AU: (*Global for chapter*) Please note all code lines highlighted in purple. You’ll need to rebreak these lines somewhere before the highlights. Do so by adding a hard return (Return) and then indenting the run-over line using spaces (not tabs). Please do not manually alter the style or indents.

AU: I’ve replaced all the code that was originally bolded with the “CodeHighlight” style—which, in the end, will result in the code being bold. Just a technicality, really.

Also, as mentioned in chapter 2, if any color is needed for the code in this chapter, please add it. You can find instructions in the author guidelines documentation. If you have questions, please contact Kevin right away. Thanks. --John

[[CE by Kezia Endsley]]

[[TE by Russell Thomas –RED has error that needs to be fixed, YELLOW needs attention. Some yellow is by CE]]

[AU: The TE has some real discrepancies he found in the code. Please address/fix the code as needed and resubmit after AR. Thanks, Kevin (PjE)]

Chapter 3: Learning the “Hello World” of Security Data Analysis

“From one thing, know ten thousand things.”

―Miyamoto Musashi, *The Book of Five Rings*

If you’ve ever tried to learn a new programming language there’s a good chance you started off with a “Hello World” example that quickly introduces basic language structure and code execution. The immediate sense of accomplishment as the syntax is verified by the compiler/interpreter and the familiar two-word output is displayed becomes a catalyst for the notion that, soon, you shall have the ability to bend this new language to your will.

This chapter takes the “Hello World” concept and expands it to a walk-through of a self-contained, introductory security data analysis use case that you will be able to follow along with, execute, and take concepts from as you start to perform your own analyses. There are parallel examples in Python and R to provide a somewhat agnostic view of the similarities, strengths, and differences between both languages in a real-life data analysis context. If you’re not familiar with one or both of those languages, you should read Chapter 2 and at least skim some of the external resources referenced there.

This is a good place to reinforce the recommendation to use IPythonNotebooks or RStudio for your analyses and exploration as they provide very robust and forgiving environments, is which means you will be much more productive compared to the alternative of writing, saving, and executing scripts within the bare interpreter shells. Remember, all the source code, sample data, and visualizations are on the book’s website, so there’s no need for transcription. You can just cut/paste and focus on the flow and concepts presented in the examples.

Listing 3-0

*# This is for the R code in the chapter*

*# set working directory to chapter location*

*# (change for where you set up files in ch 2)*

setwd("~/book/ch03")

*# 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)

}

Listing 3-1

*# This is for the Python code in the chapter*

*# loads the necessary Python library for chdir*

import os

*# set working directory to chapter location*

*# (change for where you set up files in ch 2)*

os.chdir(os.path.expanduser("~") + "/book/ch03")

Solving a Problem

Chapter 1 emphasized the criticality of developing a solid research question before going off and “playing with data.” For this “Hello World” example, you are working on a problem given to you by the manager of the Security Operations Center (SOC). It seems the SOC analysts are becoming inundated with “trivial” alerts ever since a new data set of indicators was introduced into the Security Information and Event Management (SIEM) system. They have asked for your help in reducing the number of “trivial” alerts without sacrificing visibility.

[[Authors: Just a few notes about why I made the edits I did in case you’re wondering. We avoid using “we” when you mean author and reader and prefer “you” instead (“we” to represent several authors is fine). We avoid using past tense and future tense when possible and use present tense to discuss any information in the book. Also, we use the serial comma. Kezia]]

This is a good problem to tackle through data analysis, and we should be able to form a solid, practical question to ask after we perform some exploratory data analysis and hopefully arrive at an answer that helps out the SOC.

Getting Data

We are entering an age of data in information security. The challenge is no longer where to get data from, but what to do with it. And, the kind of information in each data set will drive the type of research you perform.

“silver age” is a lesser age *following* a golden age.

Roger that.

For this example, the SOC chose to integrate AlienVault’s IP Reputation Database (<http://labs.alienvault.com/labs/index.php/projects/open-source-ip-reputation-portal/download-ip-reputation-database/>) into the SIEM. AlienVault itself develops OSSIM—an open source security information manager—and a proprietary unified security management (USM) product, both of which make use of this freely available data set that contains information on various types of “badness” across the Internet. AlienVault provides this data in numerous formats free of charge. The version you work with is the OSSIM Format (<http://reputation.alienvault.com/reputation.data>) since it provides the richest information of all the available formats.

[[ The text below suggests that this is a “quick download” via the browser, but for me it took at least 45 seconds. Granted, I’m on a rate-limited guest network, but I’d suggest using a different word other than “quick”. ]]

AU: Should we go ahead and mention the name of AlienVault’s UTM? –John

Actually USM (they changed the name since the spring, so I changed it here) is the name ☺

type="tip"

AlienVault updates their IP reputation data set hourly and produces a companion “revision” file (<http://reputation.alienvault.com/reputation.rev>), enabling you to ensure you are working with the latest data set or keep a history of data sets. If you plan on performing a long term analysis of this data set—often referred to as a *longitudinal study*—it’s a good idea to script some code to perform this check to see if it’s time to download a new one, even in scheduled jobs.

When performing an exploratory analysis or getting a first look at a data set, you might find it helpful to perform an initial download via browser (or use wget/curl if you are handy on the command line). The AlienVault database hovers near 16MB, so it may take a minute or two to download on slower connections. When you download the AlienVault IP reputation database and examine the first few data elements, you can get an idea of the contents and format, which will come in handy when you start to read in and work with the data. Here, you use some simple Linux/UNIX commands to inspect the download:

Performing a Quick Review of the Downloaded Data Set

$ head -10 reputation.data*# look at the first few lines in the file*

222.76.212.189#4#2#Scanning Host#CN#Xiamen#24.479799270,118.08190155#11

222.76.212.185#4#2#Scanning Host#CN#Xiamen#24.479799270,118.08190155#11

222.76.212.186#4#2#Scanning Host#CN#Xiamen#24.479799270,118.08190155#11

5.34.246.67#6#3#Spamming#US##38.0,-97.0#12

178.94.97.176#4#5#Scanning Host#UA#Merefa#49.823001861,36.0507011414#11

66.2.49.232#4#2#Scanning Host#US#Union City#37.59629821,-122.0656966#11

222.76.212.173#4#2#Scanning Host#CN#Xiamen#24.479799270,118.08190155#11

222.76.212.172#4#2#Scanning Host#CN#Xiamen#24.479799270,118.08190155#11

222.76.212.171#4#2#Scanning Host#CN#Xiamen#24.479799270,118.08190155#11

174.142.46.19#6#3#Spamming###24.4797992706,118.08190155#12

$ wc –l reputation.data *# see how many total records there are*

258626 reputation.data

For most projects, it’s better to get into the habit of retrieving the data source directly from your analysis scripts. If you still prefer to download files manually you should provide some type of comment in your programs that provides details about where the source data comes from and when you retrieved the data for your current analysis. These comments make it easier to repeat the analyses at a later date, and trust us, you’ll revisit your code and analyses more often than you think.

The following examples show how to perform the data retrieval in both R and Python. If you are following along with RStudio or IPython, all the code examples assume a working directory of the top level of the project structure (such as executing in the book/ch03 directory that was suggested in Chapter 2, which you either manually created or created using the prep script we provided). Code blocks are, for the most part, self-contained, but each block expects this first snippet and the snippet in the next section on “Reading in Data” to have been executed in the running RStudio or IPython session.

[[Authors: Should that be ch02 directory, since that was chapter 2? Kezia]]

[[ No. the proper directory is book/ch03, which is consistent with the text and script in Chapter 2.

However, some additional instruction should be given on how to set the working directory in both Rstudio and IPython. In Rstudio, I found that I needed to use the menu “Session/Change Working Directory…”, which executed this command: setwd("~/book/ch03")

]]

R Code to Download the AlienVault Data

Listing 3-2

*# URL for the AlienVault IP Reputation Database (OSSIM format)*

*# storing the URL in a variable makes it easier to modify later*

*# if it changes. NOTE: we are using a specific version of the data*

*# in these examples, so we are pulling it from an alternate*

*# book-specific location.*

avURL <-

"http://datadrivensecurity.info/book/ch03/data/reputation.data"

*# use relative path for the downloaded data*

avRep <- "data/reputation.data"

*# using an if{}-wrapped test with download.file() vs read.xxx()*

*# directly avoids having to re-download a 16MB file every time*

*# we run the script*

if (file.access(avRep)) {

download.file(avURL, avRep)

}

## trying URL 'http://datadrivensecurity…/ch03/data/reputation.data'

## Content type 'application/octet-stream' length 17668227 bytes

## opened URL

## ==================================================

## downloaded 16.8 Mb

[[ I get the following error when I ran this in RStudio:

trying URL 'http://www.dropbox.com/s/auj4tjrau83ed83/reputation.data'

Error in download.file(avURL, avRep) :

cannot open URL 'http://www.dropbox.com/s/auj4tjrau83ed83/reputation.data'

The problem seems to be that the URL takes me to the download page, not the file download itself. I confirmed this by loading the URL into the browser

.

Python Code to Download the AlienVault Data

Listing 3-3

*# URL for the AlienVault IP Reputation Database (OSSIM format)*

*# storing the URL in a variable makes it easier to modify later*

*# if it changes. NOTE: we are using a specific version of the data*

*# in these examples, so we are pulling it from an alternate*

*# book-specific location.*

import urllib

import os.path

avURL = "http://datadrivensecurity.info/book/ch03/data/reputation.data"

*# relative path for the downloaded data*

avRep = "data/reputation.data"

*# using an if-wrapped test with urllib.urlretrieve() vs direct read*

*# via panads avoids having to re-download a 16MB file every time we*

*# run the script*

if not os.path.isfile(avRep):

urllib.urlretrieve(avURL, filename=avRep)

[[ This doesn’t work as described. Unlike the R code, it does not give me an error message, but it still doesn’t work properly. Instead it puts a parsed version of the download web page into the file reputation.data – 27KB total.

Also, the yellow highlighted line seems to indicate that this is a shell script. If that is what is intended, the you should explicitly say “run this from a terminal shell”. ]]

AR: inadvertently moved the dropbox data. Gave it it’s own domain now.

The R and Python code looks very similar and follow the same basic structure: using variables whenever possible for URL and filenames plus testing for the existence of the data file before downloading it again. These are good habits to get into and we’ll be underscoring other suggested good practices throughout the rest of the book.

With the IP reputation data in hand, it’s now time to read in the data so you can begin to work with it.

Reading In Data

R and Python (especially with pandas) abstract quite a bit of complexity when it comes to reading and parsing data into structures for processing. R’s read.table(), read.csv(), and read.delim() functions and pandas' read\_csv()function cover nearly all your delimited file-reading needs and provide robust configuration options for even the most gnarly input file. Both tools, as you learn in later chapters, also provide ways to retrieve data from SQL and NoSQL databases, HDFS “big data” setups, and even handle unstructured data quite well.

AU: The highlights above are “functions,” right? If not, please specify what they are, similar to what I’ve done by adding “function.” I see later in the chapter you refer to the table() and sample() “commands.” Ideally, we’d be consistent in our terminology. Okay to change those to “() function”? --John

AR: Yes. Agreed that it might be confusing to folks new to R. The terms are used fairly interchangeably amongst R-folk, but consistency shld be the aim here.

type="note"

The Revolution Will Be Properly Delimited!

AU: I’d suggest a different title, as this sidebar also mentions TSV and JSON. --John

Base R and Python’s pandas package both excel at reading in delimited files. Although they are also both agnostic when it comes to what that delimiter is, there is a general acceptance in the data science community that it should either be a comma-separated value (CSV) or a tab-separated value (TSV), and the majority of the sample data sets available to practice with come in one of those two flavors. The CSV format is thoroughly defined in RFC 4180 (http://www.rfc-editor.org/rfc/rfc4180.txt) and has the following high-level attributes:

AU: Note edit above to specify that the RFC is for CSV. Shouldn’t we also include the same (RFC and attributes, if they differ) for TSV, given the importance you give it below? If these attributes also apply to TSV, let’s note that.   
  
Also, is “character” necessary, or can the sentence stand without it? Is that term redundant with “value”? Thanks --John

* There should only be one record per line.
* Data files can include an optional header line.
* Header and data rows have fields separated by commas (or tabs).
* Each line should have the same number of fields.
* Spaces in fields should be treated as significant.

AU: For parallel structure, please rephrase the first two bullets to be complete sentences. --John

Though RFC 4180 explicitly specifies the comma as the separator, the same rules apply when using tabs (there is no corresponding RFC for tab-separated files).

Many tools in the security domain can import and export CSV-formatted files. If you intend to do any work in environments like Hadoop, you haveto become familiar with CSV/TSV.

Another established format is JSON (JavaScript Object Notation), which has grown to become the preferred way to transport data between servers and browsers. As you’ll see in Chapter 8, it is also the foundational data format behind many NoSQL database environments/tools. The JSON format is defined in RFC 4627 (http://www.rfc-editor.org/rfc/rfc4627.txt) and has two primary structures:

* A collection of name/value pairs (a “dictionary”)
* An ordered list of values (an “array”)

JSON enables richer and more complex data representation than CSV/TSV and is rapidly superseding another popular, structured format—the Extensible Markup Language (XML)—as the preferred *data exchange* representation. This is because it’s syntactically less verbose, much easier to parse, and (usually) more readable. XML has and will continue to excel at document representation, but you should strongly consider using JSON for your structured data-processing needs.

From a cursory examination of the downloaded file, you can see the AlienVault data has a fairly straightforward record format with eight primary fields using a # as the field separator/delimiter.

222.76.212.189#4#2#Scanning Host#CN#Xiamen#24.479799270,118.08190155#11

Notice also that the reputation data file lacks the optional header, so the example code segment assigns more meaningful column names manually. This is a completely optional step, but it helps avoid confusion as you expand your analyses and, as you see in later chapters, helps build consistency across data frames if you bring in additional data sets.

The consistency in the record format makes the consumption of the data equally as straightforward in each language. In each language/environment, we follow a typical pattern of:

* Reading in data
* Assigning meaningful column names (if necessary)
* Using built-in functions to get an overview of the structure of the data
* Taking a look at the first few rows of data, typically with the head() function

that we’ll cover in more detail in Chapter 4.

The code below builds on the code from the previous section. It won’t work correctly otherwise. This is the pattern we will follow in the book, so you should load and run the code in each chapter sequentially.

Add:

“The code below builds on the code from the previous section. It won’t work correctly otherwise. This is the pattern we will follow in the book, so you should load and run the code in each chapter sequentially.” ]]

[AU: That’s a good idea! Thanks, Kevin (PjE)]

AR: Agreed. Done.

R Code to Read in the AlienVault Data

Listing 3-4

*# read in the IP reputation db into a data frame*

*# this data file has no header, so set header=FALSE*

av <- read.csv(avRep,sep="#", header=FALSE)

*# assign more readable column names since we didn’t pick*

*# any up from the header*

colnames(av) <- c("IP", "Reliability", "Risk", "Type",

"Country", "Locale", "Coords", "x")

str(av)

## 'data.frame': 258626 obs. of 8 variables:

## $ IP : Factor w/ 258626 levels "1.0.232.167",..: 154069 154065

## 154066 171110 64223 197880 154052 154051 154050 56741 ...

## $ Reliability: int 4 4 4 6 4 4 4 4 4 6 ...

## $ Risk : int 2 2 2 3 5 2 2 2 2 3 ...

## $ Type : Factor w/ 34 levels "APT;Malware Domain",..: 25 25 25 31 25

## 25 25 25 25 31 ...

## $ Country : Factor w/ 153 levels "","A1","A2","AE",..: 34 34 34 143

## 141 143 34 34 34 1 ...

## $ Locale : Factor w/ 2573 levels "","Aachen","Aarhus",..: 2506 2506

## 2506 1 1374 2342 2506 2506 2506 1 ...

## $ Coords : Factor w/ 3140 levels "-0.139500007033,98.1859970093",..:

## 489 489 489 1426 2676 1384 489 489 489 489 ...

## $ x : Factor w/ 34 levels "11","11;12","11;2",..: 1 1 1 7 1 1 1 1 1

## 7 ...

head(av) *# take a quick look at the first few rows of data*

## IP Reliability Risk Type Country Locale

## 1 222.76.212.189 4 2 Scanning Host CN Xiamen

## 2 222.76.212.185 4 2 Scanning Host CN Xiamen

## 3 222.76.212.186 4 2 Scanning Host CN Xiamen

## 4 5.34.246.67 6 3 Spamming US

## 5 178.94.97.176 4 5 Scanning Host UA Merefa

## 6 66.2.49.232 4 2 Scanning Host US Union City

## Coords x

## 1 24.4797992706,118.08190155 11

## 2 24.4797992706,118.08190155 11

## 3 24.4797992706,118.08190155 11

## 4 38.0,-97.0 12

## 5 49.8230018616,36.0507011414 11

## 6 37.5962982178,-122.065696716 11

Python Code to Read in the AlienVault Data

Listing 3-5

*# first time using the pandas library so we need to import it*

import pandas as pd

*# read in the data into a pandas data frame*

av = pd.read\_csv(avRep,sep="#")

*# make smarter column names*

av.columns = ["IP","Reliability","Risk","Type","Country",

"Locale","Coords","x"]

print(av) # take a quick look at the data structure

## <class 'pandas.core.frame.DataFrame'>

## Int64Index: 258626 entries, 0 to 258625

## Data columns (total 8 columns):

## IP 258626 non-null values

## Reliability 258626 non-null values

## Risk 258626 non-null values

## Type 258626 non-null values

## Country 248571 non-null values

## Locale 184556 non-null values

## Coords 258626 non-null values

## x 258626 non-null values

## dtypes: int64(2), object(6)

*# take a look at the first 10 rows*

av.head().to\_csv(sys.stdout)

## ,IP,Reliability,Risk,Type,Country,Locale,Coords,x

## 0,222.76.212.189,4,2,Scanning Host,CN,Xiamen,"24.4797992706,

## 118.08190155",11

## 1,222.76.212.185,4,2,Scanning Host,CN,Xiamen,"24.4797992706,

## 118.08190155",11

## 2,222.76.212.186,4,2,Scanning Host,CN,Xiamen,"24.4797992706,

## 118.08190155",11

## 3,5.34.246.67,6,3,Spamming,US,,"38.0,-97.0",12

## 4,178.94.97.176,4,5,Scanning Host,UA,Merefa,"49.8230018616,

## 36.0507011414",11

[[ I don’t get this output. Instead, I get this:

Out[4]: &ltclass 'pandas.core.frame.DataFrame'&gt Int64Index: 5 entries, 0 to 4 Data columns (total 8 columns):

IP 5 non-null values

Reliability 5 non-null values

Risk 5 non-null values

Type 5 non-null values

Country 5 non-null values

Locale 4 non-null values

Coords 5 non-null values

x 5 non-null values

dtypes: int64(2), object(6)

AR: FIXED! #ty

Within Canopy, IPython set of functions to output data to a more viewer-friendly HTML format can be used to make the head() output in Listing 3-5 much easier to read (see Figure 3-1).

AU: If you make adjustments above based on the TE’s comment, presumably you’ll also need to update figure 3-1.

Also, “head()” is not mentioned in the text of the chapter. Shouldn’t it be, given its mention in the following code title and the figure caption? –John

AR: Figure updated; Added mention of head()

IPython Code to Display head() as an HTML Table

Listing 3-6

*# require object: av (3-5)*

*# See corresponding output in Figure 3-1*

*# import the capability to display Python objects as formatted HTML*

from IPython.display import HTML

*# display the first 10 lines of the dataframe as formatted HTML*

HTML(av.head(10).to\_html())

[[Authors: All figs need a text reference--I added a simple one above. Kezia]]

[[Authors: You’ve just introduced “IPython Notebooks” but you haven’t explained what they are, nor explained how they appear in the Canopy console. The IPython Notebooks I’m familiar with operate inside of a web browser and have their own user interface and interaction patterns. For example, in IPython Notebooks, each “In[xxx]:” appears in it’s own cell, which can be highlighted, edited, and rexecuted, and also reordered. Cells can be merged, split, etc. ]]

AR: Canopy behaves a bit like IPython Notebooks, but on review I agree that this might be confusing for some folks, so I modified it

Figure 3-1 IPython HTML head() output [9781118793725 c03f001.png]

Exploring Data

Now that you have a general idea of the variables and how they look, it’s time to bring your security domain expertise into the mix to explore and discover what is interesting about the data. This will enable you to form good questions to ask and answer. Despite having almost 260,000 records, you have many tools at your disposal to help get a feel for what it contains.

Before going any deeper into the data, however, there are some tidbits of information you know about the data, so we will summarize them here:

* Reliability, Risk, and x are *integers.*
* IP, Type, Country, Locale, and Coords are *character strings.*
* The IP address is stored in the dotted-quad notation, not in hostnames or decimal format.
* Each record is associated with a unique IP address, so there are 258,626 IP addresses (in this download).

Each IP address has been geo-located into the latitude and longitude pair in the Coords field, but they are in a single field separated by a comma. You will have to parse that further if you want to use that field.

When you have quantitative variables (which is a fancy way to say “numbers representing a quantity”), a good first exploratory step is to look at the basic *descriptive statistics* on the variables. These are comprised of the following:

* *Minimum* and *maximum* values; taking the difference of these will give you the *range* (*range* = *max* - *min*)
* *Median* (the value at the middle of the data set)
* *First* and *third quartiles* (the 25th and 75th percentiles, or you could think of it as the median value of the first and last halves of the data, respectively)
* *Mean* (sum of all values divided by the number of count)

You may see the min, max, median, and quartiles referred to as the *five number summary* of a data set (as developed by Tukey), and both languages have built-in functions to calculate them—summary() in R and describe() in Python— along with the mean. Take a look at the summary on the two primary numeric columns: Reliability and Risk.

R Code to Look at the Central Tendency of Reliabilty and Risk

Listing 3-7

*# require object: av (3-4)*

summary(av$Reliability)

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

## 1.000 2.000 2.000 2.798 4.000 10.000

summary(av$Risk)

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

## 1.000 2.000 2.000 2.221 2.000 7.000

Python Code to Look at the Central Tendency of Reliabilty and Risk

AU: Should the text of the chapter discuss describe()?

\*\*\* Please keep this query in mind for all code that includes a function (or what have you) that then is not also mentioned in the text of the chapter, as appropriate. Thanks –John

AR: understood ; added reference

Listing 3-8

*# require object: av (3-5)*

av['Reliability'].describe()

## count 258626.000000

## mean 2.798040

## std 1.130419

## min 1.000000

## 25% 2.000000

## 50% 2.000000

## 75% 4.000000

## max 10.000000

## Length: 8, dtype: float64

av['Risk'].describe()

## count 258626.000000

## mean 2.221362

## std 0.531571

## min 1.000000

## 25% 2.000000

## 50% 2.000000

## 75% 2.000000

## max 7.000000

## Length: 8, dtype: float64

As you look at these results, note that the Reliability column spreads across the *documented* potential range of [1…10] (Slide 10 of http://www.slideshare.net/alienvault/building-an-ip-reputation-engine-tracking-the-miscreants)

[[ This is a good URL but it does not list or describe the potential range of the Reliability score. ]]

AR: It’s on slide 10. I updated the paragraph text before the URL ref

but the Risk column—which AlienVault says has a documented potential range of [1…10]—only has a spread of [1…7]. You can also see that both Risk and Reliability appear to center on a value of 2.

You can now dig a bit deeper and use the fact that the Reliability, Risk, Type, and Country fields can be used together to define data set categories. Even though we just treated Reliability and Risk as numbers, they actually are ordinal, meaning each entry is assigned an integer and a value of 4 is not necessarily twice the Reliability or Risk of 2. It only means that Reliability or Risk that is scored 4 are higher than those scored 2. In other words, the number is has more meaning as a label than a measurement. Categorical data may also be referred to as *nominal values*, *factors*, or in some cases, *qualitative variables*.

type="note"

Isn’t “data” just “data”?

You may be used to treating data holistically, thinking that the contents of a log file or database extract is just, well, *data*. If you’re used to working with data in spreadsheet form (like Microsoft Excel) you aren’t really encouraged to think of it any other way. Individual data elements can, however, be broken down into two broad categories: *quantitative* and *qualitative*. Quantitative data elements represent actual quantities whereas qualitative (or *categorical*) data elements are more descriptive in nature

TCP or UDP port numbers may be numeric, but they don’t actually represent a quantity; they are just parts of a category, in this case numerically named entities. Port “22” is not truly greater or less port “7070”. Conversely, “number of bytes transferred” or “number of infected hosts” represents actual quantities that can be compared numerically.

Categorical data is easily manipulated in R as factors and in Python as a pandas Categorical class. In fact, both R and Python have extensive functions that all you to group, split, extract and perform analysis on and with factors. You can see in Listing 3-4 that R made a correct educated guess that IP, Type, Country, and Locale were all categorical in nature as it scanned through the AlienVault IP reputation data file. Country names and malware types are easily identified as just classifications (*nominal* data in statistics terms). You can also see that R did *not* properly recognize that Reliability and Risk were both qualitative in nature. Even though there is a meaningful sequence to them—risk level “5” is greater than “1”—the numeric, *ordinal* arrangement is not expressing quantity (i.e. you should not try to calculate the mean of the Risk values or subtract one Risk value from another).

[[Authors: I wonder if you should have a separate paragraph explaining Nominal (Categorical), Ordinal, Interval, and Ratio scale data, and how both R and Python handle them. The current paragraph may be correct but it might also be confusing.]]

AR: made it a feature (but veered a bit from explicitly dealing with every nuance of categorical/quantative if that’s acceptable)

Within R, the difference between the two is automatically handled by the summary()function, and it displays the count for each category. This doesn’t work on the quantitative variables though. In order to get a count of those, you can use the table() function if there are not too many unique values in the variable. Within Python, you can create a short function that leverages pandas to convert a data frame column (which is just an array) into a very appropriately named Categorical object, which you can tweak a bit to give you similar helpful output.

[[Authors: We don’t use quotes around terms that have code font applied, unless the quotes are part of the actual code. Kezia]]

AR: Apologies, again. That was not clear at the start of all this. I’m afraid we created a great deal of additional work for you.

R Exploratory Code for AlienVault Data

Listing 3-9

*# require object: av (3-4)*

table(av$Reliability)

## 1 2 3 4 5 6 7 8 9

## 5612 149117 10892 87040 7 4758 297 21 686

## 10

## 196

table(av$Risk)

## 1 2 3 4 5 6 7

## 39 213852 33719 9588 1328 90 10

*# summary sorts by the counts by default*

*# maxsum sets how many factors to display*

summary(av$Type, maxsum=10)

## Scanning Host Malware Domain

## 234180 9274

## Malware IP Malicious Host

## 6470 3770

## Spamming C&C

## 3487 610

## Scanning Host;Malicious Host Malware Domain;Malware IP

## 215 173

## Malicious Host;Scanning Host (Other)

## 163 284

summary(av$Country, maxsum=40)

## CN US TR DE NL RU GB

## 68583 50387 13958 10055 9953 7931 6346 6293

## IN FR TW BR UA RO KR CA

## 5480 5449 4399 3811 3443 3274 3101 3051

## AR MX TH IT HK ES CL AE

## 3046 3039 2572 2448 2361 1929 1896 1827

## JP HU PL VE EG ID RS PK

## 1811 1636 1610 1589 1452 1378 1323 1309

## VN LV NO CZ BG SG IR (Other)

## 1203 1056 958 928 871 868 866 15136

Python Exploratory Code for AlienVault Data

Listing 3-10

*# require object: av (3-5)*

*# factor\_col(col)*

*#*

*# helper function to mimic R's "summary()" function*

*# for pandas "columns" (which are really just Python arrays)*

def factor\_col(col):

factor = pd.Categorical.from\_array(col)

return pd.value\_counts(factor,sort=True).reindex(factor.levels)

rel\_ct = pd.value\_counts(av['Reliability'])

risk\_ct = pd.value\_counts(av['Risk'])

type\_ct = pd.value\_counts(av['Type'])

country\_ct = pd.value\_counts(av['Country'])

print factor\_col(av['Reliability'])

## 1 5612

## 2 149117

## 3 10892

## 4 87040

## 5 7

## 6 4758

## 7 297

## 8 21

## 9 686

## 10 196

## Length: 10, dtype: int64

print factor\_col(av['Risk'])

## 1 39

## 2 213852

## 3 33719

## 4 9588

## 5 1328

## 6 90

## 7 10

## Length: 7, dtype: int64

print factor\_col(av['Type']).head(n=10)

## APT;Malware Domain 1

## C&C 610

## C&C;Malware Domain 31

## C&C;Malware IP 20

## C&C;Scanning Host 7

## Malicious Host 3770

## Malicious Host;Malware Domain 4

## Malicious Host;Malware IP 2

## Malicious Host;Scanning Host 163

## Malware Domain 9274

## Length: 10, dtype: int64

print factor\_col(av['Country']).head(n=10)

## A1 267

## A2 2

## AE 1827

## AL 4

## AM 6

## AN 3

## AO 256

## AR 3046

## AT 51

## AU 155

## Length: 10, dtype: int64

These numerical tables help you get a general view of the data, but a graph of the distribution of the data has the potential to provide a whole new perspective, often times giving insights that numbers alone cannot reveal. We start with a simple bar chart to get a very quick visual overview of the Country, Reliability, and Risk factors (see Figures 3-2 through 3-4, respectively). You’ll need to execute each R code listing individually to see each graph.

R Code for Visualizing Portions of AlienVault Data

[[Editor: I think each of these code snippets should be listed separately with the graph figure in between. Same for the Python code, below.]] //Authors, We can try to space it out that way if you want. There’s never a guarantee that the figures will land exactly where we place them, but we can certainly try to intersperse them with the code if you think that will help. Thanks, Kevin (PjE)

AR: Totally agree. I split them out that way

Listing 3-11

*# require object: av (3-4)*

*# We need to load the ggplot2 library to make the graphs*

*# See corresponding output in Figure 3-2*

*# NOTE: Graphing the data shows there are a number of entries without*

*# a corresponding country code, hence the blank entry*

library(ggplot2)

*# Bar graph of counts (sorted) by Country (top 20)*

*# get the top 20 countries' names*

country.top20 <- names(summary(av$Country))[1:20]

*# give ggplot a subset of our data (the top 20 countries)*

*# map the x value to a sorted count of country*

gg <- ggplot(data=subset(av,Country %in% country.top20),

aes(x=reorder(Country, Country, length)))

*# tell ggplot we want a bar chart*

gg <- gg + geom\_bar(fill="#000099")

*# ensure we have decent labels*

gg <- gg + labs(title="Country Counts", x="Country", y="Count")

*# rotate the chart to make this one more readable*

gg <- gg + coord\_flip()

*# remove "chart junk"*

gg <- gg + theme(panel.grid=element\_blank(),

panel.background=element\_blank())

*# display the image*

print(gg)

Figure 3-2 Country factor bar chart (R) [9781118793725 c03f002.eps]

Listing 3-12

*# requires packages: ggplot2*

*# require object: av (3-4)*

*# See corresponding output in Figure 3-3*

*# Bar graph of counts by Risk*

gg <- ggplot(data=av, aes(x=Risk))

gg <- gg + geom\_bar(fill="#000099")

*# force an X scale to be just the limits of the data*

*# and to be discrete vs continuous*

gg <- gg + scale\_x\_discrete(limits=seq(max(av$Risk)))

gg <- gg + labs(title="'Risk' Counts", x="Risk Score", y="Count")

gg <- gg + theme(panel.grid=element\_blank(),

panel.background=element\_blank())

print(gg)

Figure 3-3 Risk factor bar chart (R) [9781118793725 c03f003.eps]

[[ I get the following error message for the last two lines:

> g <- g + scale\_x\_discrete(limits=seq(max(av$Risk))

+ g <- g + labs(title="'Risk' Counts", x="Risk Score")

Error: unexpected symbol in:

"g <- g + scale\_x\_discrete(limits=seq(max(av$Risk))

g"

> print(g)

stat\_bin: binwidth defaulted to range/30. Use 'binwidth = x' to adjust this.

I don’t know what is causing the errors.

AR: I missed a right parenthesis in the previous iteration

Listing 3-13

*# requires packages: ggplot2*

*# require object: av (3-4)*

*# See corresponding output in Figure 3-4*

*# Bar graph of counts by Reliability*

gg <- ggplot(data=av, aes(x=Reliability))

gg <- gg + geom\_bar(fill="#000099")

gg <- gg + scale\_x\_discrete(limits=seq(max(av$Reliability)))

gg <- gg + labs(title="'Reliabiity' Counts", x="Reliability Score",

y="Count")

gg <- gg + theme(panel.grid=element\_blank(),

panel.background=element\_blank())

print(gg)

[[Authors: Figures need text references before they appear. See my additions above. Kezia]]

AU: To verify, is it correct that there is no country code for the 4th entry in figure 3.2? –John

AR: Correct. I noted it in the comments of the code that generate the figure

Figure 3-4 Reliability factor bar chart (R) [9781118793725 c03f004.eps]

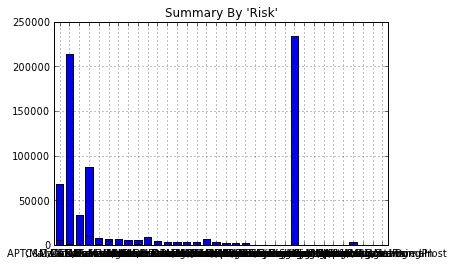
[AU: Now that we know we’re going to be a color book, can any of these figures take more advantage of color? If so, redo and resupply them. Thanks, Kevin (PJE)]

AR: Aye. Re-generated all of the graphics

The Python version uses

Python Code for Visualizing Portions of AlienVault Data

running this as a whole does not work correctly. It seems to plot them all together in the same plot. Here’s what I get:



Listing 3-14

*# require object: av (3-5), factor\_col (3-10)*

*# See corresponding output in Figure 3-5*

*# NOTE: Notice the significant differnce in the Python graph in that the*

*# blank/empty country code entries are not in the graph*

*# need some functions from matplotlib to help reduce 'chart junk'*

import matplotlib.pyplot as plt

*# sort by country*

country\_ct = pd.value\_counts(av['Country'])

*# plot the data*

plt.axes(frameon=0) *# reduce chart junk*

country\_ct[:20].plot(kind='bar',

rot=0, title="Summary By Country", figsize=(8,5)).grid(False)

Figure 3-5 Country factor bar chart (Python*)* [9781118793725 c03f005.png]

[[ break snippet ]]

Listing 3-15

*# require object: av (3-5), factor\_col (3-10)*

*# See corresponding output in Figure 3-6*

plt.axes(frameon=0) *# reduce chart junk*

factor\_col(av['Reliability']).plot(kind='bar', rot=0,

title="Summary By 'Reliability'", figsize=(8,5)).grid(False)

Figure 3-6 Reliability factor bar chart (Python) [793725c03f006.png]

[[ break snippet ]]

Listing 3-16

*# require object: av (3-5), factor\_col (3-10)*

*# See corresponding output in Figure 3-7*

plt.axes(frameon=0) *# reduce chart junk*

factor\_col(av['Risk']).plot(kind='bar', rot=0,

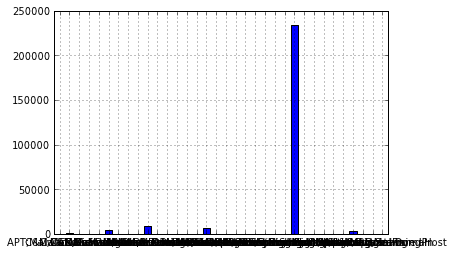
title="Summary By 'Risk'", figsize=(8,5)).grid(False)

Figure 3-7 Risk factor bar chart (Python) [9781118793725 c03f007.png]

[[ break snippet ]]

[[ I don’t get the same result as Figure 3.7. Here’s what I get instead:]]

AR: Fixed



[[Authors: Please add text reference above for these three figs. Kezia]]

AR: Done in the listing comments

The Country chart, as shown in Figure 3-5, shows there are definitely some countries that are contributing more significantly to the number of malicious nodes, and you can go back to numbers for a moment to look at the percentages for the top ten in the list:

R Code Comparing Country Percentage Makeup

[[Authors: would it read better to change “Compare” above to Comparing”? Kezia]]

AR: Agreed. Change made.

Listing 3-17

*# require object: av (3-4)*

country10 <- summary(av$Country, maxsum=10)

*# now convert to a percentage by dividing by number of rows*

country.perc10 <- country10/nrow(av)

*# and print it*

print(country.perc10)

## CN US TR DE NL

## 0.26518215 0.19482573 0.05396983 0.03887854 0.03848414 0.03066590

## RU GB IN (Other)

## 0.02453736 0.02433243 0.02118890 0.30793501

Python Code Comparing Country Percentage Makeup

[[Authors: would it read better to change “Compare” above to Comparing”? Kezia]]

AR: Agreed. Change made.

Listing 3-18

*# require object: av (3-5)*

*# extract the top 10 most prevalent countries*

top10 = pd.value\_counts(av['Country'])[0:9]

*# calculate the % for each of the top 10*

top10.astype(float) / len(av['Country'])

## CN 0.265182

## US 0.194826

## TR 0.053970

## DE 0.038484

## NL 0.030666

## RU 0.024537

## GB 0.024332

## IN 0.021189

## FR 0.021069

## Length: 9, dtype: float64

[[ I get different results – only slightly different, but even a little difference is disturbing:

CN 0.265179

US 0.194826

TR 0.053970

DE 0.038484

NL 0.030666

RU 0.024537

GB 0.024333

IN 0.021189

FR 0.021069

dtype: float64

AR: Results were most likely due to a different reputation.data file being used.

These quick calculations show that China and the United States together account for almost 46 percent of the malicious nodes in the list, and Russia accounts for just 2.4 percent. One avenue to explore here is to see how this compares with various industry reports since you would expect many of these countries to be in the top ten. However, the amount that some countries contribute suggest that there might be some bias in the data set. You can also see that 3 percent of the nodes cannot be geo-located (in the R output, (Other) category).

type="note"

Chapter 5 covers the challenges and pitfalls of IP address geo-location, so we’ll refrain from exploring that further here.

AU: My TOC has chapter 4 as “Analyzing Badness.” Please confirm. --John //Authors, I checked Ch. 4 and the reference looks correct to me, but please confirm. Thanks, Kevin (PJE)

AR: Changed reference to Chapter 5 #ty

Looking at the Risk variable, you can see that the level of risk of most of the nodes is *negligible* (i.e. so low that they can be disregarded). There are other elements that stand out with this data though, foremost being that practically no endpoints are in categories 1, 5, 6, or 7, and none in the rest of the defined possible range [8-10]. This anomaly is a sign to you that it is worth digging a bit deeper, but the anomaly is significant evidence of bias in the data set.

Finally, the Reliability rating of the nodes also appears to be a bit skewed (i.e. the distribution is extended to one side of the mean or central tendency). The values are mostly clustered in levels 2 and 4, with not many ratings above level 4. The fact that it completely skips a reliability rating of 3 should raise some questions in your mind. It could indicate a systemic flaw in the assignment of the rating, or it could be that you have at least two distinct data sets. Either way, that large quantity of 2s and 4s and low quantity of 3s is a clear sign that you should investigate further, because it’s just a little odd and surprising.

[[Authors: Edits to last sentence above okay? Kezia]]

AR: Yes.

You now have some leads to pursue and a much better idea of the makeup of the key components of the data. This preliminary analysis gives you enough information to formulate a research question.

Homing In on a Question

Consider both the problem and the primary use case for the AlienVault reputation data: importing it into a SEIM or Intrusion Detection System/Intrusion Prevention System (IDS/IPS) to alert incident response team members or to log/block malicious activity. How can this quick overview of the reputation data influence the configuration of the SIEM in this setting to ensure that the least number of “trivial” alerts are generated?

Let’s take a slightly more practical view of those questions by asking, “which nodes from the reputation database represent a potentially real threat?”

[[Authors: I don’t know why this view is more “deterministic”. Are you saying “Does the reputation database provide solid evidence regarding the relative risk of each node for purposes of prioritization?” And why introduce “threat” here?

AR: Changed ‘deterministic’ to ‘practical’; Introducing ‘threat’ to start shifting the exercise into the security realm and going beyond just looking at basic data statistics. Since it’s a security book, a ‘threat’ is a good question to ask about, esp with this data.

There *is* a reason AlienVault included both Risk and Reliability fields, and you should be able to use these attributes to classify nodes into two categories: 1) the nodes you really care about and 2) everything else. The definition of “really care about” can be somewhat subjective, but it is unrealistic to believe you would want to generate an alert on all detected activity by one of these 258,626 nodes. Some form of prioritization triage and prioritization *must* occur and it is a far better approach to base the triage and prioritization on statistical analysis of data and evidence rather than a “gut call” or solely on “expert opinion” alone.

It’s possible to see which nodes should garner our attention by comparing the Risk and Reliability factors. To do this, we use a *contingency table*, which is a tabular view of the multivariate frequency distribution of specific variables. In other words, a contingency table helps show relationships between two variables. After building a contingency table, you can take both a numeric and graphical look at the results to see where the AlienVault nodes “cluster”.

The output from the R code in Listing 3-19 is Figure 3-8 which shows the output of the contingency table as a level plot and uses size and color to show quantity, whereas the Python code in Figure 3-9 is used to generate a standard heat map that relies on color alone to show quantity. (A heat map is a graphical representation of data where the individual values contained in a matrix are represented as colors. <http://en.wikipedia.org/wiki/Heat_map>) With both factors combined, it is very apparent that the values in this data set bias are concentrated around [2, 2], which might be a sign of bias.

[[Authors: Edits above okay? Kezia]]

AU: Please add a more specific intro to this code. (It may be implied, but I think it could be stated more directly.)

Also, I’ve moved the figures to below so that they follow their references (SOP). –John

AR: tweaked the wording; move paragraph up before code & plots

R Code for Risk/Reliability Contingency Table Generation

Listing 3-19

*# require object: av (3-4)*

*# See corresponding output in Figure 3-8*

*# compute contingency table for Risk/Reliability factors which*

*# produces a matrix of counts of rows that have attributes at*

*# each (x, y) location*

rr.tab <- xtabs(~Risk+Reliability, data=av)

ftable(rr.tab) # print table

## virtually identical output to pandas (See Listing 3-20)

*# graphical view of levelplot*

*# need to use levelplot function from lattice package*

library(lattice)

*# cast the table into a data frame*

rr.df = data.frame(table(av$Risk, av$Reliability))

*# set the column names since table uses "Var1" and "Var2"*

colnames(rr.df) <- c("Risk", "Reliability", "Freq")

*# now create a level plot with readable labels*

levelplot(Freq~Risk\*Reliability, data=rr.df, main="Risk ~ Reliabilty",

ylab="Reliability", xlab = "Risk", shrink = c(0.5, 1),

col.regions = colorRampPalette(c("#F5F5F5", "#01665E"))(20))

Python Code for Risk/Reliability Contingency Table Generation

Listing 3-20

*# require object: av (3-5)*

*# See corresponding output in Figure 3-9*

*# compute contingency table for Risk/Reliability factors which*

*# produces a matrix of counts of rows that have attributes at*

*# each (x, y) location*

*# need cm for basic colors*

*# need arange to modify axes display*

*from matplotlib import cm*

from numpy import arange

pd.crosstab(av['Risk'], av['Reliability'])

## Reliability 1 2 3 4 5 6 7 8 9 10

## Risk

## 1 0 0 16 7 0 8 8 0 0 0

## 2 804 149114 3670 57653 4 2084 85 11 345 82

## 3 2225 3 6668 22168 2 2151 156 7 260 79

## 4 2129 0 481 6447 0 404 43 2 58 24

## 5 432 0 55 700 1 103 5 1 20 11

## 6 19 0 2 60 0 8 0 0 1 0

## 7 3 0 0 5 0 0 0 0 2 0

*# graphical view of contingency table (swapping risk/reliability)*

xtab = pd.crosstab(av['Reliability'], av['Risk'])

plt.pcolor(xtab,cmap=cm.Greens)

plt.yticks(arange(0.5,len(xtab.index), 1),xtab.index)

plt.xticks(arange(0.5,len(xtab.columns), 1),xtab.columns)

plt.colorbar()

Figure 3-9 Risk/reliability contingency table heat map (Python) [9781118793725 c03f009.png]

[[Authors: Figures 3.8 and 3.9 need text references. Kezia]]

type="general"

As a fun aside, you can determine whether the patterns you’re seeing are occurring by chance, or whether there is some underlying meaning to them. Although you could do some fancy-pants statistics here and maybe apply Fisher’s exact test, you don’t need to get crazy. What if you assumed that every value of Risk and Reliability had an equal chance of occurring? What would the level plot look like? You should expect some amount of natural variation—both in the systems and the data collection process—so some combinations would naturally occur more often than others. But how different would it look from the current data?

[[Authors: Edits above make sense? Sentence missing at the end of the para above? I assumed not, and deleted the “In”? Kezia]]

AU: I’ve converted the preceding paragraph (“aside”) to be sidebar note. Please confirm, as this assumes the following paragraph continues from the paragraph above this note (the paragraph ending in “which might be a sign of bias”). –John

AR: I converted it to general as it’s not 100% necessary to the flow of section but has important information to convey. It’s definitely not a sidebar (at least if I understand that that is) since it’s has multiple paragraphs and a graphic to go with it.

You can use the sample() function to generate random samples from a Uniform distribution [1,7] and [1,10] and then build a contingency table from those random samples. Running this multiple times should produce a different set of random tables each time. Each run is called a *realization* of the random processes.

The R code in Listing 3-21 produces the levelplot in Figure 3-10 and shows two things. First, you can make some pretty and colorful random boxes with a few lines of code. Second, there is definitely something pulling nodes into the lower Riskand Reliability categories (i.e., toward zero for each). It could be because the world just has low risk and reliability or the sampling method or scoring system is introducing the skew.

R Code to Generate Baseline “Random” Sample for Contingency Table Comparison

Listing 3-21

*# require object: av (3-4), lattice (3-19)*

*# See corresponding output in Figure 3-10*

*# generate random samples for risk & reliability and re-run xtab*

*# starting PRNG from reproducable point*

set.seed(1492) # as it leads to discovery

*# generate 260,000 random samples*

rel=sample(1:7, 260000, replace=T)

rsk=sample(1:10, 260000, replace=T)

*# cast table into data frame*

tmp.df = data.frame(table(factor(rsk), factor(rel)))

colnames(tmp.df) <- c("Risk", "Reliability", "Freq")

levelplot(Freq~Reliability\*Risk, data=tmp.df, main="Risk ~ Reliabilty",

ylab="Reliability", xlab = "Risk", shrink = c(0.5, 1),

col.regions = colorRampPalette(c("#F5F5F5", "#01665E"))(20))

AU: Although the figure captions specify R and Python, I’d suggest revisiting the text above to clarify that 3-10 shows the R table and 3-11 shows the Python table. –John

AR: Python code removed, as the redundancy was not necessary in a “feature”

Figure 3-10 “Unbiased” risk/reliability contingency table (R) [9781118793725 c03f010.eps]

[[Authors: Should figure 3.11 have labels on the axes like the other figures do? Kezia]] //Author, Please redo and resubmit if so. Thanks, Kevin (PjE)

AR: Pulling it out as a feature is cool, but the code and graphic go with it. I formatted it the best as I know how in the Wiley template. There is no corresponding FeatureCodeListing and FeatureCodeHead so I’m not sure what to do there.

Now turn your attention to the Type variable to see if you can’t establish a relationship with the Risk and Reliability ratings. Looking closely at the Type variable, you notice that some entries have more than type assigned to them and they are separated by a semicolon (there are 215 Scanning Host;Malicious Host values, for example). Since you want to see how those types compare, those with a combination of types shouldn’t be mixed with other types. So, rather than try to parse out the nodes with multiple types, you can just reassign all of them into a category of Multiples to show that they were assigned more than one type. Then you can create a three-way contingency table and see how that looks. Pull in the Type column and see how that impacts the view.

The R code in Listing 3-22 produces the three-way contingency table lattice graph in Figure 3-11, enabling you to visually compare the amount of impact Type has on the Risk and Reliability classifications. The Python code in 3-23 also computes the three-way contingency table but shows an alternate output representation in a simple bar chart.

R Code to Generate a Three-Way Risk/Reliability/Type Contingency Table

Listing 3-22

*# require object: av (3-4), lattice (3-19)*

*# See corresponding output in Figure 3-11*

*# Create a new varible called "simpletype"*

*# replacing mutiple categories with label of "Multiples"*

av$simpletype <- as.character(av$Type)

*# Group all nodes with mutiple categories into a new category*

av$simpletype[grep(';', av$simpletype)] <- "Multiples"

*# Turn it into a factor again*

av$simpletype <- factor(av$simpletype)

rrt.df = data.frame(table(av$Risk, av$Reliability, av$simpletype))

colnames(rrt.df) <- c("Risk", "Reliability", "simpletype", "Freq")

levelplot(Freq ~ Reliability\*Risk|simpletype, data =rrt.df,

main="Risk ~ Reliabilty | Type", ylab = "Risk",

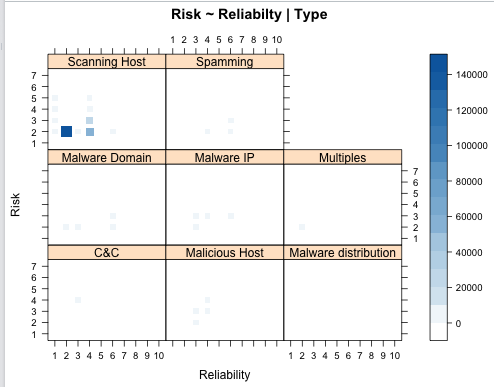
xlab = "Reliability", shrink = c(0.5, 1),

col.regions = colorRampPalette(c("#F5F5F5","#01665E"))(20))

Figure 3-11 Three-way risk/reliability/type contingency table (R) [9781118793725 c03f011.eps]

[[ I don’t get he same result as Figure 3.12. I don’t know if this is an error or just a side-effect of randomrealization. Here’s what I get:]]

AR: corrected; they shld be identical if you re-run; TO ALL: Figures were re-numbered due to removing one



Python Code to Generate a Three-Way Risk/Reliability/Type Contingency Table

Listing 3-23

*# require object: av (3-5)*

*# See corresponding output in Figure 3-12*

*# compute contingency table for Risk/Reliability factors which*

*# produces a matrix of counts of rows that have attributes at*

*# create new column as a copy of Type column*

av['newtype'] = av['Type']

*# replace multi-Type entries with Multiples*

av[av['newtype'].str.contains(";")] = "Multiples"

*# setup new crosstab structures*

typ = av['newtype']

rel = av['Reliability']

rsk = av['Risk']

*# compue crosstab making it split on the*

*# new type column*

xtab = pd.crosstab(typ, [ rel, rsk ],

rownames=['typ'], colnames=['rel', 'rsk'])

*# the following print statement will show a huge text*

*# representation of the contingency table. The output*

*# is too large for the book, but is worth looking at*

*# as you run through the exercise to see how useful*

*# visualizations can be over raw text/numeric output*

print xtab.to\_string() #output not shown

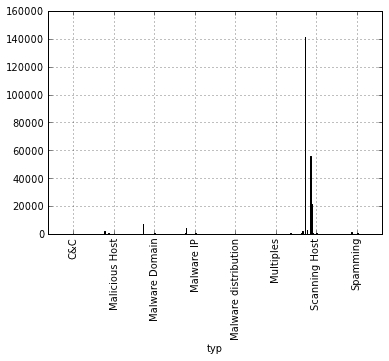
xtab.plot(kind='bar',legend=False,

title="Risk ~ Reliabilty | Type").grid(False)

Figure 3-12 Three-way risk/reliability/type contingency table bar chart (Python) [9781118793725 c03f012.eps]

[[I don’t get the results shown in Fig 3.12. I get a table full of data, and then one graphic. Here’s the graphic:

AR: fixed



[[Authors: this figure needs a text reference. Kezia]] //Authors, Yes, you need some text that indicates what the readers are seeing in the figure, that introduces it for them. It should be somewhere prior to the figure. Thanks, Kevin (PjE)

AU: Do we need the Python version of this figure? –John

AR: The R version and Python version are different to show different output versions of the 3-way table without being overly redundant in the code . Folks should also be used to generating similar graphics in each.

They say a picture is worth a thousand words, but in this case it’s worth about 234,000 data points in the Scanning Hosts category (about 90 percent of the entries are classified as scanning hosts). That category is so large and generally low risk that it is overshadowing the rest of the categories. Remove it from the Type factors and regenerate the image. This isn’t to say the Scanning Hosts category isn’t important, but remember you are trying to understand which of these entries you really care about. Nodes with low risk and reliability ratings are things you don’t want to be woken up from your nap for. You want to peel those away and look at the relationships that exist underneath the scanning hosts. We continue the examples from Listings 3-22 and 3-23 and generate new corresponding Figures 3-13 (R lattice) and 3-14 (Python bar chart) in Listings 3-24 and 3-35.

R Code to Filter Out “Scanning Host” Type

Listing 3-24

*# require object: av (3-4), lattice (3-19)*

*# See corresponding output in Figure 3-13*

*# from the existing rrt.df, filter out 'Scanning Host'*

rrt.df <- subset(rrt.df, simpletype != "Scanning Host")

levelplot(Freq ~ Reliability\*Risk|simpletype, data =rrt.df,

main="Risk ~ Reliabilty | Type", ylab = "Risk",

xlab = "Reliability", shrink = c(0.5, 1),

col.regions = colorRampPalette(c("#F5F5F5","#01665E"))(20))

Figure 3-13 Three-way risk/reliability/type contingency table without “Scanning Host” (R) [9781118793725 c03f013.eps]

Python Code to Filter Out “Scanning Host” Type

Listing 3-25

*# require object: av (3-5)*

*# See corresponding output in Figure 3-14*

*# filter out all "Scanning Hosts"*

rrt\_df = av[av['newtype'] != "Scanning Host"]

typ = rrt\_df['newtype']

rel = rrt\_df['Reliability']

rsk = rrt\_df['Risk']

xtab = pd.crosstab(typ, [ rel, rsk ],

rownames=['typ'], colnames=['rel', 'rsk'])

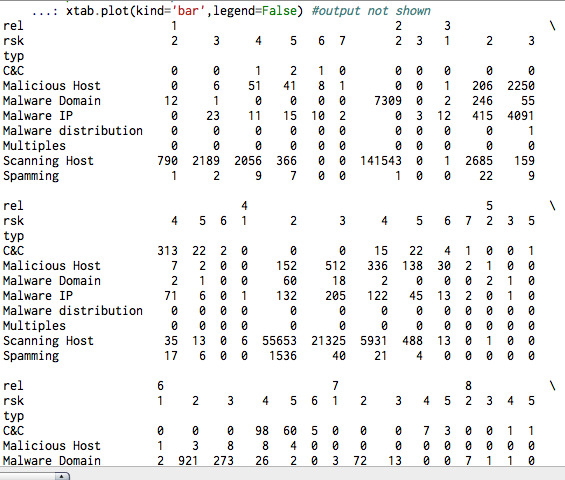
xtab.plot(kind='bar',legend=False,

title="Risk ~ Reliabilty | Type").grid(False)

Figure 3-14 Three-way risk/reliability/type contingency table bar chart without “Scanning Host” (Python) [9781118793725 c03f014.eps]

I don’t get the result shown in Fig. 3-13. Instead, I get this result instead (plus a graphic bar chart at the end) ).

AR: Fixed



[[Authors: this fig needs a text reference. Kezia]] //Authors, Yes, again, you need some text that indicates what the readers are seeing in the figure, that introduces it for them. It should be somewhere prior to the figure. Thanks, Kevin (PjE)

AU: Do we need the Python version of this figure? –John

AR: no. just continuing what was setup previously with the two different ways to look at the data.

Now you are getting somewhere. In Figure 3-13, you can see the Malware domain type has risk ratings limited to 2s and 3s, and the reliability is focused around 2, but spreads the range of values. You can also start to see the patterns in the other categories as well even in Figure 3-14, but it's time to regenerate the graphics once more after you remove the Malware domain. Also, it looks like Malware distribution does not seem to be contributing any risk, so you can filter that factor out of the remaining types as well to get the final results in Figure 3-15 (R lattice plot) and Figure 3-16 (Python bar chart).

R Code to Filter Out Remaining Types

Listing 3-26

*# require object: av (3-4), lattice (3-19), rrt.df (3-24)*

*# See corresponding output in Figure 3-15*

rrt.df = subset(rrt.df,

!(simpletype %in% c("Malware distribution",

"Malware Domain")))

sprintf("Count: %d; Percent: %2.1f%%",

sum(rrt.df$Freq),

100\*sum(rrt.df$Freq)/nrow(av))

## [1] Count: 15171; Percent: 5.9%

levelplot(Freq ~ Reliability\*Risk|simpletype, data =rrt.df,

main="Risk ~ Reliabilty | Type", ylab = "Risk",

xlab = "Reliability", shrink = c(0.5, 1),

col.regions = colorRampPalette(c("#F5F5F5","#01665E"))(20))

Figure 3-15 3-Way risk/reliability/type contingency table — final (R) [9781118793725 c03f015.eps]

Python Code to Filter Out Remaining Types

Listing 3-27

*# require object: av (3-5), rrt\_df (3-25)*

*# See corresponding output in Figure 3-16*

rrt\_df = rrt\_df[rrt\_df['newtype'] != "Malware distribution" ]

rrt\_df = rrt\_df[rrt\_df['newtype'] != "Malware Domain" ]

typ = rrt\_df['newtype']

rel = rrt\_df['Reliability']

rsk = rrt\_df['Risk']

xtab = pd.crosstab(typ, [ rel, rsk ],

rownames=['typ'], colnames=['rel', 'rsk'])

print "Count: %d; Percent: %2.1f%%" % (len(rrt\_df), (float(len(rrt\_df))

/ len(av)) \* 100)

## Count: 15171; Percent: 5.9%

6Python6

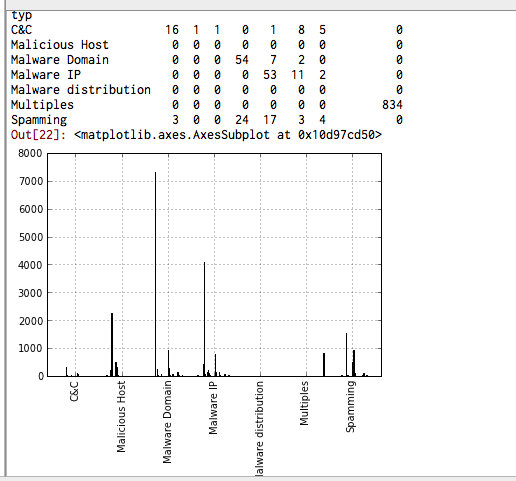
[[I get this error message:

File "<ipython-input-26-5d24c677430c>", line 9 print "Count: %d; Percent: %2.1f%%" %   
 ^ SyntaxError: invalid syntax

AR: re-wrapped the offending line to Python won’t complain

[[ I get different results from 3.14. I get this (bottom):]]

AR: everything should be the same now



With this final bit of filtering, you’ve reduced the list to less than 6 percent of the original and have honed in fairly well on the nodes representing the ones you really should care about. If you wanted to further reduce the scope, you could filter by various combinations of Reliability and/or Risk. Perhaps you want to go back to the categories you filtered out and bring a subset of those back in.

[[“Command and Control” is not code, but a data value.

AR: removed the reference as well

AR: correct.

AU: Do we need the Python version of this figure? –John

AR: yes. And it’s been included.

The rather simple parsing and slicing done here doesn’t show which variables are most important; it simply helps you understand the relationships and the frequency with which they occur. Just because 90 percent of the data was Scanning Hosts, perhaps you only want to filter those hosts with a risk of 2 or below. This analysis has merely helped you identify a set of nodes on which you can generate higher priority alerts. You can still capture the other types into a lower priority or into an informational log.

Since AlienVault updates this list hourly, you can create a script to do this filtering before importing new revisions into your security tools. You can then keep track of the percentage of nodes filtered out as a flag for the need to potentially readjust the rules. Furthermore, you should strongly consider performing this exploratory analysis on a semi-frequent basis. This will help you determine whether you need to re-think your perspective on what constitutes non-trivial nodes.

[[Authors: Edits above okay? Kezia]]

AR: yep. Thx

Summary

This chapter introduced the core structure and concepts of data analyses in Python and R. It incorporated basic statistics, foundational scripting/analysis patterns, and introductory visualizations to help you ask and answer a pertinent question. In addition, each example demonstrated the similarity of Python (with pandas) and R coding techniques and generated output. The steps presented are just one direction this particular analysis could lead. Every situation is different and will require you to pull in different tools and techniques as needed.

Future chapters focus mainly on R code, with some Python sprinkled in on occasion. If you are familiar with Python/pandas, the previous examples should help you translate between the two languages. If you are new to both R and Python, the standardization of future examples in one language should help you follow along with less confusion and help you learn R a bit better.

Recommended Reading

*Statistics and Data with R: An Applied Approach Through Examples* by Yosef Cohen and Jeremiah Y. Cohen (John Wiley & Sons, Ltd. ISBN: 9-780-470758052)

*Python for Data Analysis* by Wes McKinney (O’Reilly Media, Inc. ISBN: 9-798-1-4493-1979-3)

[[Supporting files:

ch03-jay.R

runs correctly (all output generated correctly) except for this line:

> setwd("~/Dropbox/datavizbook/chapters/ch03")

Error in setwd("~/Dropbox/datavizbook/chapters/ch03") :

cannot change working directory

Same error in the file ch03.R

The file ch03.py doesn’t run properly within IPython. I get the error:

File "<ipython-input-28-57c342698104>", line 121

print "Count: %d; Percent: %2.1f%%" % ^

SyntaxError: invalid syntax^

AR: Both files should run fine now when snippets are executed as instructed.