Mapping “Badness”

“Even before you understand them, your brain is drawn to maps”

Ken Jennings, author and Jeopardy champ

We have looked Maps have always been in the realm of cartographers and artists, but the barrier to entry realm has been steadily opening over the last few decades. Technology computers and the Internet , now any yahoo can map out data on Google (that play on words was odd and intentional). And as we saw in Chapter 4, it’s possible to pull geo-location of IP addresses and get spatial data, and that’s what we’re going to focus on.

How do maps help us within information security?

The data we are going to use for the first half of this chapter comes to us from Symantec, who shared a list of clients infected with the ZeroAccess Botnet as collected over a 24 hour period. But they didn’t share the IP But rather than share the IP address

The question we’ll really want to target in this chapter though, is whether or not we should map the virtual world into the physical?

24 hour period

Who is more likely to get a bot infection?

Who is more likely to be infected with ZeroAccess?

Is some proportion of the population more at risk of a ZeroAccess infection?

Simplifying Maps

It’s easy to get all wrapped up thinking that visualizing spatial data (maps) are special, complicated or will somehow take a lot more effort. But with the right tools (and there are plenty available) working with spatial data can not only be relatively simple, but pretty darn fun. In order take some of the mystique out of maps, let’s start by loading up the latitude and longitude points we got from Symantec and just treat them as x,y coordinates and create a simple scatter plot:

# read the CSV with headers

**za <- read.csv("ch05/data/zeroaccess.csv", header=F)**

# create a ggplot instance with zeroaccess data

**gg <- ggplot(data=za, aes(x=long, y=lat))**

# add the points, set transparency to 1/40th

**gg <- gg + geom\_point(size=1, color="#000099", alpha=1/40)**

# add axes labels

**gg <- gg + xlab("Longitude") + ylab("Latitude")**

# simplify the theme for aesthetics

**gg <- gg + theme\_bw()**

**print(gg)**

Figure 5.1 Basic Scatterplot using Latitude and Longitude [FILENAME 793725c05f001]

See the map created out of points in figure 5.1? This works with our data because we have over 800,000 data and one point is covering more than a large city. We made it a little less menacing by setting the alpha (transparency of the color as we covered in chapter 3) to be 1/40th of a full color. But just from this basic scatter plot, we can see the density in the eastern half and west coast of the U.S. and most of Europe is covered. We see some concentration in Brazil, and India is outlined quite well. One interesting thing to note here is that China has almost no density and Japan is clearly visible. But at this point, we can only make guesses as to what’s going on with what looks like a significant difference in Asian countries. Let’s just keep focusing on the graphics for now and add some detail to the map.

Now comes the secret sauce about maps: it’s all about the projection. Everyone is familiar with the Cartesian coordinate system, if not by name then by sight. It’s the fancy name for coordinate system of the simple x,y plotting we just did. By plotting the latitude and longitude points as same-spaced x and y coordinates, we can see the map, but it looks a little odd. South America looks like it is stretched long for example. This is where map projections come in because we need to “project” a three-dimensional spherical world onto a two-dimensional flat canvas. This creates some problems since there are multiple ways to do that projection as we see in figure 5.2

Figure Map Projects from Yau (?) [FILENAME]

Let’s go back to Figure 5.1, it’s a little hard to know where all those points land unless we are good at world geography. Let’s recreate that image and create a map with a specific projection and then add the points on top of it. Luckily, within R, most of the basic map data is already available with a few packages installed. ggplot2 has a function of map\_data() that wraps the maps package to return a ggplot2-compatable data frame. It’s possible to load up the data for a world map with a single command:

# load map data of the world

**world <- map\_data(“world”)**

This loads up just over 25 thousand rows of map data into a data frame (less after we filter out Antartica). Remember we can explore the data with commands like str(), head() or summary(). We can plot the countries by tracing a path along the the lat/long pairs in the map data, which has the effect of drawing the country borders. We have to group the paths by the column labeled group (which in this data groups the country) and the data frame must be sorted in order (we will have to remember this later). We will then call coord\_map() to create the map projections and we’ll use a simple black and white theme on it with theme\_bw(). Once we have the countries traced out we will then add the points from the ZeroAccess data on the map as if we are creating a scatter plot like we did before.

# load map data of the world

**world <- map\_data("world")**

# nothing personal penguins, but strip out Antarctica

**world <- subset(world, world$region!="Antarctica")**

# load world data into ggplot object

**gg <- ggplot(data=world, aes(x=long, y=lat))**

# trace along the lat/long coords by group (countries)

**gg <- gg + geom\_path(aes(group=group), colour="gray70")**

# now project using the mercator projection

# try different projections with ?mapproject

**gg <- gg + coord\_map("mercator", xlim=c(-200, 200))**

# load up the points, overiding the default data set

**gg <- gg + geom\_point(data=za, aes(long, lat),**

**colour="#000099", alpha=1/40, size=1)**

# add axes labels and theme

**gg <- gg + xlab("Longitude") + ylab("Latitude")**

**gg <- gg + theme\_bw()**

**print(gg)**

Figure 5.3 Worldwide ZeroAccess Infections [FILENAME 793725c05f003]

Now that’s a real map, but what can we learn from it? The answer is not much, it doesn’t tell us much more than the ZeroAccess botnet is an international traveller (and nobody should be surprised by that).

How many ZeroAccess infections per country?

It’s very difficult to look at figure 5.3 and determine which countries have the most infections. It looks like the U.S. and Europe are covered, but it’s difficult to see specific comparisons in this type of picture, so let’s try a different type of map. We need to count how many infections we have in each country and then we can visualize that with a chloropleth. A chloropleth is a map where the country is filled with some type of color attribute associated with the data. For our first chloropleth, we will have to figure out which country the points are in and then we will use a single continuous color scale to represent that quantity. To convert latitude and longitude to a country, we will adapt a function from github and call our function latlong2map(). That function will accept a data frame of longitude and latitude pairs along with the name of a map to translate onto.

# slightly modified verison of Ryan Weald’s function

# https://gist.github.com/rweald/4720788

**latlong2map <- function(pointsDF, mapping) {**

# load up the map data

**local.map <- map(mapping, fill=TRUE, col="transparent", plot=FALSE)**

# pull out the IDs from the name

**IDs <- sapply(strsplit(local.map$names, ":"), function(x) x[1])**

# Prepare SpatialPolygons object

**maps\_sp <- map2SpatialPolygons(local.map, 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, maps\_sp)**

# Return the names of the Polygons object containing each point

**mapNames <- sapply(maps\_sp@polygons, function(x) x@ID)**

# now return a vector of names that match the points

**mapNames[indices]**

**}**

We will get back a vector of names (country names in this case) and then we want to count up how many times the country appears with the table() command. Next we’ll want to merge() the count of countries with the map data (then reorder it for the plotting). By merging our data directly into the map data we can then associate the fill of the country with an attribute in our data, specifically the count of infections in that country. We will use the scale\_fill\_gradient2 function within ggplot2 to get the color gradient associated with the quantity of infections.

# convert ZeroAccess long/lat into country names from world map

**zworld <- latlong2map(data.frame(x=za$long, y=za$lat), "world")**

# count up points in the country and conver to data frame

**wct <- data.frame(table(zworld))**

# label the country as "region" to match map data

**colnames(wct) <- c("region", "count")**

# merge will match on "region" in each and add "count" to "world"

**za.chloro <- merge(world, wct)**

# now we sort the map data to original sequence

# otherwise the map is disasterous

**za.chloro <- za.chloro[with(za.chloro, order(group, order)), ]**

# and plot

**gg <- ggplot(za.chloro, aes(x=long, y=lat, group=group, fill=count))**

**gg <- gg + geom\_path(colour="#666666") + geom\_polygon()**

**gg <- gg + coord\_map("mercator", xlim=c(-200, 200), ylim=c(-60,200))**

**gg <- gg + scale\_fill\_gradient2(low="#FFFFFF", high="#4086AA",**

**midpoint=median(za.chloro$count))**

**gg <- gg + theme\_plain()**

**print(gg)**

Figure 5.4 Chloropleth of ZeroAccess Infections [FILENAME 793725c05f004]

And voila, we get a rather good-looking map and it look like the U.S. has the market cornered on ZeroAccess infections. But because it’s very difficult to tell quantity by color density (we’ll cover that in Chapter 6), we want to take a step back for a moment and get a feel for the proportion of infections in the U.S. Look back at the wct variable we created above. Let’s see what that data looks like:

**head(wct)**

region count

1 Afghanistan 53

2 Albania 1166

3 Algeria 3014

4 Andorra 4

5 Angola 160

6 Argentina 6016

We can use the wct variable to get an understanding of the proportion of infections in the U.S.:

# for each wct$count, divide by sum

**perc <- wct$count/sum(wct$count)**

# covert to a readable format

**wct$perc <- round(perc, 4)\*100**

# now order the highest proportions on top

**wct <- wct[with(wct, order(perc, decreasing=T)), ]**

# look at the top few entries.

**head(wct)**

# output:

region count perc

148 USA 261627 35.23

24 Canada 35607 4.79

74 Japan 33590 4.52

145 UK 31813 4.28

50 Germany 27336 3.68

71 Italy 25717 3.46

So to answer our question, “**How many ZeroAccess infections per country?**” we could have just created this table in the beginning, but the gap between the U.S. at 35% and the rest of the world is much more prominent when displayed visually. We should also keep in mind that these are just total counts. At this point, the 35% represents a proportion within our data, and we should not infer more into this data without further analysis.

Let’s get back to our main question though, “**Who is more likely to be infected with ZeroAccess?**” We want to simplify our data for the U.S. and understand how that may change our question. We want to do this not just because working with over 800,000 data points can be a bit slow on some systems, but also it will be much easier for us to focus in on the U.S. because of our knowledge of the geography and accessibility of data for us. Although this does change the type of question we are answering. We are now trying to answer “**Who is more likely to be infected with ZeroAccess in the U.S.?**” because we cannot project what we learn from infections in the U.S. to other countries and/or cultures. That is going beyond and outside the data we are looking at and we just do not know if we can do that. In some cases it may work out that we can project like that, but it should never be assumed.

If we attempt to plot a U.S. map and then project all the points on it, the auto-scaling feature in ggplot we get a rather funny picture because it will show all of the world points in the data set, but only trace out the U.S. map. Therefore we have to remove data that are not in the U.S. We can use the latlong2map() command again and this time, map the points to U.S. states and anything that does not get mapped to a U.S. state will be retuned as the NA value, which can then be filtered out of our data.

**zstate <- latlong2map(data.frame(x=za$long, y=za$lat), "state")**

# select rows from za where the zstate is not NA

**za.state <- za[which(!is.na(zstate)), ]**

And now we can make a nice map of the continental U.S. showing all the ZeroAccess infections in the country. But let’s make it even nice by removing all the extra things on the map. We will create a theme for ggplot that will not show the axes or the box around the map, and have just a nor-frills, plain old simple map. By creating this theme as a function, we can just add on theme\_plain() instead of having to type this in for every plot when we call ggplot()from here on out.

# create a plain theme for ggplot maps

**theme\_plain <- function() {**

**theme(axis.title = element\_blank(),**

**axis.text = element\_blank(),**

**panel.background = element\_blank(),**

**panel.grid = element\_blank(),**

**axis.ticks.length = unit(0, "cm"),**

**axis.ticks.margin = unit(0, "cm"),**

**panel.margin = unit(0, "lines"),**

**plot.margin = unit(c(0,0,0,0), "lines"),**

**complete=TRUE)**

**}**

# load map data of the U.S.

**state <- map\_data("state")**

**gg <- ggplot(data=state, aes(x=long, y=lat))**

**gg <- gg + geom\_path(aes(group=group), colour="gray80")**

**gg <- gg + coord\_map("mercator")**

**gg <- gg + geom\_point(data=za.state, aes(long, lat),**

**colour="#000099", alpha=1/40, size=1)**

**gg <- gg + theme\_plain()**

**print(gg)**

Figure 5.5 ZeroAccess Infections in the U.S. [FILENAME 793725c05f005]

Look at Figure 5.5 and think how that looks. Does it look strange? This is where we really have to be careful because after working with spatial data, we can tell you this sure looks like map showing population density and not infections. Looking at figure 5.5 we find ourselves asking a slightly different question. **Could ZeroAccess infections just be a reflection of the population?** We could apply a statistical technique called regression analysis (and we will later) but let’s stick with pictures and create another chloropleth, but this time we will break up the data and count based on the U.S. states.

The Potwin effect

But at this point, we are digging deeper than country and we will have to account for something we call “the Potwin effect” after the town by that name in Kansas with a population of 449. The population is important because if we take a look at this data, we’d see we have 12,643 reported ZeroAccess infections in the town of Potwin, Kansas. See, we’ve played this game before and when we first stumbled into this, we spent days trying to understand why Potwin, KS was so odd. We realized that these couldn’t be valid entries and looking at the data, we noticed the entries were oddly rounded off to integers and they were all “38,-97”. Finally it dawned on us. IP geo-location services should always know what country an IP address is in because the IANA records are clear about that. But if the geo-location service cannot get any more specifics beyond the country, they return a rounded-off integer location near the geographic center of the country. In the U.S., the geographic center is just outside of Potwin, KS. For this purpose, they are “unknown U.S. locations” and not really in Kansas, so we are going to remove these data points the next bit of code as not to unfairly assign infections to Kansas.

In this map, we want to use color again to show quantity, but rather than just using a single hue (a fancy term for the color of the rainbow), let’s use a diverging color scheme and assign the mid-point of the range to the mean count per state. This will allow us to show states with above average infection counts with one hue and the below average state with another. As a side note, let’s also change the projection from the Mercator projection to the Polyconic. That projection looks odd at the world level, but puts a nice slope and curve in a U.S. map and we want to try a different projection.

# create a chloropleth of the U.S. states

# because all of these vectors are from the same source (za),

# we can cross the indexes of the vectors

# pull out those that are not NA, and take care of Potwin effect

**state.index <- which(!is.na(zstate) & za$lat!=38 & za$long!=-97)**

# now create a count of states and filter on those indexes

**sct <- data.frame(table(zstate[state.index]))**

**colnames(sct) <- c("region", "count")**

# merge with state map data

**za.sct <- merge(state, sct)**

# Now plot a choropleth using a diverging color

**colors <- suda.pal(5, "div")**

**gg <- ggplot(za.sct, aes(x=long, y=lat, group=group, fill=count))**

**gg <- gg + geom\_polygon(colour="black")**

**gg <- gg + coord\_map("polyconic")**

**gg <- gg + scale\_fill\_gradient2(low=colors[5], mid=colors[3],**

**high=colors[1],**

**midpoint=mean(za.sct$count))**

**gg <- gg + theme\_plain()**

**print(gg)**

Figure 5.6 Chloropleth of U.S. States with ZeroAccess [FILENAME 793725c05f006]

Wow, this is looking nice, but don’t get too excited as this is relatively useless for comparing the states. We can easily see that California, Texas, Florida and New York are above average, but it’s also worthwhile to know that the 4 most populated states are also California, Texas, New York and Florida in that order. In other words, we may just be seeing a reflection of population in this map, so we have to normalize this data to the population. In order to normalize we could take a multiple approaches. The two simplest approaches are to normalize the population to the infections and ask, “**How many people per one infection?**” or we could reverse that and ask, “**What proportion of the people are infected?**” The difference is subtle, and in this case we will do the former because we will get whole numbers and it will be a little easier to conceptualize. In order to get the number of people to infection, we will divide the population in a state by the number of infections in that state. In this case, we’ve already scraped the data from http://www.internetworldstats.com/stats26.htm and made the data available in an easy format on the book website. We are also including the count of internet users if you would like to try to create a chloropleth normalized on estimated internet users per state (it’s a slightly prettier picture).

# read in state population and internet users

# data scraped from http://www.internetworldstats.com/stats26.htm

**users <- read.csv("data/state-internets.csv", header=T)**

# all the state names are lower case in map data, so convert

**users$state <- tolower(users$state)**

# now merge with the sct data from previous example

# merge by sct$region and users$state

**za.users <- merge(sct, users, by.x="region", by.y="state")**

# calculate people to infection

# change this to internet users if you would like to try that

**za.users$pop2inf <- round(za.users$population/za.users$count, 0)**

# and create a simple data frame and merge

**za.norm <- data.frame(region=za.users$region,**

**count=za.users$pop2inf)**

**za.norm.map <- merge(state, za.norm)**

# now create the chloropleth

**gg <- ggplot(za.norm.map, aes(x=long, y=lat, group=group, fill=count))**

**gg <- gg + geom\_polygon(colour="black")**

**gg <- gg + coord\_map("polyconic")**

**gg <- gg + scale\_fill\_gradient2(low=colors[5], mid=colors[3],**

**high=colors[1],**

**midpoint=mean(za.norm.map$count))**

**gg <- gg + theme\_plain()**

**print(gg)**

Figure 5.7 Normalized ZeroAccess Infections: Number of People in the state per one infection [FILENAME 793725c05f007]

Remember California, Texas, Florida and New York having the highest infection counts? When we normalize to population California and New York drop to below average with one infection per 1,440 and 1,287 people on average respectively. Using the za.norm data we generated in the above code, we can view the exact counts. Wyoming now sticks out as the most infected state since one in 724 people in Wyoming appear to have ZeroAccess infections.

Is this Weird?

We have to stop for a moment and look at what we have. We have a range of normalized values from 724 in Wyoming to 1,550 in Washington State. Does this mean that the citizens in Wyoming are much more careless than those in Washington? Or are the range of observations from just natural variation in our measuring accuracy and the world? Is Wyoming the most infected state because someone had to be in last place and in this data it just so happened to be Wyoming? This may be easier to answer if we collected this data over time and had more confidence in the accuracy of geo-location services. But we have what we have and let’s run a few tests for outliers in this data. Going back to the work of John Tukey, the boxplot was devised to visually show a distribution and in doing that, it attempts to estimate (and display) any possible outliers. We will cover the details of the boxplot later, but let’s use the default R graphics and whip out a quick boxplot (and save the results into popbox). Usually, when we are just looking at the data for ourselves we will not set the title, but let’s be thorough in our graphic creations:

# create a box plot of the count

**popbox <- boxplot(za.norm$count,**

**main="Distribution of Normalized\nState Infections")**

Figure 5.8 Normalized State Infections [FILENAME 793725c05f008]

Looks like we may have a few outliers and looking at the return value of the boxplot function we can see which ones:

**za.norm[za.norm$count %in% popbox$out, ]**

# output:

region count

8 district of columbia 777

43 utah 1536

44 vermont 1525

46 washington 1550

49 wyoming 724

But let’s run another test for outliers and calculate what’s known as a z-score, which will help us get a feel for just how much of an outlier things are. In order to do this calculation, we will need to know the standard deviation and mean of our distribution. Then, for each value in the distribution, we will calculate how many standard deviations from the mean the observation is. That is, we will subtract the mean from each value and divide by the standard deviation. We want to compare what we see in our distribution to something known as the “empirical rule” of a standard normal distribution. In a normal distribution (the famous bell curve or Gaussian distribution), we should expect that roughly 68% of the distribution will fall within 1 standard deviation (above or below) of the mean, further 95% of the data should fall within 2 standard deviations, and 99.7% should be within 3. When using this approach, generally anything above 3 standard deviations could be labeled as an outlier, and we could look at anything more than 2 standard deviations as possible outliers.

# get the standard deviation

**za.sd <- sd(za.norm$count)**

# get the mean

**za.mean <- mean(za.norm$count)**

# now calculate the z-score and round to 1 decimal

**za.norm$z <- round((za.norm$count-za.mean)/za.sd, 1)**

# truncate the value, get the absolute and add 1

# print a table (count) of entries within each std dev

**print(table(abs(trunc(za.norm$z))+1))**

# output

1 2 3

38 6 5

Looks like those same 5 entries fall within three standard deviations. But knowing that we aren’t sure if population is the best thing to normalize too (perhaps “internet users” would be a better measure, hint, hint). There just is not enough evidence to say we have problem or exemplary states quite yet and so we must answer our question “Is this weird?” with either a squishy “Probably not” or a non-committal “not so sure”.

Taking it down to U.S. Counties

It is difficult to generalize at the state level because, well, it is very general because it is obscuring a whole lot of diversity behind a single label or descriptive statistic. We would be hard-pressed to calculate the influence of income on ZeroAccess infections at the state level. We can get more granular by repeating this process again but at the county level. But we have a few more things to consider as we get into more detailed breakdown of geo-location of IP addresses. Most of the popular IP geo-location services publish estimations of their accuracy beyond country. For example, the service used on this data claims just over 4 out of 5 entries are accurate to about 25 miles and about 1 out of 7 are resolved to an incorrect city. We should pause the analysis here and discuss a statistical concept:

Here’s another statistical concept: natural variations will cancel out more often then stack up. This is kind of a tricky concept so we’ll use an example. Let’s say we are manufacturing a physical part and we want it to be 100 millimeters long. But natural variation in the quality of materials and manufacturing process produces parts between 98 and 102 millimeters. Within engineering, we are taught if we stack up 100 of those parts, we could expect variation of plus or minus 2 x 100. Meaning it is possible that all 100 parts will be 98 millimeters and so we expect any variation to be compounded the further we go out.

But in statistics, if we can assume that each part has an equal chance of being any length within tolerance (something we want to blindly assume), they will begin to cancel each other out, but let’s model this. Let’s generate 100 parts and have them uniformly be “manufactured” between 98 and 102 millimeters, then let’s take the average (mean, could also be sum or something other measurement, but mean works here). Our engineering brains say this could be between 98 and 102, but let’s see:

#setting seed for reproducibility

**set.seed(1492)**

**mean(runif(100, min=98, max=102))**

100.0141

After one run, we got 100.0141, but let’s manufacture maybe ten thousand sets of 100 stacked parts and see how many get to the edge of our range, surely if it’s possible, we should see at least a few sets within ten thousand sets push towards the edge, right?

#setting seed for reproducibility

**set.seed(1492)**

# iterate seq(10000) times, generate a set of 100 parts and calc mean

**parts <- sapply(seq(10000), function(x) mean(runif(100, min=98, max=102)))**

# show the min and max of these parts

**range(parts)**

99.57977 100.47559

What is up with this? We can visualize all of our parts by generating a quick histogram by running hist(parts). We see a nice symmetric distribution centering around 100. Even though the parts could all be 98 or 102, the variation will cancel out, especially as the sets increase (rather then 100 in a set, try 1,000 or 10,000 in the runif command). As we add more parts with variance, we become more likely to cluster around the mean. There are actually two concepts at play here.

Regression to the mean the more parts

, if we design a component to be 100 centimeters long and we have a tolerance of 2 centimeters. If we chain 10 of these together

plus or minus 2

Down to Counties

We begin by calling the same latlong2map function on the same ZeroAccess data, but ask it to translate to the county names. Keep in mind, there are over 3000 counties in the U.S. and over 800,000 latitude/longitude pairs to go through, so depending on the system, this could take a few seconds or so to run. Then, like last time we want to ignore anything that doesn’t resolve in the U.S. (is set to NA in the data) and account for the Potwin effect. But now rather than count things with table and tossing into a data frame, we have to do some transformation. The county names come back from latlong2map as a single text string in the “state, county” format. We will use the strsplit() function to split the county names and that returns a list object, so we convert it to a vector with the unlist() function. Now this will be one long vector with the values alternating state and county, which is okay because we’ll transform this into a matrix with 2 columns (state and county) with the ncol=2 argument and tell it to go row by row (rather than column by column). The result is then converted into a data frame, along with the count of infections in each county.

## now to county

**county <- latlong2map(data.frame(x=za$long, y=za$lat), "county")**

**za.county <- county[which(!is.na(county) & za$lat!=38 & za$long!=-97)]**

# count the occurances

**county.count <- table(za.county)**

# need to convert "county, state" into a data frame

# so we split it out by comma

**temp.list <- strsplit(names(county.count), ",")**

# convert the list into a vector

**temp.list <- unlist(temp.list)**

# force the vector into a 2 column matrix, filling row by row

**temp.matrix <- matrix(temp.list, ncol=2, byrow=T)**

# and now create the data frame with the count of county infections

**za.county <- data.frame(temp.matrix, as.vector(county.count))**

# finally assign names to the fields

# names match the field names in the county map\_data

**colnames(za.county) <- c("region", "subregion", "infections")**

What we have is a data frame with three columns, the state, county and count of infections and so we label the columns accordingly. Now with this we are able to combine with any other data that is split out by county. One great place to get such data is the U.S. Census Bureau and their website at <http://factfinder2.census.gov/>. Using their “Guided Search” feature, we can select a topic and/or demographic and then the geography of “All U.S. Counties”. This is where the data munging skill comes in handy as there will be A LOT of data there. For the purpose of serving as an example, we’ve pulled out a few statistics by county and made it available on the book website.

* region and subregion are the county and state respectively
* pop is the estimated county population
* high is the proportion of the population over the age of 25 that has a high school or equivalent degree (as a percentage)
* college is the proportion of the population over the age of 25 that has some type of college degree (as a percentage)
* income is the median income for the county
* poppermile is the density of population per square mile in the county
* establishments is the number of registered businesses in the county
* employees is the number of employed people by those establishments

And as luck would have it (for you), this data is in a perfect state so it can be read in and simply merged with the ZeroAccess county data we just created. But there is one special note with the merge command: by default it will drop any rows that are not in both data sets. In our case we have 160 counties not represented in the ZeroAccess data. This could be for a variety of reasons, perhaps the IP geolocation services are inaccurate in those counties or they are just sparsely populated counties and not having infections isn’t weird. Feel free to try and dig into the values, but sure enough, we took a look and 90% of the uninfected counties have a population of less than 10,000. By specifying all.x=T in the merge command, we are telling it to not drop any rows from the “x” data, which is the first passed in, or county.census in our command.

# read up census data per county

**county.census <- read.csv("data/county-census.csv", header=T)**

# notice the all.x option here

**za.county <- merge(county.census, za.county, all.x=T)**

# replace all NA's with 0

**za.county$za[is.na(za.county$za)] <- 0**