Comp: CodeHeads font size seems to be inconsistent. Please fix accordingly.

Comp: Global all Listing Heads to be not italic.

Au: Please check chapter cross-refs (highlighted)

5

From Maps to Regression

"Even before you understand them, your brain is drawn to maps."

Ken Jennings, author and ***Jeopardy***! champ

OK

You have been learning some basics about security data and how to pull meaning from IP addresses. As briefly discussed in Chapter 4, IP addresses can be associated with geographic data if you look them up using a geolocation service. But what is the value in doing that? How much can you learn by associating a longitude and latitude with your data? The answer to that is dependent on what the IP represents and how deep you are willing to go. In order to describe the value of mapping the virtual world into the physical, this chapter begins 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-hour period during the month of July in 2013.

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

- How is ZeroAccess distributed across geographic areas and is there any significance to this distribution?
- What types of clients are 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, this chapter hones in on that last question. It is the most important and worthy of some serious research (anyone have some spare grant money?). But seriously, our purpose is to explore the benefits (and pitfalls!) you'll get from tying secondary data points, like alien visitors, to the primary data. Oftentimes, more can be learned through the combination and merging of related data, than just the original data in isolation. Therefore, as you bring the lessons from this chapter back to your own work, realize that insight may not just be in the primary data you collect, but in how it relates to other data you can collect from your environment. For example, we could ask the following questions that will combine two or more sources of data:

- Is there a relation between phishing victims and their HR data (education, pay grade, etc.)?
- Is there a relation between netflow (network) patterns and the software and services running on hosts?
- Is there a relationship between surfing habits and productivity or performance review scores of employees?

This is where you are heading in this chapter. You'll begin with one single data point (location data for systems infected with ZeroAccess) and explore the relationships within the data. Then you'll combine the location data with other geographic observations and apply a statistical technique known as ***linear regression*** to test the relationships of the various data points and look for significant (and perhaps even spurious) relationships. Prepare for the examples in this chapter by setting the directory to the working directory for this chapter and make sure the R libraries are installed (Listing 5-0).

not italic (consistency)

LISTING 5-0

```
# set working directory to chapter location
# (change for where you set up files in ch 2)
setwd("~/book/ch05")
```

```
# make sure the packages for this chapter
# are installed, install if necessary
pkg <- c("ggplot2", "scales", "maptools",
         "sp", "maps", "grid", "car" )
new.pkg <- pkg[!(pkg %in% installed.packages())]
if (length(new.pkg)) {
  install.packages(new.pkg)
}
```

Simplifying Maps

It's easy to get all wrapped up thinking that visualizing spatial data (maps) is 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 also quite fun. In order to take some of the mystique out of maps, we want to start by loading the latitude and longitude points from Symantec and treating them as x,y coordinates to create a simple scatterplot (Listing 5-1).

LISTING 5-1

```
# Load ggplot2 to create graphics
library(ggplot2)
# read the CSV with headers
za <- read.csv("data/zeroaccess.csv", header=T)

# 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()
# this may take a while, over 800,000 points plotted
print(gg)
```

Figure 5-1 looks remarkably like a world map without placing borders or loading any map specific tasks. This works with this data because there are 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) to be 1/40th of a full color. With the alpha at 1/40th, it will take 40 points on top of one another to create a non-transparent color. With 20 points, for example on top of one another, you see 50% transparency.

delete: th x2 (it is a fraction)

comma x2 / percent

From this basic scatterplot, you can see the density in the Eastern half and West coast of the United States and most of Europe is covered. You 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, you can only make guesses as to what's going on with what looks like a significant difference in Asian countries.

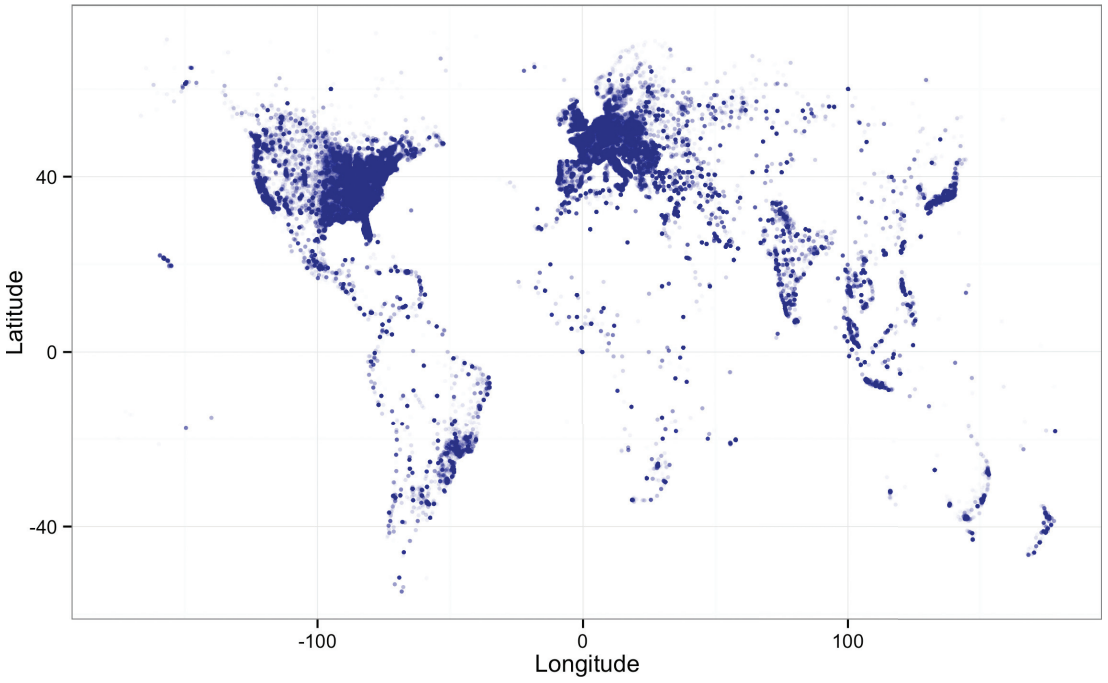

FIGURE 5-1 *Basic scatterplot using latitude and longitude*

There is something unique about maps, because you need to "project" the three-dimensional spherical world onto a two-dimensional flat canvas. When you do that, aspects of the map are distorted—shapes will be distorted, land areas will shrink or be over-represented, or distances will be skewed. But for most applications within information security, you 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.

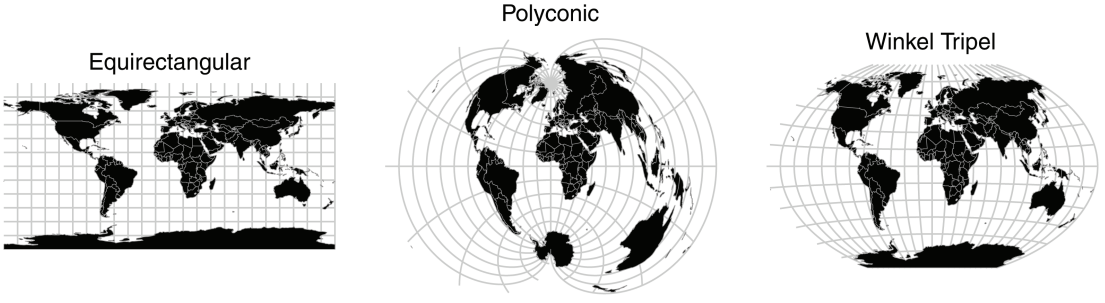

FIGURE 5-2 *Map projections*

If you take another look at Figure 5-1, it's a little hard to know where all those points land unless you 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 of the landmasses, 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. The `ggplot2` package has a function called `map_data()` that wraps the `maps` package to return a `ggplot2`-compatible data frame.

Calling `map_data()` with one character string of "`world`" will load just over 25 thousand rows of map data into a data frame, which means, as you've seen in Chapter 3, you can explore any and all of this data with commands like `str()`, `head()`, and `summary()`. You 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 you'll see later). To create the final map (see Listing 5-2), you call `coord_map()` to create the map projections (you will use the Mercator projection for this example), and you'll use a simple black and white theme on it with the `theme_bw()` function. Once you have the countries traced, you then add the points from the ZeroAccess data on the map as if you are creating a scatterplot like you did before (see Figure 5-3).

OK

comma

comma

LISTING 5-2

```
# requires package : ggplot2
# requires object: za (5-1)
# the "maps" and "mapproj" packages are used by ggplot2
# 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)
# remove text, axes ticks, grid lines and do gray border on white
gg <- gg + theme(text=element_blank(),
                 axis.ticks=element_blank(),
                 panel.grid=element_blank(),
                 panel.background=element_rect(color="gray50",
                                               fill="white"))
print(gg)
```

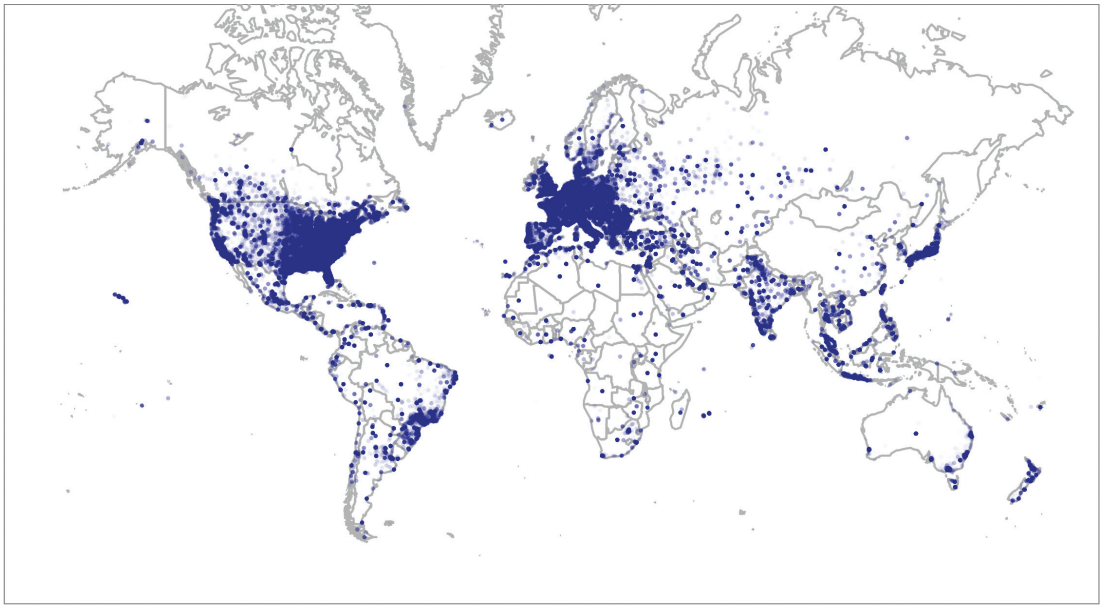

FIGURE 5-3 *Worldwide ZeroAccess infections*

Now, ***that's*** a real map; but, what can you learn from it? The answer is not much. This map communicates very little beside the fact that the ZeroAccess botnet is an international traveler, and that should surprise nobody. It's time to probe a bit deeper and see whether you can use maps to help you 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. You can't expect anyone to look at a map like this and extract the proportion of bot infections in countries. It looks like the United States and Europe are blanketed in bots; so, let's try a different type of map. You need to count how many infections you have in each country and then you can visualize the results with a ***choropleth***. A choropleth is a map in which the country is shaded or filled with color that is then associated with the data. For your first choropleth, you will have to figure out which country the latitude/longitude points are in and then you will use a single continuous color scale to represent that quantity (see Listing 5-3). To convert latitude and longitude to a country, you will adapt a function from Ryan Weald and call the function `latlong2map()`. That function will accept a data frame of longitude and latitude pairs along with the name of a map to translate onto.

LISTING 5-3

```
# require packages: maps, maptools
# packages are not required to create function
```

```
# but it cannot be executed without these loaded
library(maps)
library(maptools)
# 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]
}
```

Move these two Over to be aligned at end of arrow

Au: One line of code x3. OK as broken?

Should be aligned under and with "map2…"

Should be aligned under and with "Spatial"

The function returns a vector of names (country names in this case), and you can count how many times the country appears with the `table()` command. Next in Listing 5-4, you'll want to `merge()` the count of countries with the map data and reorder it for the plotting. By merging the data directly into the map data, you can associate the shading of the country with an attribute in the data, specifically the count of infections in that country. You will use the `scale_fill_gradient2()` function within `ggplot2` to get the color gradient associated with the quantity of infections. See the result in Figure 5-4.

LISTING 5-4

```
# requires packages: ggplot2, maps and maptools
# requires objects: za (5-1), world (5-2), latlong2map (5-3)
# 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)
```

(continues)

Listing 5-4 (*continued*)

```
# 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),
                                name="Infections")
# remove text, axes ticks, grid lines and do gray border on white
gg <- gg + theme(axis.title=element_blank(),
                 axis.text=element_blank(),
                 axis.ticks=element_blank(),
                 panel.grid=element_blank(),
                 panel.background=element_rect(color="gray50",
                                               fill="white"))
print(gg)
```

FIGURE 5-4 *Choropleth of ZeroAccess infections*

Voila! You have a rather good-looking (some might say, "spiffy") map and it looks like the United States has cornered the market on ZeroAccess infections. There would be no way you could learn that from the points in Figure 5-3. However, it's very difficult to tell the specific quantity by color density (Chapter 6 discusses visualization techniques in more detail); all you can tell from this type of map is that the United States has more infections. To learn just how much more, you can look into the `wct` variable and then calculate the proportion of infections in the United States (Listing 5-5).

OK

LISTING 5-5

```
# requires object: wct (5-4)
head(wct)
##        region count
## 1 Afghanistan    53
## 2     Albania  1166
## 3     Algeria  3014
## 4     Andorra     4
## 5      Angola   160
## 6   Argentina  6016

# for each wct$count, divide by sum, gives us proportion of the whole
perc <- wct$count/sum(wct$count)
# covert to a readable format, round it and create percent
wct$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)
##      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
```

You could have just created this table in the beginning to answer the question, "How is ZeroAccess distributed across geographic areas?" The map visually highlights the gap between the United States and the rest of the world. Also keep in mind that these are just total counts and not normalized for population or anything. So at this point, the 35 percent represents a proportion solely within the ZeroAccess data, and you should not infer more without further analysis.

Changing the Scope of Your Data

You can't lose sight of the goal here, which is to find a way to correlate Zero Access infections with other data points like alien visits. To get closer to answering this, you can simplify the data set to the U.S. infections. We chose to do this not just because working with over 800,000 data points can be a bit slow on some systems, but also because it will be much easier to focus in on the United States because of our knowledge of the geography and accessibility of data (especially around alien visitors).

However, as you change the scope within the data like this, you need to consider how this may change the question you're able to answer. You can no longer generalize about every infection everywhere because you cannot readily transfer what you learn from infections in the United States to other countries and/or cultures. Another way to state this is that you cannot be sure that the factors that contribute to infections in the United States will match the factors elsewhere. Those considerations go beyond and outside the data

you are looking at now, so be sure to avoid making any broad assumptions as you continue your analysis and present your results.

If you 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 will trace out only the U.S. map. You need to reduce the data size and remove data that are not in the United States. You can reuse the `latlong2map()` function, this time mapping the points to the United States by specifying "state" as the second argument. 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 the data.

Once all that processing is done, you can make a nice map of the continental United States showing all the ZeroAccess infections in the country (Figure 5-5). Notice that for this plot and the last few you have been removing all the extra ***chart junk*** (a term coined by Edward Tufte) on the map. This is done with the `theme()` function at the end and removing graphical features by assigning them to `element_blank()`. See Listing 5-6.

chartjunk (style sheet)

LISTING 5-6

```
# requires package: ggplot2, maps, maptools
# requires objects: za (5-1), latlong2map (5-3)
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)), ]
# 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)
# stripping off the "chart junk"
gg <- gg + theme(axis.title=element_blank(),
                 axis.text=element_blank(),
                 axis.ticks=element_blank(),
                 panel.grid=element_blank(),
                 panel.background=element_blank())
print(gg)
```

Consider Figure 5-5 for a moment. Does it look…***strange***? This is where you really have to be careful because after working with spatial data, we can tell you ***this looks like a reflection of population density*** and ***not*** infections. Therefore, after reviewing Figure 5-5, you might find yourself asking a slightly different question: ***Could ZeroAccess infections just be a reflection of the population***? You ***could*** stop and apply a statistical technique called regression analysis (you will later), but for now stick with pictures and create another choropleth. This time you'll break up the data and perform counts based on the U.S. states.

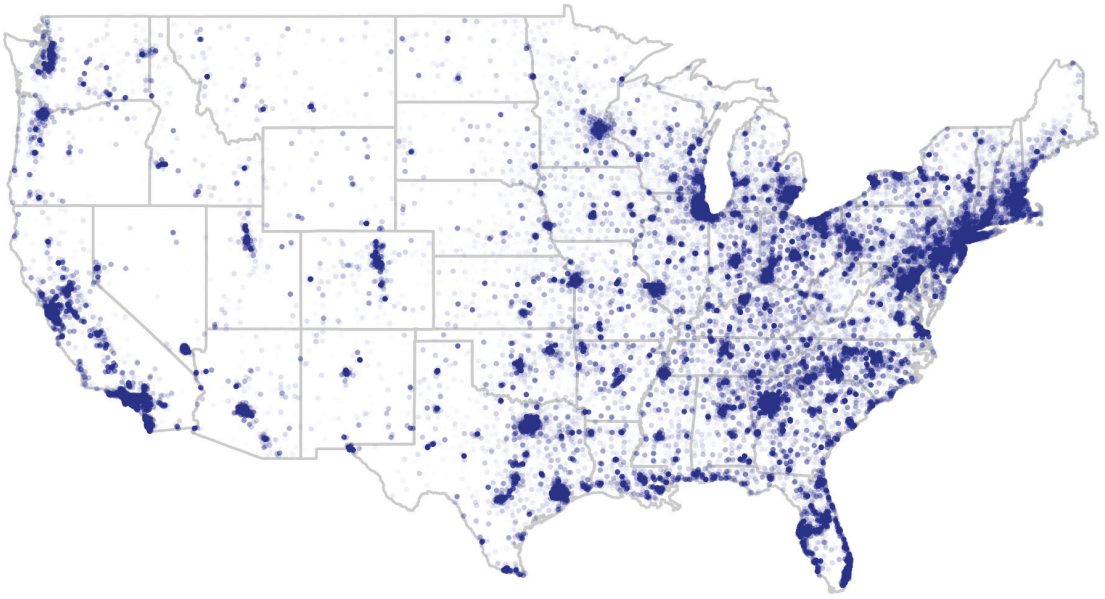

FIGURE 5-5 *ZeroAccess infections in the United States*

The Potwin Effect

As you dig deeper than just country, you 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 you examine the data, you'll see that there are 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 United States, the geographic center is just outside of Potwin, Kansas. For this purpose, they are "unknown U.S. locations" and not really in Kansas, so you are going to remove these data points from the next bit of code to avoid unfairly assigning infections to Kansas.

In this map, you want to again use color to show quantity. Rather than just using a single hue (a fancy term for color), you'll use a ***diverging color scheme*** (two opposite colors) and assign the mid-point of the range to the mean count per state (see Listing 5-7). This will allow you to show states with above average infection counts with one hue and the below average states 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. It's good (and dare we say fun!) to play around with different projections.

LISTING 5-7

```
# requires package: ggplot2, maps, maptools
# requires objects: za (5-1), latlong2map (5-3)
# 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 <- c("#A6611A", "#DFC27D", "#F5F5F5", "#80CDC1", "#018571")
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),
                                name="Infections")
gg <- gg + theme(axis.title=element_blank(),
                 axis.text=element_blank(),
                 axis.ticks=element_blank(),
                 panel.grid=element_blank(),
                 panel.background=element_blank())
print(gg)
```

Figure 5-6 shows another handsome but ***relatively useless*** map. You can easily see that California, Texas, Florida, and New York are above average, but it's also good to have the wherewithal to realize that the four most populated states are California, Texas, New York, and Florida, in that order.

In other words, you are just seeing a reflection of population in this map, so you have to ***normalize*** this data to the population. In order to normalize you can take multiple approaches. The simplest ways to normalize involve answering one of these questions:

- How many people per one infection?
- What proportion of the people are infected?
- How many infections per 1,000 people?

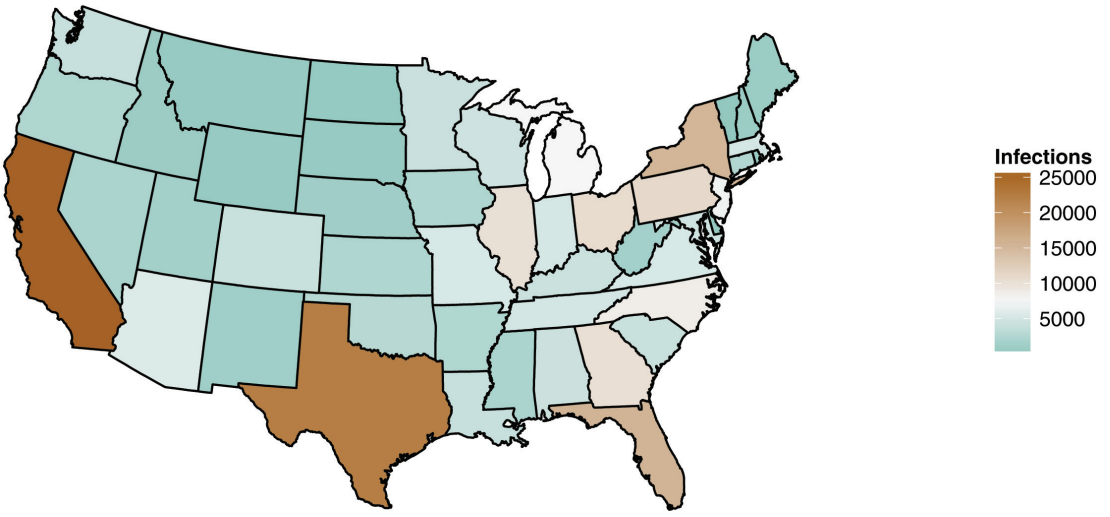

FIGURE 5-6 *Choropleth of U.S. states with ZeroAccess*

The differences between these questions are subtle, and in this case you will do the first method because you will get whole numbers and it will be a little easier to conceptualize for your reader. In order to determine the number of people per infection, you divide the population in a state by the number of infections in that state (Listing 5-8). In this case, we have already scraped population data from `http://www.internetworldstats.com/stats26.htm` and made it available in an easy format on the book's website (`state-internets.csv` in the Chapter 5 download materials at `www.wiley.com/go/datadrivensecurity`).

comma

rebreak

LISTING 5-8

```
# requires package: ggplot2, maps, maptools
# requires objects: sct (5-7), colors (5-7), latlong2map (5-3)
# 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)
```

(continues)

Listing 5-8 (*continued*)

```
# 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),
                                name="People per\nInfection")
gg <- gg + theme(axis.title=element_blank(),
                 axis.text=element_blank(),
                 axis.ticks=element_blank(),
                 panel.grid=element_blank(),
                 panel.background=element_blank())
print(gg)
```

Remember California, Texas, Florida, and New York having the highest infection counts? Using the `za.norm` data generated in the Listing 5-8, you can view the exact counts. When you normalize to population, California and New York drop to below average with one infection per 1,440 and 1,287 people on average, respectively (see Figure 5-7). Wyoming now sticks out as the most infected state since one in 724 people in Wyoming appear to have ZeroAccess infections.

delete: the

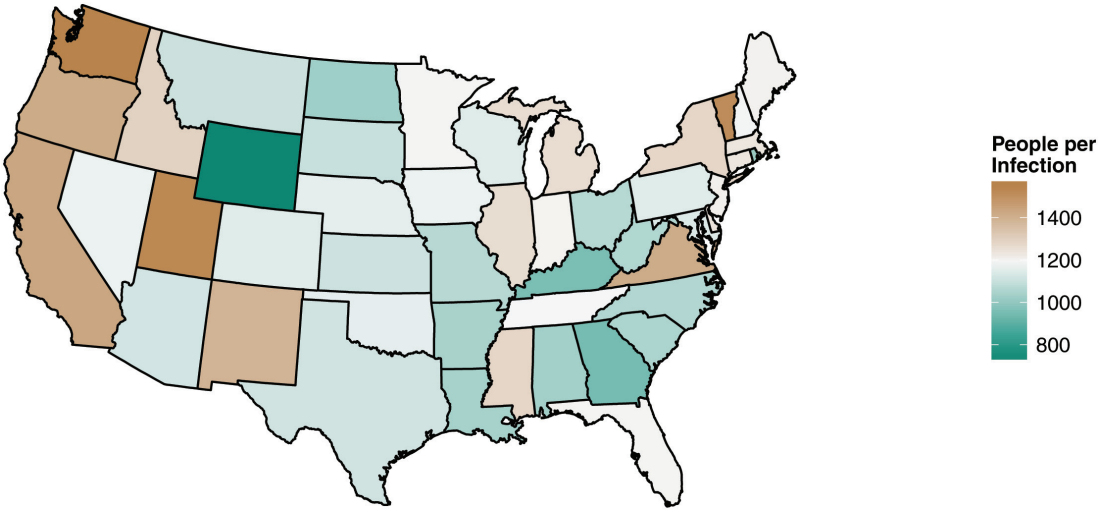

FIGURE 5-7 *Normalized ZeroAccess infections: Number of people in the state per one infection*

> ***Note***
>
> *In the `state-internets.csv` data, we also included the count of Internet users if you want to try to create a choropleth normalized on estimated Internet users per state (it is a prettier picture).*

monofont not italic

Is This Weird?

Let's stop for a moment and look at the current results. You have a range of normalized values from 1 in 724 people with an 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—is the range of observations simply from ***natural variation in the 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? You need to understand if the extreme values are ***outliers*** or if they are within expectations. There are two key methods to test for outliers:

rebreak

- Using a boxplot (the "IQR" method)
- Calculating a z-score

OK

Using a Boxplot to Find Outliers

The boxplot was develop by John Tukey (you met him briefly in Chapter 1) and was designed to show a distribution of values visually. It does this by plotting a box from the 25th percentile to the 75th percentile in the distribution. This distance is called the ***interquartile range*** (IQR). Then lines are extended from the box for a distance one and half times the length of the IQR. Anything beyond the length of these lines is a good candidate to be labeled as an outlier, and is represented by point. The further these points are the more likely they an outlier. In order to create a boxplot, you will use the default R graphics `boxplot()` function. You'll save the results returned into a variable called `popbox` (see Listing 5-9) for exploring in Listing 5-10. While there are multiple ways to create a boxplot, the default function just accepts in a vector of values for the distribution and then it works its magic (see Figure 5-8).

developed

hyphen (style sheet)

comma

comma

LISTING 5-9

```
# requires objects: za.norm (5-8)
# create a box plot of the count
popbox <- boxplot(za.norm$count)
```

Looks like you may have a few outliers, which are represented by individual points. There are clearly three points above the plot and two points below. Although you could sort the data in `za.norm` and look for the top three and bottom two, you saved the output from `boxplot()`, which has various data points about the boxplot, into the `popbox` variable, so you can look up the values in the `popbox$out` (the outliers) vector in the original data (Listing 5-10).

Not Bold and Not Italic on $out, treat it as one word.

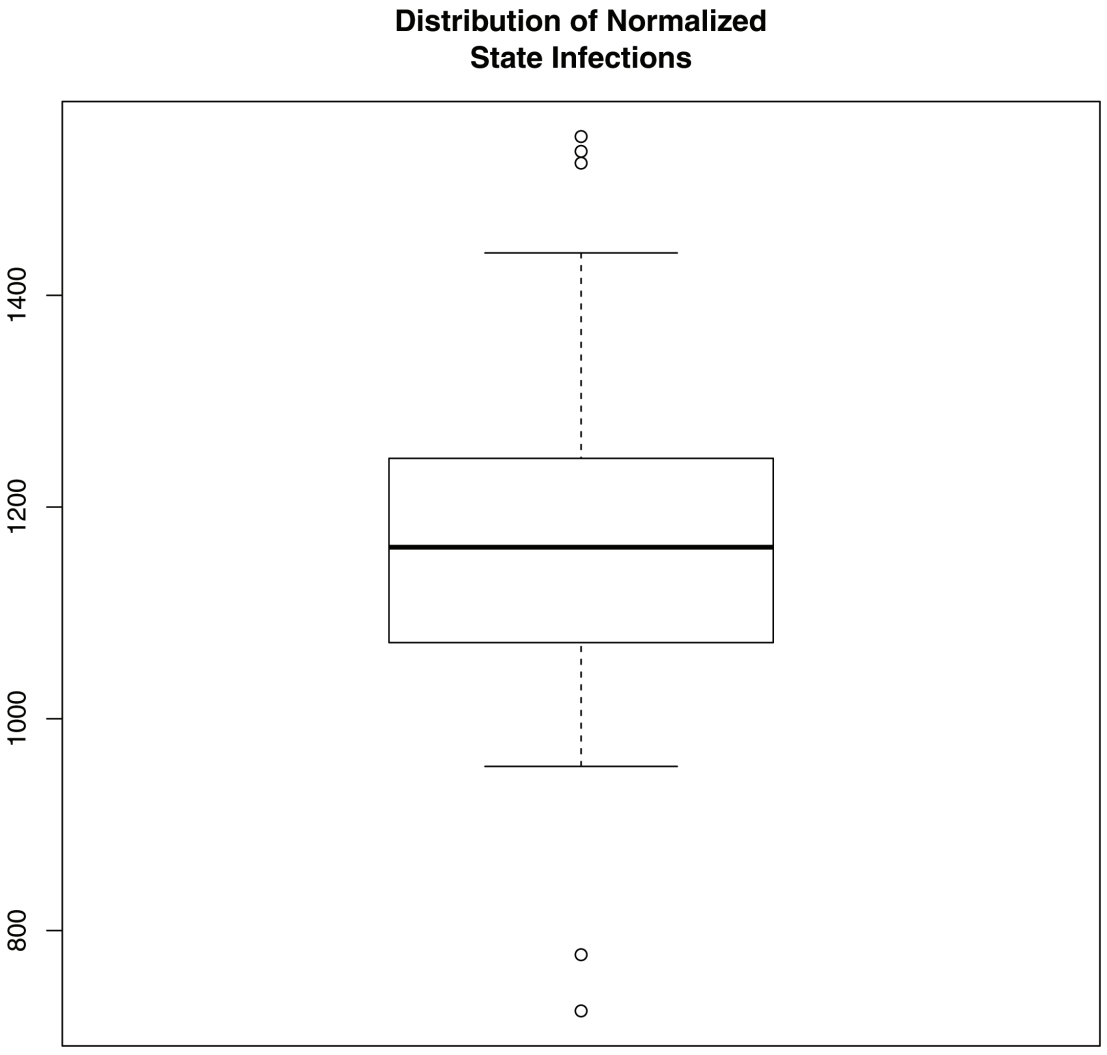

FIGURE 5-8 *Distribution of normalized state infections*

LISTING 5-10

```
# requires objects: za.norm (5-8), popbox (5-9)
# the values that are considered outliers
print(popbox$out)
## [1]  777 1536 1525 1550  724

# pull the rows from za.norm that have those values
za.norm[za.norm$count %in% popbox$out, ]
```

```
##                  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, you could consider these five states as being odd (outliers).

Calculating a z-score to Find Outliers

Cap x2 / ?

There's another measure of determining oddballs; you can calculate what's known as a ***z-score***. It will help you get a feel for just how much of an outlier a point is by showing how many ***standard deviations*** from the ***mean*** it is. A z-score is most often used to compare distributions from completely different scales, a method sometimes labeled "standardizing" the data. In order to do this calculation, you need to know the standard deviation and mean of the distribution. Then, for each value in the distribution, you calculate how many standard deviations from the mean the observation is. That is, you subtract the mean from each value and divide by the standard deviation. (See Listing 5-11.)

labeled

If your eyes started to glaze over from the z-score description, don't worry—every time we calculate one, we have to look up how it's done. You'll want to compare what you see in the distribution to something known as the "empirical rule" of a standard normal distribution. In a normal distribution (the familiar bell curve, which is also known as the ***Gaussian distribution***), you expect that roughly 68 percent of the distribution will fall within one standard deviation (above or below) of the mean, and 95 percent of the data will fall within two standard deviations, and then 99.7 percent should be within three standard deviations.

One point to note—this method doesn't work well if the data is skewed, so you should probably check a quick histogram (pass `za.norm$count` into the `hist()` function) to be sure it's not obviously skewed.

When using this approach, anything outside of three standard deviations is typically labeled as an outlier, and you might even consider anything more than two standard deviations as a possible outlier.

LISTING 5-11

```
# requires objects: za.norm (5-8)
# 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
# pull out values where absolute value of z-score is > 2
za.norm[which(abs(za.norm$z)>2), ]
##                  region count    z
## 8  district of columbia   777 -2.4
## 43                 utah  1536  2.2
```

(continues)

Listing 5-11 *(continued)*

```
## 44              vermont  1525  2.1
## 46           washington  1550  2.2
## 49              wyoming   724 -2.7
```

It appears those same five entries fall within three standard deviations. This knowledge calls into question the use of population in the normalization process (perhaps "Internet users" would be a better measure—hint, hint).

Rather than focus on solving things at the state level, you could bring this data down to the county level within the states. Doing so will supply more data points and allow a finer separation of the population, which will open up more possibility.

Au: Note OK set here?

Yeah, OK

Note

Overall, it'd be okay to consider these values within expectation given they are within three standard deviations, but if you had time, it would be a good practice to look into these more and be sure the measurements are valid. The takeaway is that you must answer the question, "Is this weird?" with either a squishy "probably not" or a non-committal "not so sure."

s / that

What's the p-value?

Cap x2 / ?

OK

Trying to identify ***weird*** versus ***normal*** is a core concept within statistics, and, depending on the circumstances, there is usually more than one way to identify it. At the heart of "statistically significant" is knowing whether something is weird or just the result of natural variations. One very common and widespread approach is the p-value. 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, you begin with a statement (technically called a ***null hypothesis***) and calculate the probability of the 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) assumptions like it's the probability of the statement being true (it's not).

comma

comma

To complicate this concept 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. You will be revisiting the value of p-values when you read about regression analysis later in this chapter. Just tuck the term "p-value" away in your memory bank as a measure of significance or, in this case, of "weirdness."

Counting in Counties

It is difficult to generalize at the state level because, well, it is a very generalized population. You would be obscuring a wide range of diversity among people behind a single label. You would be hard-pressed

to calculate the influence of something like income or—more importantly—alien visits on ZeroAccess infections at the state level. You can get more granular by repeating `latlong2map()` again, but at the county level.

There are a few additional items to consider as you get into a 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 that just over four out of five entries are accurate to about 25 miles and about one out of seven are resolved to an incorrect city. Does that mean you should be very wary of this data? In order to answer that question, you need to understand a statistical concept—natural variations will cancel out more often than stack up, especially as you get more data (and over 3,000 U.S. counties does represent ***more*** data).

lc

do

Does Variation Stack or Cancel?

Within statistics, natural variations generally cancel each other out, but this is counterintuitive to fields in engineering (like computer science) where people are taught that if they add components that all have a slight variation, the effect will compound itself and they should expect a wide range of results. What's the difference? Which viewpoint is right?

comma

This is kind of a tricky concept so consider an example. Say you are manufacturing a physical part and you want it to be 100 millimeters (mm) 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 they stack up 100 of those parts, they can expect something equally likely between plus or minus two times the number of parts (100). Meaning, it is possible that all 100 parts will be 98 millimeters, or it's possible that all the parts will be 102 millimeters, so they can expect a wide range in the output. The more they stack, the wider the range of output.

In statistics, if you can assume that each part has an equal chance of being any length within the range (and you'll want to validate that assumption in the real world), the differences in lengths will begin to cancel each other out. Thanks to a basic understanding of programming, you can model this and see how variation occurs across multiple parts.

it / the / some / the

The example generates 100 parts and uniformly "manufactures" them between 98 and 102 millimeters. It then averages the length (could also be sum or something other measurement, but mean works here). The engineering brains out there will guess that this will appear between 98 and 102, but let's see (Listing 5-12).

LISTING 5-12

```
#setting seed for reproducibility
set.seed(1492)
# run 100 times, getting random values between 98 and 102
mean(runif(100, min=98, max=102))
## [1] 100.0141
```

After one run, you get 100.0141. Let's manufacture 10,000 sets of 100 stacked parts and see how many get to the edge of the range (Listing 5-13). Surely if it's possible, you should see at least a few sets in 10,000 push towards the edge, right?

toward
(style sheet)

Wasn't this some type of call-out section? It's non-sequitor and should have some difference to it.

Continued - this should be a feature of some kind.

LISTING 5-13

```
#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)
## [1]  99.57977 100.47559
```

What is up with this? Even with 10,000 iterations of 100 random parts, none of them get close to the ranges of 98 or 102. You can visualize all of the parts in a quick histogram by running `hist(`***parts***`)`. You 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 than 100 in a set, try 1,000 or 10,000 in the `runif` function). As you add more parts within the range, the results are more likely to cluster even closer 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, ***you shouldn't toss out less-than-perfect data***. If the variations are caused by natural or random variations, you can assume the variation has more of a cancelling effect than a stacking effect. Now, this doesn't mean you should ignore variations like this, but instead it means that the variation will have less of an impact on throwing the analysis off than you think. You should still account for this variation in your work.

Relating this back to the analyses, you have all sorts of items in the spatial data that may be throwing off the 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, end up in New Hampshire). Several of the data points will be farther off than that. But these might cancel out if the variation is random (meaning points in New Hampshire could also just as easily end up in Maine). This doesn't mean the data is worthless. Until you can learn some more advanced techniques, you can just take the error introduced as a grain a salt. In other words, you can use this data to estimate how much of an effect alien visits have, but you wouldn't want to balance the fate of a company on this analysis—at least not without a lot more rigor and investigation of the data.

Moving Down to Counties

To transition the data down to the county level, you'll begin by calling the same `latlong2map()` function on the same ZeroAccess data, but ask it to translate to the county names (Listing 5-14). Keep in mind, there are over 3,000 counties in the United States 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. Like last time, you want to ignore anything that doesn't resolve in the United States (is set to `NA` in the data) and account for the Potwin Effect (anything below country should account for it). But rather than count things with `table()` and toss them into a data frame, you 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. You can

use the `strsplit()` function to split the county names. It returns a list object, so you 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 you'll transform this into a matrix with two 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 you're beginning to see how fun this data-munging thing can be, right?

LISTING 5-14

```
# requires package: maps, maptools
# requires objects: za (5-1), latlong2map (5-3)
## now mapping lat/long down 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")
head(za.county)
##    region subregion infections
## 1 alabama   autauga         44
## 2 alabama   baldwin        184
## 3 alabama   barbour         13
## 4 alabama      bibb         13
## 5 alabama    blount         26
## 6 alabama   bullock         11
```

You now have a data frame with three columns—the state, county, and count of infections—and you need to label the columns accordingly. There is a lingering "so what?" you 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. You may see some hot spots and you may be able to compare different areas on the map, but you can't really learn much from this amount of detailed data on a map. Moving forward, you'll switch from creating maps with this data to performing some ***real*** analysis to see whether you can find an explanation for the infections.

Au: OK Note set here?

No, this should be in below the section below titled "Introducing Linear Regression", not in this section.

> ***Note***
>
> *This section applies some techniques that you should not take lightly. This section focuses more on walking through the concepts and techniques rather than attempting to perform insightful research. Using statistical methods without fully understanding them is akin to an unlicensed teenager taking a car out for a spin.*

You'll need to pull in other data here (also split out by the county), just as you did at the state level with population. Then you may be able 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 want to cover.

We scoured the Internet, pulled together a collection of rather interesting data points, and did the data-munging to produce the data you'll use here. For the purposes of creating a tutorial, we've extracted a few statistics by county from various places and made it available on the book's website (`county-data.csv` on `www.wiley.com/go/datadrivensecurity` as part of Chapter 5's download materials).

rebreak

period

period

period

period

- `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).

period

rebreak & delete hyphen

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 you 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 this case, you 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.

Cap

comma

comma

Feel free to dig into the values, but sure enough, 90 percent of the uninfected counties have a population of less than 10,000. By specifying `all.x=T` in the `merge()` command, you are telling it not to drop any rows from the x data, which is the first argument passed into the `merge()` function, or ***county.data*** in the command (see Listing 5-15). To help illustrate we are including the argument labels when we call `merge()`.

LISTING 5-15

```
# requires objects: za.county (5-14)
# 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(x=county.data, y=za.county, all.x=T)
```

```
# replace all NA's with 0
za.county$infections[is.na(za.county$infections)] <- 0
summary(za.county)
##      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          infections
## Min.   :        0   Min.   :  0.000   Min.   :   0.00
## 1st Qu.:     5367   1st Qu.:  0.000   1st Qu.:   6.00
## Median :    15289   Median :  2.000   Median :  17.00
## Mean   :   387973   Mean   :  7.943   Mean   :  83.33
## 3rd Qu.:    62594   3rd Qu.:  6.000   3rd Qu.:  55.25
## Max.   :223441040   Max.   :815.000   Max.   :7692.00
```

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

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

Introducing Linear Regression

This section discusses a collection of techniques loosely called ***linear regression***, but it's important to know that college courses focus on nothing but linear regression for a semester and still don't cover all aspects of it. Books such as ***Applied Linear Statistical Models*** are over 1,300 pages 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 clear up some of the mystery around regression analysis and put the technique in context, while at the same time introduce enough warnings and common pitfalls so that you don't end up shooting yourself in the foot with this powerful and flexible technique.

Regression analysis is a workhorse and it is behind many of the scientific findings you hear about. Works that say, "Scientists find a link between something and something else" are almost always based on regression analysis. Researchers use regression analysis for two general purposes.

- **First, it can be used to estimate how different observable inputs contribute to an observable output**. In this case, you want to estimate how alien visits to U.S. counties (observable inputs) contribute to the rate of ZeroAccess infections in that county (observable output). With regression analysis, not only can you estimate the significance (or lack thereof) of each variable, you can also estimate how strong that contribution is. Don't worry if it is a bit confusing now; we will cover this more as you get into the data. Regression analysis is a powerful tool to describe relationships between observations.

an

Au: Ref?
In reference section

comma

- **The second purpose for regression analysis is *prediction***. The output of regression analysis is a formula. Given specific inputs, you 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 you add other variables such as male and female, age, and so on, you can determine an expected value, and establish an expected range of a person's weight. This is the method doctors use to tell patients they are above or below their expected weight, height, and so on. Regression analysis is a powerful tool for estimating and comparing observed outputs.

To demonstrate these two purposes, you'll create fictitious (and rather simple) data. You'll start with a single input variable and generate random data points from a normal distribution (any distribution will work, the normal is just pretty). You'll use the `rnorm()` command and create 200 points with a mean of 10 and a standard deviation of 1 (the default). See Listing 5-16.

LISTING 5-16

```
# 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.
## 7.785   9.386   9.951  10.040  10.610  12.400
```

If you look at the summary in Listing 5-16, you see the result is data that ranges from 7.2 to 12.9. You now need to generate the output data. You want to create a linear relationship between the input and the output, so you'll pass the mean in as double the input variables. By using `rnorm()` you are introducing more random variations so you don't have perfectly linear relationship, but by centering the randomness (mean) on the input variable you are creating enough of a linear relationship to model. You then create a data frame out of the input and output for easy handling and plotting.

With the input and output created, you can pass all of this into `ggplot` (Listing 5-17) and create a scatterplot to visualize the relationship. Let's add something special by including the `geom_smooth()` function and telling it to use a linear model, `"lm"`. This will overlay a single straight line that best describes the relationship between the input and output data (see Figure 5-9).

LISTING 5-17

```
# 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)
gg <- ggplot(our.data, aes(input, output))
gg <- gg + geom_point()
gg <- gg + geom_smooth(method = "lm", se=F, color="red")
gg <- gg + theme_bw()
print(gg)
```

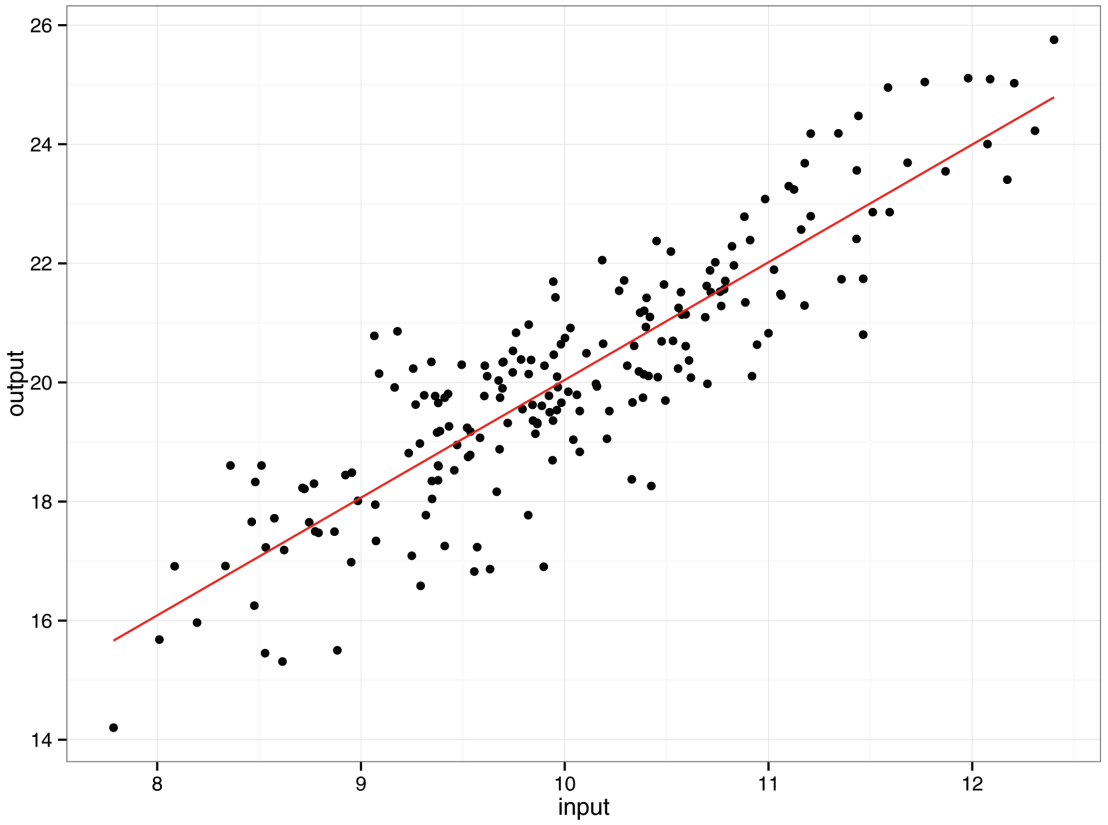

FIGURE 5-9 *Sample data with regression line*

You can see from Figure 5-9 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 the data, you call one very simple command (Listing 5-18):

LISTING 5-18

```
# requires objects: input (5-16), output (5-17)
model <- lm(output ~ input)
```

Congratulations! You have just run your first linear regression. Take a look at the output with the `summary()` function, and we'll walk through it.

```
summary(model)
## Call:
## lm(formula = output ~ input)
##
```

```
## 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 you used ("Call") and a summary of the ***residuals***. The residuals are the difference between what the model predicts and what you 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 this residuals section, as there are better methods for interpreting the residuals (we will revisit the evaluation of residuals in Chapter 9).

Oftentimes

Strike this whole reference.

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

that

Looking at the coefficients, you can construct the model:

$$\text{output} = 0.27224 + 1.97692 \times \text{input}$$

You would use this model to estimate new output values given an observed input (or just use the `predict.lm()` command passing in your model and new input variables to predict with). But remember that you generated the data by multiplying the input by 2? The linear model here thinks you multiplied by 1.97692, which is pretty close. This coefficient for the input variable (or variables) is where you can begin to see the power of regression analysis when used for inference about the input variables. You can interpret it like this:

Au: style OK?

OK

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

Since you have only one input variable, you have nothing else to hold constant. Even if you have dozens of variables, you can isolate the effect of the individual variables with regression analysis. In the next section you'll go back to the ZeroAccess infection data and use this approach to make inferences about the effects of alien visits on infections.

The next column in the coefficients represents the standard error. You can use this along with the estimated coefficient to generate a confidence interval for the coefficient. Or you can do this by passing the output of the `lm()` command to the `confint()` function (Listing 5-19) and the confidence interval is calculated for you:

LISTING **5-19**

```
# requires objects: model (5-18)
confint(model)
##                 2.5 %   97.5 %
## (Intercept) -1.263895 1.808368
## input        1.824502 2.129343
```

The output tells you that, with 95 percent confidence, the input coefficient is between 1.82 and 2.13 (the 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 is called the p-value, which was introduced earlier in this chapter. As a simplification, smaller p-values contribute more significantly to the overall model and larger p-values mean the relationship between this input variable and the output is more likely to be chance. When you have a high p-value, you might 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. This means that if the p-value is less than 0.05 (and your p-value is well beneath it), the variable is significant and should stay in the model. When you have a p-value that is above 0.05, you should consider tossing it to the curb. Although, it's slowly becoming a common practice to evaluate p-values using at least three different thresholds of significance: 0.1, 0.05, and 0.01, which allows some flexibility to enter into the model and reduces an arbitrary cutoff point that was traditional in academic publications. You can see in the output of the model in Listing 5-16 that it denotes which level of significance our p-value is rated at (0.001 in this case).

There are two other points to consider with linear regression output. 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 the model. Values ranges from 0, meaning the model is no better than using the output mean, to 1, meaning the model describes the output perfectly. In this model it was calculated as 0.76, which means the linear model can reasonably explain 76 percent of the variation in the output data. There is no magic number you want the R2 to be because it's relative. If you are starting from a place where you are simply guessing at the output (that is, you can't explain any of the variation in the output), then an R2 of 0.05 is a little helpful. But if you have an existing model at 0.76, then 0.05 is a large step backward.

Au: Please check if Listing 5-16 is correct.

No, 5-18

Au: Is it clear to the reader where to look for this?

Could make it "in the second-to-last-line in Listing 5-18."

This must be Listing 5-18

rebreak

The "R2" here needs to have the 2 to be "sqauared" like it is in the note.

> **Note**
>
> *When people want a quick understanding of a model, they focus on the R^2 value.*

Like This

Au: Is it clear to the reader where to look for this?

Could add "…on the bottom line in Listing 5-18"

The last thing to consider is the p-value on the bottom line. This is the p-value of the entire model. At this point, you probably have a feel as to whether this is a good model or not, but keep an eye on this p-value. In this example, the p-value is tiny, so you should have confidence in 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 so we must also include some of the common pitfalls.

You Cannot Extrapolate Beyond the Data

Your data represents the entire range of your knowledge. You can verify that there is a linear relationship in the data you have, but you cannot extend that belief beyond the input values. As an example, say you've developed a simple model to estimate the cost of a data breach (output) from a count of records lost (input). If you look only at breaches that have lost 1,000 to 100,000 records, you cannot extend this to breaches with more than 100,000 or less than 1,000 records lost. You have no confidence that the relationship holds beyond the data you have. (Although if you did develop such a blatantly ridiculous model, you'd be sure to discuss the small R2 value so people may have a fair shot at dismissing such a simple model.)

The "2" must be raised as above.

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 you 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 good practice to verify its validity before continuing. Sometimes outliers are valid and you must include them and account for them in the model. Other times, they may just be a result of mistyping or recording something in different units of measurement. Those types of outliers should be fixed or removed.

comma

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 you have to approach these relationships 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 you want to be careful of something called ***multicollinearity***. If you have two or more input variables that are highly correlated to each other, you may be incorrectly assigning meaning where none exits. You will see an example of this when you get back to the ZeroAccess data later in this chapter.

Too Many Variables

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 concepts beyond regression analysis. These misleading findings occur because as more and more variables are added to the model, the likelihood of a spurious relationship (statistically significant correlation caused by pure chance) increases and

is exacerbated if the analysis is complex or done without the common sense of domain expertise. It's common and a good practice with linear regression to seek out the smallest set of input variables and not uncommon to exclude variables that only marginally improve the model for the sake of simplicity (even if the variable has a tiny p-value). This is also leading to a discussion of over-fitting, which is when the model works really well on the initial data, but performs quite poorly when applied to real data.

comma

overfitting

> ***Note***
>
> *We'll be discussing challenges with overfitting in Chapter 9 and methods to reduce overfitting.*

OK

Visualize and Apply the Sniff Test

It's a good idea to visually inspect the data before jumping into regression analysis. In the example here, you created a simple scatterplot and added the regression line. This gets a little more complicated as you add multiple variables, but it's a good habit to get into. But even beyond that, you 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.

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 you can screw up when using it. Now you can start to pull more meaning from the spatial data than maps would allow.

understanding

the

Let's do a simple regression on the real data and see how well "visits from aliens" describes ZeroAccess infections. Although 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. You 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, you again call the built-in `lm()` function and specify the output variable (`infections` in the `za.county` data frame), and then the tilde character followed by variable with alien sightings (see Listing 5-20). If you wanted to add more variables you could add them—literally—with the plus (+) symbol. If you wrap the whole command in the `summary()` call, you get the output immediately. The output is trimmed to the relevant information.

comma

comma

the

LISTING 5-20

```
# requires object: za.county (5-14 and 5-15)
summary(lm(infections ~ ufo2010, data=za.county))
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 17.97998    2.63775   6.816 1.12e-11 ***
## ufo2010      8.22677    0.08843  93.029  < 2e-16 ***
## ---
```

(continues)

Listing 5-20 (*continued*)

```
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 140.9 on 3070 degrees of freedom
## Multiple R-squared:  0.7382,  Adjusted R-squared:  0.7381
## F-statistic:  8654 on 1 and 3070 DF,  p-value: < 2.2e-16
```

Using your new skills, you 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 you that for every UFO sighting, you should expect about eight more ZeroAccess infections. This is an incredibly strong model, and there is enough to submit to a hoity-toity peer-reviewed journal explaining how you have scientifically proven that UFOs are causing the spread of ZeroAccess malware! We can see the headlines already:

Researchers Link ZeroAccess Infections to Alien Visitors

Before you get ahead of yourself, maybe you should looks at some of these other variables. Even though we cautioned about adding in too many variables, you'll need to explore these relationships and various models. Chapter 9 will discuss some techniques for variable selection. For now, run another regression with all of these variables and see what happens (Listing 5-21).

LISTING 5-21

```
# requires object: za.county (5-14 and 5-15)
summary(lm(infections ~ pop + income + ipaddr + ufo2010,
           data=za.county))
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)  1.091e+01  5.543e+00   1.968   0.0492 *
## pop          7.700e-04  9.072e-06  84.876  < 2e-16 ***
## income      -2.353e-04  1.215e-04  -1.937   0.0528 .
## ipaddr       2.281e-06  3.027e-07   7.534 6.41e-14 ***
## ufo2010      5.495e-01  9.943e-02   5.526 3.54e-08 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 74.9 on 3067 degrees of freedom
## Multiple R-squared:  0.9261,  Adjusted R-squared:  0.926
## F-statistic:  9610 on 4 and 3067 DF,  p-value: < 2.2e-16
```

Scanning down the p-values, it looks like all of these are quite tiny with the exception of income, which looks like it may be suspect, and you could try re-running this with income removed. However, the influence of IP address and UFO visits still appear strong. Notice that as you've added more variables, the influence of UFO visits has dropped (the coefficient is smaller and p-value increased) and is accounted for in other variables.

Although you have all of these variables in this model, you should check for something we discussed in the "Understanding Common Pitfalls in Regression Analysis" section earlier in this chapter called

that x2

Au: style OK?

It's a headline type reference, maybe make it bold?

the

multicollinear variables. This is when two or more of the input variables are correlated and that relationship is masking the [in]significance of a variable. You 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 (see Listing 5-22). As a general rule, if the square root of the variance inflation is greater than 2 (something I have to look up every time I do this), the variables are correlated and you shouldn't trust that both are significantly contributing to the model.

LISTING 5-22

```
# requires object: za.county (5-14 and 5-15)
library(car) # for the vif() function
model <- lm(infections ~ pop + income + ipaddr + ufo2010,
            data=za.county)
sqrt(vif(model))
##      pop   income   ipaddr  ufo2010
## 2.165458 1.038467 1.046051 2.115512
```

You 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, you normalize the population. To do so, you just divide both values by the population, making them infections and sighting per capita, and rerun the single regression (Listing 5-23).

LISTING 5-23

```
# requires object: za.county (5-14 and 5-15)
za.county$za.by.pop <- za.county$infections/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) 7.050e-04  1.213e-05  58.106  < 2e-16 ***
## ufo.by.pop  2.679e-01  6.956e-02   3.852  0.00012 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.0005793 on 3070 degrees of freedom
## Multiple R-squared:  0.004809,  Adjusted R-squared:  0.004485
## F-statistic: 14.84 on 1 and 3070 DF,  p-value: 0.0001197
```

the "2" should be raised as above

Great! The p-value is still under 0.05! But . . . oh . . . wait a second, the R2 value is telling you that this model is quite useless, as it describes 0.4 percent of the data (R2 is 0.004). 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 that all of this data is available for download from the book's website along with the code in this chapter. There is plenty of room for exploration here.

What Is Correlated to ZeroAccess Infections?

Say you have a strong suspicion (or you've applied some of the variable selection techniques discussed in Chapter 9) that the population of a county is the best overall ***predictor*** of how many infections occur in that county (see Listing 5-24).

LISTING 5-24

```
# requires object: za.county (5-14 and 5-15)
summary(lm(infections ~ pop, data=za.county))
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) 4.545e-01  1.435e+00   0.317    0.752
## pop         8.204e-04  4.247e-06 193.199   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 75.92 on 3070 degrees of freedom
## Multiple R-squared:  0.924,  Adjusted R-squared:  0.924
## F-statistic: 3.733e+04 on 1 and 3070 DF,  p-value: < 2.2e-16
```

With an R^2 value of 0.94, you're going to be hard pressed to add more variables in here that mean much. Sure enough, when you cycle through the other variables, you'll find that income and the number of IP addresses in that county do not add much to the overall model. What you 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 you invert that number (1/0.0008313), you can determine that for about every 1,200 people a county has, you can expect one more infection of ZeroAccess.

Go back to the maps to see whether you can't visualize what this looks like at the county level. You can generate a choropleth map at the county level for the number of infections and the population. If the regression analysis is accurate, you should see a very clear relationship between the two.

Set Summary after Figure 5-10 / ?

Summary

You created quite a few maps in this chapter, with points and with choropleths. Although you can pick out variations across the map quite rapidly with the visual representation, you shouldn't rely solely on visualizations with spatial data. Even though the maps showed variation, the data showed through linear regression that ***population largely explains the variation***. That's something to consider when you're creating maps (or any other visualization for that matter). Be sure to take a step back and ask the ever-popular question of "So what?!" If you can't answer that question, maybe you don't need the map at all and the analysis needs to go in a different direction.

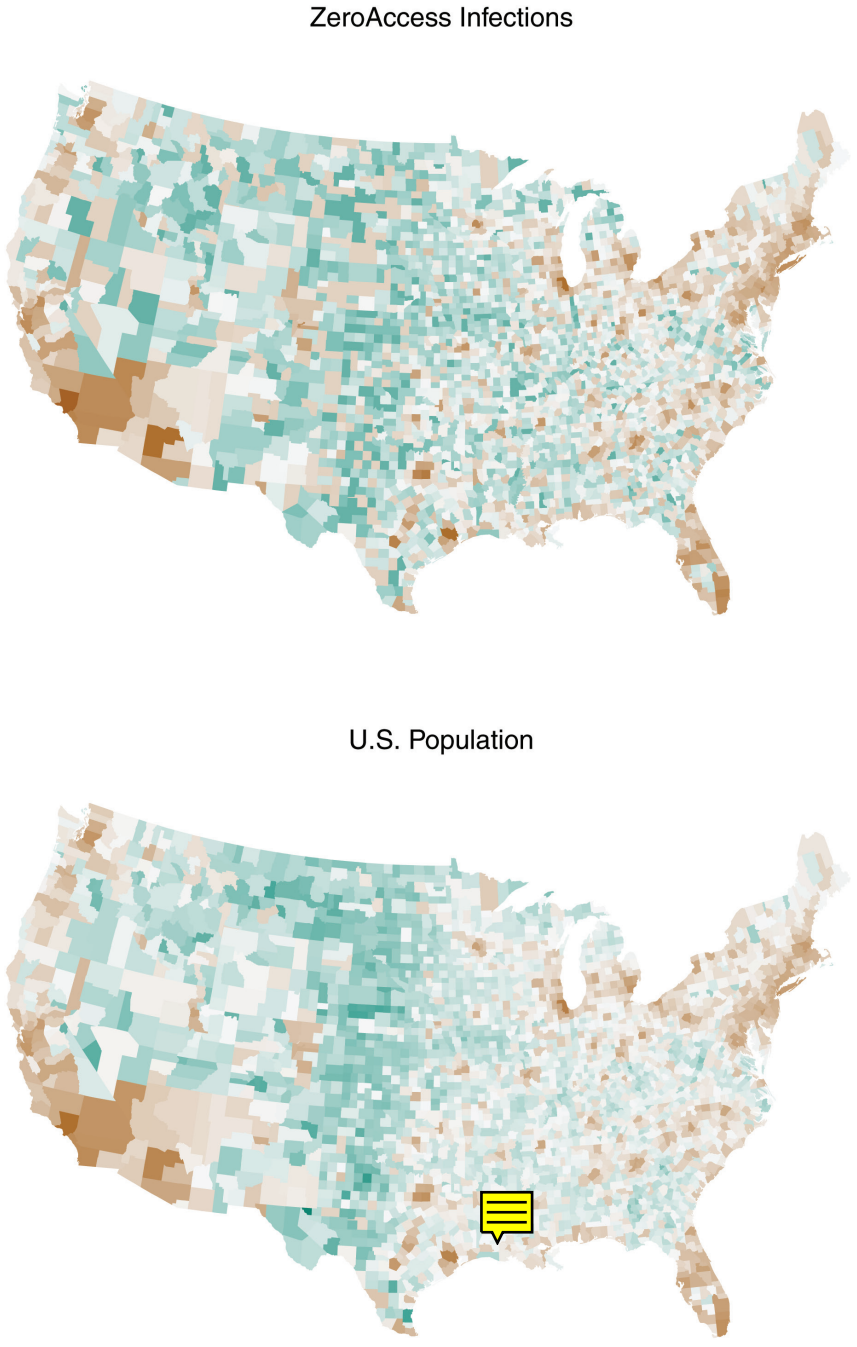

FIGURE 5-10 *Visual relationship between ZeroAccess infections and population*

What Citation? We made both these maps.

Recommended Reading

The following are some recommended readings that can further your understanding on some of the topics we touch on in this chapter. For full information on these recommendations and for the sources we cite in the chapter, please see Appendix B.

OK

Data Points: Visualization That Means Something **by Nathan Yau**—Yau provides several beautiful geospatial visualizations and discusses the design principles behind them. There is no example code, but plenty of inspiration among the pages, plus he included a map of UFO sightings.

R Graphics Cookbook **by Winston Chang**—If you will be doing any visualizations with R, you should have this book. Winston Chang goes into more depth on map creation than we do here and includes hands-on examples and explanations for R.

Naked Statistics: Stripping the Dread from the Data **by Charles Wheelan**—When it comes to introductory material on statistics, nothing beats this book. Wheelan presents the statistical concepts without heavy math and builds up to a chapter on linear regression, covering many of the assumptions and pitfalls of the technique.