From Maps to Regression

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

Ken Jennings, author and Jeopardy champ

We have been learning some basics about security data and how to pull meaning from IP addresses. As we briefly discussed in chapter 4, IP addresses could have spatial data associated with them (if we look them up in a geolocation service). But what is the value in doing that? How much can we learn by associating a longitude and latitude with our data? The answer to that is very dependent on what the IP represents and how deep we are willing to go. In order to describe the value of mapping the virtual world into the physical we’re going to begin with a list of over 800,000 latitude/longitude pairs shared by our friends at Symantec. The location data is from client IP addresses infected with the ZeroAccess rootkit, collected over a 24 hours period during the month of July in 2013.

Now that we know these are locations of hosts with ZeroAccess, we could ask a series of questions:

* How is ZeroAccess distributed across geographic areas and is there any significance in how its distributed?
* Who is more likely to be infected with ZeroAccess? Do things like education and income affect the rate of infection?
* Are ZeroAccess infections the result of alien visitors?

Obviously, we’re going to hone in on that last question. It is the most important and worthy of some serious research (anyone have some spare grant money?) but, let’s get started and see what we can learn.

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 cannot only be relatively simple, but quite 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]

Figure 5.1 looks remarkably like a world map without even placing borders. This works with our data because we have over 800,000 coordinate pairs and one point is covering more than a large city. We made the points a little less overwhelming by setting the alpha (transparency of the color as we covered in Chapter 3) to be 1/40th of a full color. 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 the moment and add some detail to the map.

There is something unique about maps, because we need to “project” the three-dimensional spherical world onto a two-dimensional flat canvas. When we do that though, we will distort something about the map, either shapes will be distorted, land areas will be shrunk or over-represented and/or distances will be skewed. But for most applications within information security we are simply trying to represent some attribute of, or difference between geographic areas. So the choice of map projections is more about personal preference and aesthetics rather than communicating a specific geographic message. Figure 5.2 shows a few different map projections.

Figure 5.2 Map Projections [FILENAME 793725c05f002-horiz]

If we take another look at Figure 5.1, it’s a little hard to know where all those points land unless we were among the few who didn’t fall asleep during world geography in high school. Let’s recreate that image, build a map with a specific projection, *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. The ggplot2 package has a function called map\_data() that wraps the maps package to return a ggplot2-compatible 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 just over 25 thousand rows of map data into a data frame, which means, as we’ve seen from Chapter 3, we can explore any and all of this data with commands like str(), head() or summary(). We can plot the countries by tracing a path along the latitude and longitude pairs in the map data, which has the effect of drawing the country borders. Paths are grouped by the column labeled group (in this data, groups are the country) and the data frame must be sorted in order (an important detail, as we’ll see later). To create the final map, we call coord\_map() to create the map projections (we will use the Mercator projection for this example) and we’ll use a simple black and white theme on it with the theme\_bw() function. 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 ZeroAccess 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. This map communicates very little besides the fact that the ZeroAccess botnet is an international traveller (and nobody should be surprised by that). Let’s probe a bit deeper and see if we can’t use maps to help us visualize the data a bit better.

How many ZeroAccess infections per country?

It’s very difficult to look at figure 5.3 and determine which countries have the most infections. We can’t expect someone (or ourselves) to look at a map like this and extract the proportion of bot infections in countries. It looks like the U.S. and Europe are blanketed in bots; 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 the results with a *choropleth*. A choropleth is a map where the country is shaded or filled with color that is then associated with the data. For our first choropleth, we will have to figure out which country the latitude/longitude 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 Ryan Weald 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 (@rweald) 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]**

**}**

The function returns a vector of names (country names in this case), and we can 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 and reorder it for the plotting. By merging our data directly into the map data we can then associate the shading 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.choro <- merge(world, wct)**

# now we sort the map data to original sequence

# otherwise the map is disasterous

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

# and plot

**gg <- ggplot(za.choro, 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.choro$count))**

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

**print(gg)**

Figure 5.4 Choropleth of ZeroAccess Infections [FILENAME 793725c05f004]

Voila! We have a rather good-looking (some might say, “spiffy”) map and it looks like the U.S. has the market cornered on ZeroAccess infections. There would be no way we could learn that from the points in figure 5.3. Though it is very difficult to tell the specific quantity by color density (we’ll cover that in Chapter 6), all we can tell from this type of map is that the U.S. has more infections. To learn just how much more, we can look into the data for a moment to see the proportion of infections in the U.S by examining the wct variable we created above.

**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, gives us proportion of the whole

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

# covert to a readable format, round it and create percentage.

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

# now order the highest percentages 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

We could have just created this table in the beginning to answer our question, “How is ZeroAccess distributed across geographic areas?” The map visually highlights the gap between the U.S. and the rest of the world. We should also keep in mind that these are just total counts and not normalized for population or anything. So at this point, the 35% represents a proportion solely within our ZeroAccess data, and we should not infer more without further analysis.

We can’t lose sight of our goal here though, we need to find a way correlate infections to alien visits. To get closer to answering this, we’ll simplify our data set to the U.S. infections. 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 (especially around alien visitors) for us. But as we change the scope within our data like this, we need to consider how this may change the question we’ll be able to answer. We can no longer generalize about every infection everywhere because we cannot readily transfer what we learn from infections in the U.S. to other countries and/or cultures. We also cannot be sure that the differences in infections in the U.S. would match the differences elsewhere. Those considerations go beyond and outside the data we are looking at, so we’ll be sure to avoid making any broad assumptions as we continue our analysis and present our results.

If we attempt to plot a U.S. map and then project all the points on it, the auto-scaling feature in ggplot will create a rather funny picture. It will show all of the world points in the data set, but only trace out the U.S. map. We should reduce our data size and remove data that are not in the U.S. We can reuse the latlong2map() function , this time mapping the points to U.S. states. From an R-perspective, this means 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)), ]**

Once all that processing is done, we can make a nice map of the continental U.S. showing all the ZeroAccess infections in the country. Though we can make it even nicer by removing all the extra things (“*chartjunk*”, a term coined by Edward Tufte) on the map. To do this, we will create a theme for ggplot that will hide the axes, the border and clear the background to have just a no-frills, plain-old simple map. By creating this theme as a function, we can just call theme\_plain() instead of having to type this in for every plot when we use 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]

Consider Figure 5.5 for a moment. 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. After reviewing Figure 5.5 we find ourselves asking a slightly different question. **Could ZeroAccess infections just be a reflection of the population?** We *could* stop and apply a statistical technique called regression analysis (we will later) but let’s stick with pictures and create another choropleth, this time breaking up the data and perform counts based on the U.S. states.

The “Potwin Effect”

As we dig deeper than just country 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 examine the data, we see we have 12,643 reported ZeroAccess infections in the town of Potwin, Kansas. We first stumbled across this “anomaly” in a different (and more subtle) analysis and spent days trying to understand why Potwin was so odd. We realized that these couldn’t be valid entries after we had some crazy ideas about Potwin trying to justify the data. Finally, we remembered that there were several data points that were strangely rounded off to integers and they were all “38,-97”.

Then, it dawned on us. IP geolocation services should always know what country an IP address is in because the IANA records are clear about that. But if the geolocation service cannot get any more granular than identifying 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 again use color to show quantity. Rather than just using a single hue (a fancy term for the color of the rainbow), let’s use a *diverging color scheme* (two opposite colors) 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 (as you can see in Figure 5.2), but it puts a nice slope and curve in a U.S. map and it’s good (and fun!) to play around with different projections.

# create a choropleth of the U.S. states

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

# we can cross the indexes of the vectors

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

# 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 Choropleth of U.S. States with ZeroAccess [FILENAME 793725c05f006]

We have another handsome but **relatively useless** map. We can easily see that California, Texas, Florida and New York are above average, but it’s also good to have the wherewithal to realize the 4 most populated states are 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 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?**”; or, pick a number and ask **“How many infections per 1,000 people?”** The differences between these questions are subtle, and in this case we will do the first method 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.

# 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 choropleth

**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.

type="activity"

In the “state-internets.csv” data, we also included the count of Internet users if you would like to try to create a choropleth normalized on estimated Internet users per state (it is a prettier picture).

Is this Weird?

We have to stop for a moment and look at what we have. We have a range of normalized values from 1 in 724 people with in infection in Wyoming to 1 in 1,550 people in Washington State. Does this mean that the citizens in Wyoming are much more careless than those in Washington? Perhaps more Washingtonians run Linux? Or—and this is an important concept—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 geolocation services. We have what we have, however, so 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 In chapter 8, but let’s use the default R graphics boxplot() function and whip out a quick boxplot (and save the results returned into a variable called popbox). There are multiple ways to create a boxplot, but the default function just accepts in a vector of values and works it’s magic. Usually, when we are just looking at the data for ourselves we will not set the title, but we wanted to 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, which are represented by individual points. There are clearly three points above the plot and two points below. While we could sort the data and look for the top 3 and bottom two, we saved off the popbox variable and we can lookup the values in the popbox$out (the outliers) vector in our original data.

**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

According to the method employed by Tukey in the boxplot, we could consider these five states as being odd (as mentioned, we will re-vist the boxplot in Chapter 7). Let’s explore another measure of determining oddballs 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 by showing how many *standard deviations* from the *mean* an observation is. A z-score is most often used to compare distributions from completely different scales, a method sometimes label “standardizing” the data. 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. If your eyes started to glaze over from the z-score description don’t worry, every time we calculate this, we have to look up how it’s done. 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 familiar bell curve which is also *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. One thing to note, this method doesn’t work well if the data is skewed, so we should probably check a quick histogram (with the hist() function) to be sure we are mostly symmetrical.

When using this approach, generally anything above (or below) 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)**

# we can inspect the “z” variable for the specific z-scores

# 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

It appears those same 5 entries fall within 3 standard deviations. With that knowledge in hand it calls into question the use of population in our normalization process (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”. Rather than focusing on solving things at the state level, we could bring this data down to the county level within the states. This will supply more data points and allow a finer separation of the population, which will open up more possibility.

type="concept"

What’s the p-value?

Trying to identify *weird* versus *normal* is a core concept within statistics and depending on the circumstance there is usually more than one way to measure it. At the heart of “statistically significant” is knowing if something is weird or just the result of natural variations. One very common and widespread approach has been given the abbreviated name of “p-value”, but don’t mistake its widespread use with a widespread understanding or even consistent use. The p-value has a very specific (and difficult to remember) meaning. In order to define and calculate a p-value, we begin with a statement (technically called a *null hypothesis*) and calculate the probability of our data being generated by chance if the statement is true - this is the p-value. Now the subtlety of the p-value is often lost and people jump to convenient (and wrong) definitions like it’s the probability of the statement being true (it’s not). To complicate this even more, somehow it became generally accepted that a p-value of 0.05 (1/20th) or less was “statistically significant”, creating what is essentially an arbitrary cut-off point. We will be revisiting the value of p-values when we talk about regression analysis. Just tuck the term “p-value” away in your memory bank as a measure of significance or in our case, “weirdness”.

Counting In Counties

It is difficult to generalize at the state level because, well, it is very is a very generalized population. We would be obscuring a wide range of diversity among people behind a single label of our descriptive statistics. We would be hard-pressed to calculate the influence of something like income or—more importantly—alien visits on ZeroAccess infections at the state level. We can get more granular by repeating this process again but at the county level. There are a few additional items to consider as we get into more detailed breakdown of geolocation of IP addresses. Most of the popular IP geolocation 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. Does that mean we should be very wary of this data? In order to answer that, we should pause and discuss a statistical concept: *natural variations will cancel out more often then stack up*, especially as we get more data (and over 3,000 U.S. counties represent more data).

Does variation stack or cancel?

Within statistics, we know that natural variations will generally cancel each other out, but this is counter-intuitive to fields in engineering (like computer science), where we are taught that if we add components that all have a slight variation, the effect will compound itself and we should expect a wide range of results. What’s the difference? Which viewpoint is right? 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. Natural variation in the quality of materials and manufacturing process produces parts that range equally between 98 and 102 millimeters. Engineers are taught that if we stack up 100 of those parts, we could expect something equally likely between plus or minus 2 times the number of parts (100). Meaning, it is possible that all 100 parts will be 98 millimeters, or possible that all the parts will be 102 millimeters, so we should expect a wide range in the output. The more we stack, the wider the range of output.

In statistics, if we can assume that each part has an equal chance of being any length within the range (and we’ll want to validate that assumption in the real word), they will begin to cancel each other out. Thanks to our basic understanding of programming, we can easily model this and see how variation occurs across multiple parts. Let’s generate 100 parts and have them uniformly be “manufactured” between 98 and 102 millimeters, then let’s take the average (could also be sum or something other measurement, but mean works here). Our engineering brains say this will appear between 98 and 102, but let’s see:

#setting seed for reproducibility

**set.seed(1492)**

# run 100 times, getting random values between 98 and 102

**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)))**

# result is a vector of 10,000 sets

# show the min and max of these parts

**range(parts)**

99.57977 100.47559

What is up with this? Even with 10,000 random iterations, none of them get close to the ranges of 98 or 102. 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 function). As we add more parts within the range, we become more likely to cluster around the mean.

There are a couple of takeaways from this tangent. First, it’s really fun to geek out a bit and generate data to answer questions with “what-if” scenarios. Second, **we shouldn’t toss out less-than-perfect data**. If the variations are caused by natural or random variations we can assume the variation has more of a cancelling effect than a stacking effect. Now, this doesn’t mean we get to ignore variations like this, but instead it means that the variation will have less of an impact on throwing our analysis off then we think, but we will still want to account for this variation in our work.

Relating this back to our analyses, we have all sorts of items in our spatial data that may be throwing off our calculations. All of the geolocation lookups have a 25-mile radius of accuracy (so, some points that are supposed to be, say, in Southern Maine, actually end up in New Hampshire) and, several of our data points will be further off than that. But these may cancel if the variation is random (meaning things in New Hampshire could also just as easily end up in Maine). This doesn’t mean the data is worthless. Until we can learn some more advanced techniques, we can just take the error introduced as a grain a salt in our outcomes. In other words, we could use this data to estimate how much of an affect alien visits have, but we wouldn’t want to balance the fate of a company on analysis with this data without a lot more rigor and investigation.

Moving 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 (anything below country should account for it). But now rather than count things with table() and tossing into a data frame, we have to do some transformation on the returned names. 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 which returns a list object, so we convert it to a vector with the unlist() function. The result 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. And now we’re beginning to see how fun this data munging thing can be, right?

## 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")**

We now have a data frame with three columns, the state, county and count of infections and we will label the columns accordingly. There is a lingering “so what?” we need to answer before proceeding. Aside from the initial “*wow*” factor of generating a cool looking map, there is not much to learn from a raw count being displayed on a map. We may see some hot spots and we may be able to visually compare different areas on the map, but we can’t really learn much from this amount of detailed data on a map. Moving forward, we’ll switch from creating maps with this data to performing some *real* analysis to see if we can find some explanation for the infections.

We will want to pull in other data here (also split out by the county) just as we did at the state level with population. Then we may be able to start to understand a bit more about these malware infections. Perhaps there is some…*foreign*…(some would say, “*alien*”) data that would either help explain variations in the malware infections or help support the techniques we wanted to cover.

type="note"

A Quick Caveat

We will be applying some techniques here that should not be done as lightly as we are applying them. We are focusing more on walking through the concepts and techniques rather than attempting to perform insightful research here. Using statistical methods without fully understanding them is akin to an unlicensed teenager taking a car out for a spin.

We scoured the internet, pulled together a collection of rather interesting data points and have already done the data munging to produce the data we’ll use here (“county-data.csv” on the website). For the purpose of creating a tutorial, we’ve extracted a few statistics by county from various places and made it available on the book website.

* region and subregion are the state and county respectively
* pop is the estimated county population
* income is the median income for the county
* ufo2010 is the number of UFO sightings in the county during 2010 (as recorded on the national UFO reporting center: nuforc.org)
* ipaddr is the number of IP addresses that translate to the county (pulled from the open freegeoip.net package)

As luck would have it (for you), the data is in a perfect state so it can be read in and simply merged with the ZeroAccess county data we just created. There is one special note with the merge() function: 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 especially 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.data <- read.csv("data/county-data.csv", header=T)**

# notice the all.x option here

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

# replace all NA's with 0

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

Running summary(za.county) on our data, we can get a good feel for what things look like in there (and we learn that people who name counties have affinity for the founding fathers):

subregion region pop income

washington: 32 texas : 254 Min. : 71 Min. : 19344

jefferson : 26 georgia : 159 1st Qu.: 11215 1st Qu.: 37793

franklin : 25 kentucky: 120 Median : 26047 Median : 43332

jackson : 24 missouri: 115 Mean : 101009 Mean : 45075

lincoln : 24 kansas : 105 3rd Qu.: 67921 3rd Qu.: 50010

madison : 20 illinois: 102 Max. :9962789 Max. :120096

(Other) :2921 (Other) :2217

ipaddr ufo2010 za

Min. : 0 Min. : 0.000 Min. : 0.0

1st Qu.: 5367 1st Qu.: 0.000 1st Qu.: 6.0

Median : 15289 Median : 2.000 Median : 17.0

Mean : 387973 Mean : 7.943 Mean : 83.8

3rd Qu.: 62594 3rd Qu.: 6.000 3rd Qu.: 56.0

Max. :223441040 Max. :815.000 Max. :7692.0

Now that we’ve looked at the data, can you pick out the relationship with UFO visits? Not yet? How can we begin to pick apart the relationships in this data? Thanks to the work of statisticians, we have a technique known as *linear regression* that is extremely powerful and yet extremely dangerous.

Introducing (Briefly) Linear Regression

We are going to discuss a collection of techniques loosely called “*linear regression*”, but we should point out that college courses focus on nothing but linear regression for a semester and still don’t cover all aspects of it. And books like “Applied Linear Regression Models” are over 1300 pages long and packed with statistical notation (it’s a page-turner!). This is all to say that regression analysis is an incredibly rich and deep topic and we will barely scratch the surface here. What we hope to do here is take away some of the mystery around regression analysis and put the technique in context, while at the same time introduce enough warnings and common pitfalls that the reader doesn’t end up shooting themselves with this powerful of flexible technique.

Regression analysis is a workhorse and it is behind many of the scientific findings we may hear about. Titles that say, “*Scientists find a link between something and something else*” is almost always based on regression analysis. Researchers will use regression analysis for two general purposes. First, it can be used to estimate how **different** observable inputs contribute to an observable output. In our case we want to estimate how alien visits to U.S. counties (observable inputs) contribute to the rate of ZeroAcces infections in that county (observable output). With regression analysis, not only can we estimate how the significance (or lack thereof) of the contribution that each variable makes, we can also estimate how strong that contribution is. Don’t worry if it is a bit confusing now, we will cover this more as we get into our data. Regression analysis is a powerful tool to describe relationships between observations.

The second purpose for regression analysis is **prediction**. The output of regression analysis is a formula. Given specific inputs, we can make an estimate, or *predict* what the output will be. A classic example with this is the relationship between height and weight. It’s relatively intuitive that taller people weigh more, but if we add other observations such as male or female, age and so on, we can not only have an expected value, but we can establish an expected range of a persons weight. This is the method doctors use to tell patients they are above or below their expected weight, height, etc. Regression analysis is a powerful tool for estimation and comparisons of observed outputs.

To demonstrate this in practice we will use fictitious (and rather simple) data. We start with a single input variable, and we’ll generate random data points from a normal distribution (any distribution will work, the normal is just pretty). We’ll use the rnorm() command and create 200 points with a mean of 10 and a standard deviation of 1 (the default).

# for reproducability

**set.seed(1)**

# generate 200 random numbers around 10

**input <- rnorm(200, mean=10)**

**summary(input)**

Min. 1st Qu. Median Mean 3rd Qu. Max.

7.242 9.408 10.180 10.100 10.720 12.910

Looking at the summary, the result is data that ranges from 7.2 to 12.9. And now we want to generate the output data. We want to create a linear relationship between the input and our output, so we will pass the mean in as double the input variables. By using rnorm() we are introducing random variations, but centering on the mean we are creating that linear relationship we can model. We then create a data frame out of both the input and output for easy handling and plotting.

# generate output around a mean of 2 x input

**output <- rnorm(200, mean=input\*2)**

# put into data frame to plot it

**our.data <- data.frame(input, output)**

Now we can pass all of this into ggplot and create a scatter plot. But we will add something special by including the geom\_smooth() function and we will tell to use a linear model (“lm()”). This will overlay a single straight line that best describes our example data.

**gg <- ggplot(our.data, aes(input, output))**

**gg <- gg + geom\_point()**

**gg <- gg + ggtitle("A Sample Linear Relationship")**

**gg <- gg + geom\_smooth(method = "lm", se=F, color="red")**

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

**print(gg)**

Figure 5.9 Sample data with regression line [FILENAME 793725c05f009]

We can see that the data isn’t exactly nice and neat (this is rnorm() introducing some random variation), but there is a definite trend. As the input variable increases, the output variable also increases, and the data flows from the lower left to the upper right. It sure looks like there is a relationship from this data (of course), but it’s difficult to describe it beyond simple descriptions… enter regression analysis.

In order to run a linear regression on our data, it is one very simple command:

**model <- lm(output ~ input, data=our.data)**

Congratulations! You have just run your first linear regression. Let’s look at the output with the summary() function and talk through the output.

**summary(model)**

Call:

lm(formula = output ~ input, data = our.data)

Residuals:

Min 1Q Median 3Q Max

-2.93275 -0.54273 -0.02523 0.66833 2.58615

Coefficients:

Estimate Std. Error t value Pr(>|t|)

(Intercept) 0.27224 0.77896 0.349 0.727

input 1.97692 0.07729 25.577 <2e-16 \*\*\*

---

Signif. codes: 0 ‘\*\*\*’ 0.001 ‘\*\*’ 0.01 ‘\*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 1.013 on 198 degrees of freedom

Multiple R-squared: 0.7677, Adjusted R-squared: 0.7665

F-statistic: 654.2 on 1 and 198 DF, p-value: < 2.2e-16

There are many, many things to look at here. It starts with the function we used and a summary of the *residuals*. The residuals are the difference between what the model predicts and what we observed in the output. The line is specifically calculated so the mean of the residuals is zero (making it the “best fit” for the data). Often times, we’ll skip over the residuals (as there are better methods for interpreting the residuals).

The next section talks about the coefficients. In this model we have two: the *intercept*, which is always present, and our *input variable*. If we had more observed inputs, they would be listed here, one per line. The first column here is the estimated value for the coefficient. For most linear models, the intercept has little to no meaning. The intercept coefficient signifies if our input is at zero, we could estimate the output to be around 0.27 (which doesn’t make sense if we were talking about a person’s height). We didn’t set this when we created this data (which made it zero) so 0.27 is pretty close.

Looking at our coefficients here we can construct the model:

output = 0.27224 + 1.97692(input)

This model is what we would use to estimate new output values given an observed input. But remember we generated our data by multiplying the input by 2? The linear model here thinks we multiplied by 1.97692, which is pretty close. This coefficient for the input variable (or variables) is where can begin to see the power of regression analysis. We can interpret like this:

If all the other input variables are held constant, a one-unit change in this input variable is associated to an average change of 1.97 in the output.

Since we only have one input variable we have nothing else to hold constant. Even if we have dozens of variables, we can isolate the effect of the individual variables with regression analysis.

The next column in the coefficients represents the standard error. We can use this along with the estimated coefficient to generate a confidence interval for the coefficient by passing in the output of the lm() command to the confint() function:

**confint(model)**

2.5 % 97.5 %

(Intercept) -1.263895 1.808368

input 1.824502 2.129343

The output tells us that with 95% confidence, the input coefficient is really between 1.82 and 2.13 (and our value of 2 is well within that range).

The next two columns are measurements of how much the variable contributes to the model. The last column here is called the “*p-value*” (we mentioned it earlier in a call-out). As a general rule, smaller p-values contribute more significantly to the overall model and larger p-values means the relationship between this input variable and the output is more likely to be chance. If we have a high p-value, we may want to look for other explanatory variables and remove any variables with a high p-value. Most people settle on 0.05 as the threshold for significance. Meaning if the p-value is less than 0.05 (and ours is well beneath it) then the variable is significant and can stay in the model, above 0.05 though and we should consider tossing it to the curb.

We should be aware that a p-value is a *continuous* variable **and** a *single measurement*. The accepted significance value of 0.05 is a relatively arbitrary stake in the ground. If one variable has a p-value of 0.049 and another of 0.051, those are very, very similar and yet according to conventional practices, one is considered significant and the other should be tossed out. If you’re trying to publish in some hoity-toity journal, than you’ll have to do that. But if you’re trying to learn from the data, take that 0.05 threshold with a grain of salt.

There are two other things to look at in the output of our linear regression. Look at the “*Adjusted R-squared*” value in the second to last line. The adjusted R2 (or technically the “adjusted coefficient of determination”) signifies the amount of variation explained by our model. Values ranges from 0, meaning the model is no better than using the output mean, and up to 1, meaning the model describes the output perfectly. In our model it was calculated as 0.76, which means the linear model we generated can reasonably explain 76% of the variation in the output data. There is no magic number we want the R2 to be because it’s relative. If we are starting from a place where we are simply guessing at the output, than an R2 of 0.05 is helpful. But if we have an existing model at 0.76, then 0.05 is a large step backwards. **When people want a quick understanding of a model, they focus on the R2 value**.

The last thing to put an eye towards is the p-value on the bottom line. This is the p-value of the entire model. At this point, we probably have a good feel if this is a good model or not, but keep an eye on this p-value. In our model, the p-value is tiny, so we can continue to feel good about this model.

Understanding Common Pitfalls in Regression Analysis

We hesitated even discussing regression analysis in this book. There are so many ways things can go wrong and so many ways to screw up, not to mention all the assumptions within the process that must be kept in check. However, we did include it and we must also include some of the common pitfalls here.

We cannot extrapolate beyond our data

Our data represents the entire range of our knowledge. We can verify that there is a linear relationship in the data we have, but we cannot extend that belief above or below our input values. As an example, let’s say we’ve developed a horrendously over-simplified model to estimate the cost of a data breach (output) from just a count of records lost (input). If we only look at breaches that have lost 1,000 to 100,000 records, we cannot extend this to breaches with more than 100,000 or less than 1,000 records lost. We have no confidence that the relationship holds beyond the data we have. Though if we did develop such a blatantly ridiculous model, we’d be sure to discuss the small R2 value so people may have a fair shot at dismissing such a simple model.[[1]](#footnote-1)

Outliers have a lot of influence

Before regression analysis is performed, it’s worthwhile to validate the data and identify any outliers that are the result of mistakes or errors. Outliers will have a large influence in the output of the model and will greatly influence the model selection. This doesn’t mean that we should go through and remove all of the oddball observations (even though this was a common practice many years ago). For every observation that appears to be an outlier, it is just good practice to verify its validity before continuing on. Sometimes outliers are valid and we must include them and account for them in the model. Other times, it may just be a result of mistyping or recording something in different unit of measurement and those should be fixed or removed.

Hidden Relationships Hide Well

It’s easy to gather a whole bunch of variables and toss them into a linear regression and have many of them turn out to be significant. But we have to approach this with some element of common sense. It’s standard practice to keep the number of variables to a minimum (see the next pitfall). But internal relationships in the data can be misleading and we want to be careful of something called multicollinearity. If we have two or more input variables that are highly correlated to each other we may be incorrectly assigning meaning where none exits. We will see an example of this when we get back to our ZeroAccess data.

Too Many Variables

As stated, if you gather enough variables and toss them into regression analysis, it is inevitable that something in there will be significantly correlated. This actually applies to many things beyond regression analysis. When dealing with large quantities of variables and data, different considerations need to be made. If we use the de facto guideline of 0.05 significance testing (remember the p-value discussion earlier?) then—just by chance—1 in 20 comparisons may appear significant. Multiply that across hundreds of variables and we are setting ourselves to be misled by spurious connections. It’s considered good practice to minimize the number of variables in the model. If a variable doesn’t increase the R2 value much then we’d want to keep it out of the model, even if it has a tiny p-value.

Visualize and Apply the Sniff Test

It’s a good idea to visually inspect the data before jumping into regression analysis. In our example above we created a simple scatterplot and added in the regression line. This gets a little more complicated as we add multiple variables, but it’s a good habit to get into. But even beyond that, we want to apply a healthy dose of logic to the variables and make sure that they have at least some reason to be included. This will help reduce the overall number of variables and hopefully help the analyst get to know the data if they didn’t before.

Apply linear regression to linear relationships

The concept of linear regression applies to the coefficients, not our input variables. Our sample model ended up being:

output = 0.27 + 1.97(input)

However, if the relationship was exponential, we may have created a model on our transformed input variable by squaring it for example, and we would still technically have a linear model.

y = 0.27 + 1.97x2

While the model itself does not represent a linear relationship, we can still use a linear regression model since the coefficient is linear. This means, if the relationship is not linear then we can transform one ore more variables (input or output) by taking the log, square root or adding exponentials to our variables. There are some rules for doing transformations and we also begin to run the risk of over fitting a model. Over-fitting occurs when a model is tuned to the data we have, but then performs much differently when new data is added (or we use to estimate new values, etc.). Given enough transformations, we can create a model where R2 is 1 and all of the residuals are 0, but add new values and the R2 drops like a brick. We want our models to strike that balance between simple and accurate.

Regression on ZeroAccess Infections

If you made it through the previous section, you should have a basic understand of a regression model and some of ways we can screw up with them. Now we can start to pull more meaning from our spatial data than maps would allow us to do.

Let’s do a simple regression on our real data and see how well “visits from aliens,” describes ZeroAccess infections. While this may be silly to non-believers, we should be open to the possibility as researchers. We couldn’t find any hard data related to alien visits, but the good folks at the National UFO Reporting Center have collected sightings of the visitors. We will be using that data as a proxy and it’s in the ufo2010 variable in the prepared data. In order to run the linear regression, we again call the built-in lm() function and specify the output variable (za in our za.county data frame) then the tilde character followed by variable with alien sightings. If we wanted to add more variables we would add them—literally—with the plus (+) symbol. If we wrap the whole command in the summary() call, we get the output immediately. We are trimming the output here to the relevant information.

**summary(lm(za ~ ufo2010, data=za.county))**

Coefficients:

Estimate Std. Error t value Pr(>|t|)

(Intercept) 17.72429 2.59051 6.842 9.39e-12 \*\*\*

ufo2010 8.31867 0.08685 95.784 < 2e-16 \*\*\*

---

Signif. codes: 0 ‘\*\*\*’ 0.001 ‘\*\*’ 0.01 ‘\*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 138.4 on 3070 degrees of freedom

Multiple R-squared: 0.7493, Adjusted R-squared: 0.7492

F-statistic: 9175 on 1 and 3070 DF, p-value: < 2.2e-16

Using our new skills, we can see the p-value of the UFO variable is really tiny at “< 2e-16”, indicating the connection is significant and the R2 value is 0.74, that’s quite impressive. The coefficient on UFO sightings, (8.31867) tells us that for every UFO sighting we should expect about 8 more ZeroAccess infections. This is an incredibly strong model and there is enough to here to submit to a hoity-toity peer-reviewed journal explaining how we have scientifically proven UFO’s are causing the spread of ZeroAccess malware! We can see the headlines already:

Researchers Link ZeroAccess Infections to Alien Visitors

Before we get ahead of ourselves, maybe we should looks at some of these other variables. Let’s run another regression with all of these variables and see what happens.

**summary(lm(za ~ pop + income + ipaddr + ufo2010, data=za.county))**

Coefficients:

Estimate Std. Error t value Pr(>|t|)

(Intercept) 1.380e+01 4.864e+00 2.838 0.00457 \*\*

pop 7.881e-04 7.960e-06 99.004 < 2e-16 \*\*\*

income -3.152e-04 1.066e-04 -2.957 0.00313 \*\*

ipaddr 2.086e-06 2.656e-07 7.852 5.63e-15 \*\*\*

ufo2010 4.774e-01 8.724e-02 5.472 4.80e-08 \*\*\*

---

Signif. codes: 0 ‘\*\*\*’ 0.001 ‘\*\*’ 0.01 ‘\*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 65.72 on 3067 degrees of freedom

Multiple R-squared: 0.9435, Adjusted R-squared: 0.9434

F-statistic: 1.281e+04 on 4 and 3067 DF, p-value: < 2.2e-16

Scanning down the p-values, it looks like all of these are significant, but we have a slightly larger sample size here with over 3,000 counties in our data, so we should reduce the significance below 0.05. With that in mind, it looks like income may be suspect (meaning the relationship between income and infection rate is small or non-existent), but the number of IP address and UFO visits still appear strong. Notice that as we’ve added more variables the influence of UFO visits has dropped and is accounted for in other variables now. While we have all of these variables in this model, we should check for something called multicollinear variables. This is where two or more of the input variables are correlated and that relationship is masking the [in]significance of a variable. We check for this by looking at something called the variance inflation. R has a nice vif() function in the Companion to Applied Regression (car) package. As a general rule, if the square root of the variance inflation is greater than 2 (something I have to lookup every time I do this), the variables are correlated and we shouldn’t trust that both are significantly contributing to the model.

**library(car)** # for the vif() function

**model <- lm(za ~ pop + income + ipaddr + ufo2010, data=za.county)**

**sqrt(vif(model))**

pop income ipaddr ufo2010

2.165458 1.038467 1.046051 2.115512

We can see that the population and ufo2010 are collinear. Oh no! Is it possible that UFO sightings are just a function of population? In order to test that, we can normalize out the population. For this we’ll just divide both values by the population, making them infections and sighting per capita, and rerun the single regression.

**za.county$za.by.pop <- za.county$za/za.county$pop**

**za.county$ufo.by.pop <- za.county$ufo2010/za.county$pop**

**summary(lm(za.by.pop ~ ufo.by.pop, data=za.county))**

Coefficients:

Estimate Std. Error t value Pr(>|t|)

(Intercept) 7.052e-04 1.213e-05 58.138 < 2e-16 \*\*\*

ufo.by.pop 2.690e-01 6.954e-02 3.868 0.000112 \*\*\*

---

Signif. codes: 0 ‘\*\*\*’ 0.001 ‘\*\*’ 0.01 ‘\*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 0.0005792 on 3070 degrees of freedom

Multiple R-squared: 0.004851, Adjusted R-squared: 0.004527

F-statistic: 14.96 on 1 and 3070 DF, p-value: 0.0001118

Great! The p-value is still under 0.05! But…oh…wait a second, the R2 value is telling us that this model is quite useless as it describes 0.4% of our data. At this point, it might be safe to listen to that little voice of logic and conclude that UFO visits and ZeroAccess infections are not related (so much for the grant money).

Let’s run one more analysis, but keep in mind, all of this data is available for download from the book website along with the code in this chapter. There is plenty of room for exploration here.

What is correlated to ZeroAccess Infections?

Let’s say we have a strong suspicion (or we’ve already run through a variety of models) that population of a county is the best overall *predictor* of how many infections pop-up in that county.

**summary(lm(za ~ pop, data=za.county))**

Coefficients:

Estimate Std. Error t value Pr(>|t|)

(Intercept) -1.678e-01 1.261e+00 -0.133 0.894

pop 8.313e-04 3.731e-06 222.791 <2e-16 \*\*\*

---

Signif. codes: 0 ‘\*\*\*’ 0.001 ‘\*\*’ 0.01 ‘\*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 66.71 on 3070 degrees of freedom

Multiple R-squared: 0.9418, Adjusted R-squared: 0.9417

F-statistic: 4.964e+04 on 1 and 3070 DF, p-value: < 2.2e-16

With an R2 value of 0.94, we’re going to be hard pressed to add more variables in here that mean much. Sure enough, when we cycle through the other variables we find that income and the number of IP addresses in that county do not add much to the overall model. What we can draw from the output of the regression on population is in the coefficient of 8.313e-04, which is engineering notation for 0.0008313, if we invert that (1/0.0008313) we can calculate that for about every 1,200 people a county has, we could expect one more infection of ZeroAccess.

Now let’s go back to maps and see if we can’t visualize what this looks like at the county level. Let’s generate a choropleth map at the county level for both the number of infections and the population. If the regression analysis is accurate we should a very clear relationship between the two.

Figure 5.10 Visual Relationship Between ZeroAccess Infections and Population [FILENAME 793725c05f010]

We created quite a few maps in this chapter, both with points and choropleths. While we can pick out variations across the map quite rapidly with the visual representation, we shouldn’t always just use visualizations with spatial data. Even though the maps showed variation, we showed through linear regression, that **population largely explains the variation**. That’s something to consider when creating maps (or any other visualization for that matter). We want to take a step back and ask the ever-popular question of “so what?!” If we can’t answer that, than maybe we don’t need the map at all and the analysis needs to go in a different direction.

1. Apologies for going overboard in this section, but Larry, we’re looking at you here. [↑](#footnote-ref-1)