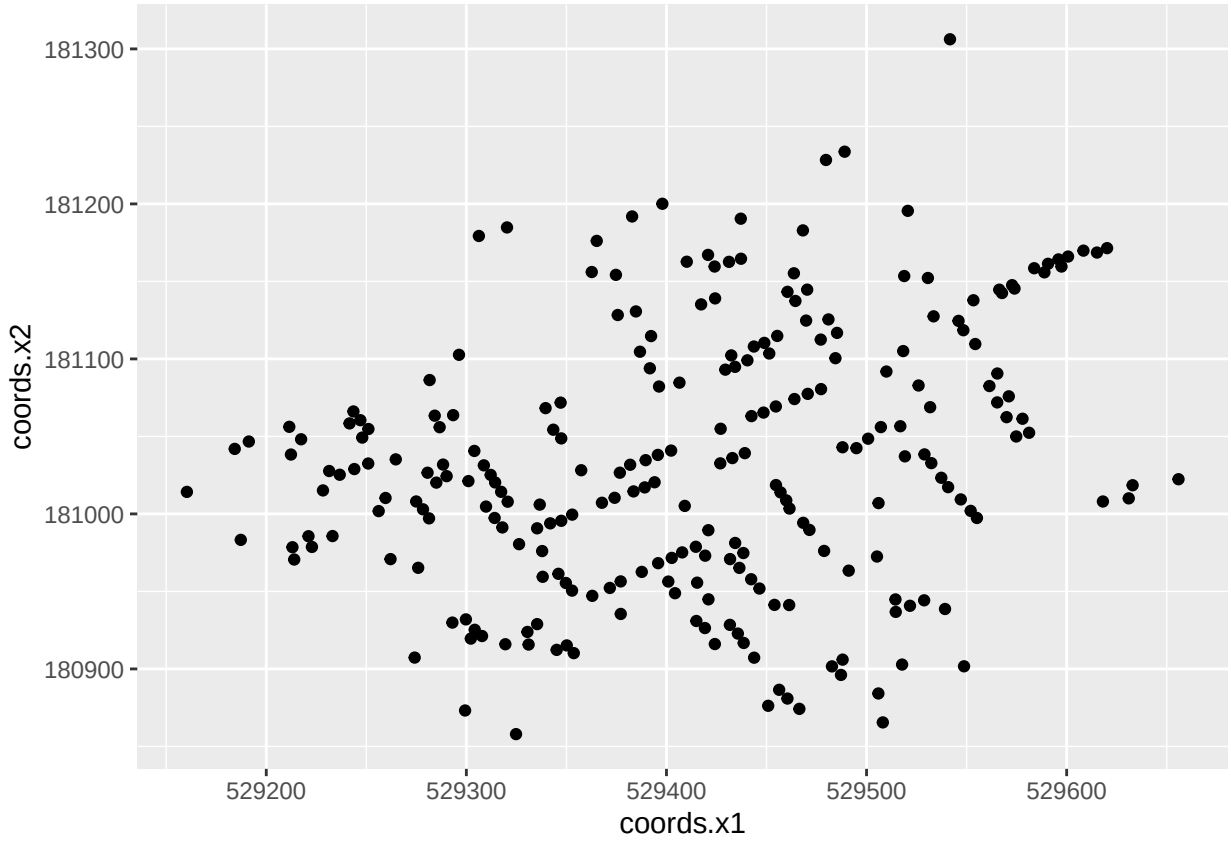
```
# I don't find the summary too helpful here
# but its good to confirm you are at least
# seeing "Object of class SpatialPointsDataFrame"
# to confirm you have the correct object
summary(CholeraDeaths)
```

```
## Object of class SpatialPointsDataFrame
## Coordinates:
##                min      max
## coords.x1 529160.3 529655.9
## coords.x2 180857.9 181306.2
## Is projected: TRUE
## proj4string :
## [+proj=tmerc +lat_0=49 +lon_0=-2 +k=0.9996012717 +x_0=400000
## +y_0=-100000 +ellps=airy +units=m +no_defs]
## Number of points: 250
## Data attributes:
##        Id        Count
##  Min.   :0   Min.   : 1.000
##  1st Qu.:0   1st Qu.: 1.000
##  Median :0   Median : 1.000
```

```
##  Mean   :0   Mean   : 1.956
##  3rd Qu.:0   3rd Qu.: 2.000
##  Max.   :0   Max.   :15.000
```

```
cholera_coords <- as.data.frame(coordinates(CholeraDeaths))

ggplot(data = cholera_coords) +
  geom_point(aes(x = coords.x1, y = coords.x2))
```



Let's add the pump locations:

```
Pumps <- readOGR(
  dsn=paste0(my_path,"/SnowGIS_SHP"),
  layer="Pumps")
```

```
## OGR data source with driver: ESRI Shapefile
## Source: "/Users/jackdove/Desktop/Data Science/Stat231JackDove/Homeworks/SnowGIS_SHP", layer: "Pumps"
## with 8 features
## It has 1 fields
```

```
pump_coords <- as.data.frame(coordinates(Pumps)) %>%
  mutate(type = "Water pump")

both_coords <- cholera_coords %>%
  mutate(type = "Cholera death") %>%
  bind_rows(pump_coords)
```

```
ggplot(data = both_coords) +
  geom_point(aes(x = coords.x1, y = coords.x2
                 , color = type
                 , shape = type)) +
  labs(color = "", shape = "")
```

This is not much of a MAP yet!

b. leaflet

Figure 14.7 in MDSR (page 329) shows the death and pump locations overlaid on a static map of London using `ggmap`. We can't use `ggmap`, but we can create a dynamic version of this figure with `leaflet`. (Note that the `leaflet` package is loaded in the set-up code chunk above.)

Run the code below (one object at a time to see what each part is doing). You should get a dynamic map with lots of navy blue dots (representing the location of cholera deaths) and some red dots (representing the location of the pumps). Zoom in and out of the map to confirm that:

- there is a death in the middle of Hopkins Street
- there is a pump near the intersection of Brewer Street and Lexington Street

When you knit this file to a PDF, the map will be static (and overcrowded looking). That's fine; there's nothing more you need to do for this problem set. Nice work!

```
# transform cholera data to same coordinate system as leaflet
cholera_latlong <- CholeraDeaths %>%
  spTransform(CRS("+init=epsg:4326")) %>%
  as.data.frame() %>%
  mutate(type = "Cholera death")

# transform pump data to same coordinate system as leaflet
pumps_latlong <- Pumps %>%
  spTransform(CRS("+init=epsg:4326")) %>%
  as.data.frame() %>%
  mutate(count = 1, type = "Pump")

# join cholera and pumps datasets together
both_latlong <- cholera_latlong %>%
  bind_rows(pumps_latlong)

# create dynamic map
m <- leaflet() %>%
  addTiles() %>%
  addCircleMarkers(data = both_latlong,
             lng = ~coords.x1,
             lat = ~coords.x2,
             radius = ~ifelse(type == "Cholera death", yes = 6, no = 10),
             color = ~ifelse(type == "Cholera death", yes = "navy", no = "red"),
             stroke = FALSE,
             fillOpacity = 0.5
           )

library(devtools)
```

```
## Loading required package: usethis
```

```
install_github("wch/webshot")
```

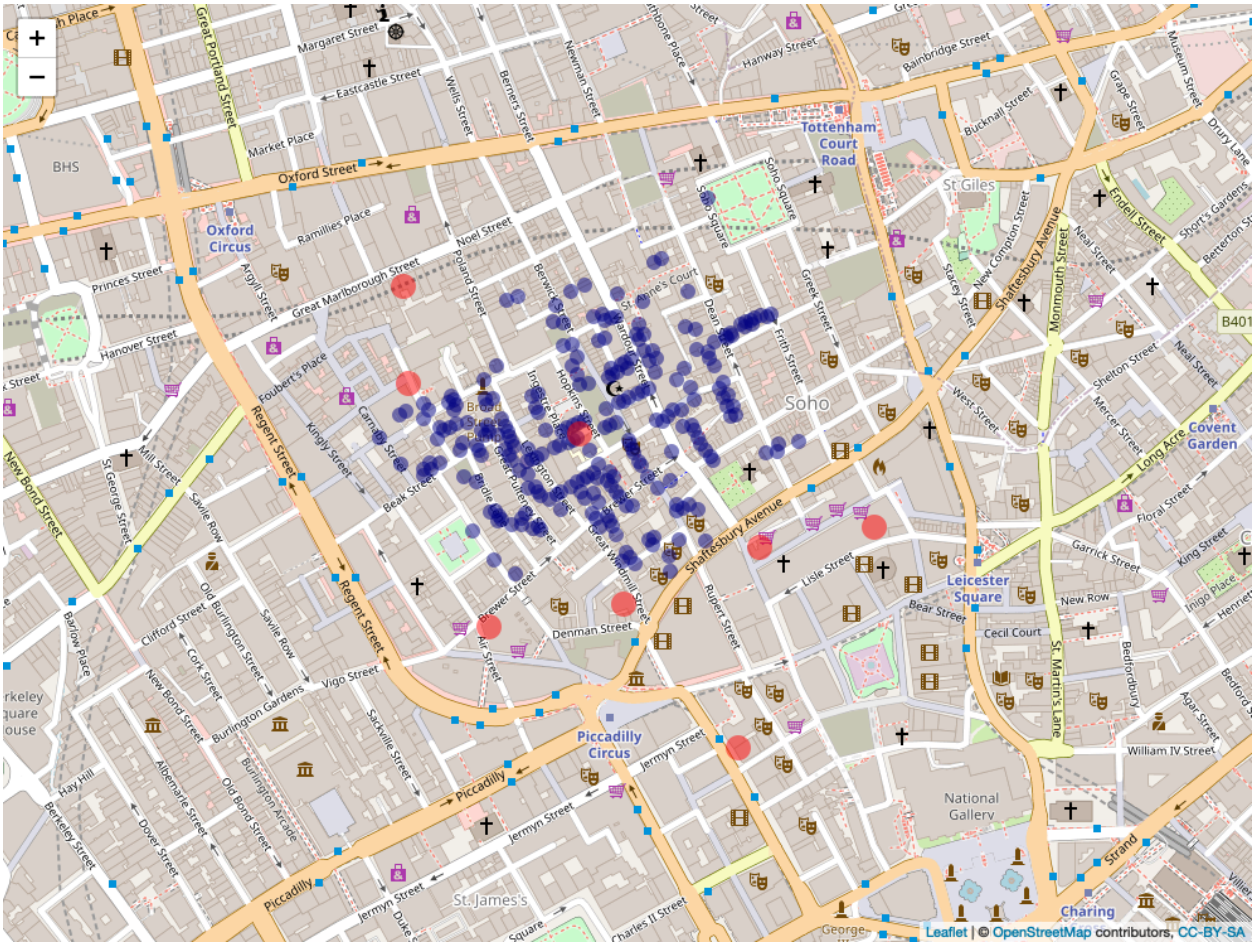
```
## Skipping install of 'webshot' from a github remote, the SHA1 (d206bd7d) has not changed since last i
##   Use `force = TRUE` to force installation
```

```
## load packages
library(leaflet)
library(htmlwidgets)
library(webshot)

webshot::install_phantomjs()
```

```
## It seems that the version of `phantomjs` installed is greater than or equal to the requested version
```

```
saveWidget(m, "temp.html", selfcontained = FALSE)
webshot("temp.html", file = "Rplot.png",
        cliprect = "viewport")
```


Oxford Street
Regent Street
Piccadilly
Piccadilly Circus
Soho
Leicester Square
Covent Garden
Tottenham Court Road
Oxford Circus
St Giles
St. James's
National Gallery
Charing
Shaftesbury Avenue
Leaflet | © OpenStreetMap contributors, CC-BY-SA