27th ANNUAL FIRST CONFERENCE BERLIN

14-19 JUNE 2015

UNIFIED SECURITY: IMPROVING THE FUTURE

Incident Response Programming with R

Eric Zielinski
Sr. Consultant, Nationwide

About Me?

- Cyber Defender for Nationwide
- Over 15 years in Information Security
- Speaker at various conferences FIRST, CEIC, FS-ISAC etc.
- Focus on blue team activities such as Forensics, Incident Response, and Data Exfiltration
- 4th most punctual guy I know

Agenda

- Why R?
- Overview of R
- Reading data sets
- Case Study
- Extending R with packages

Disclaimer

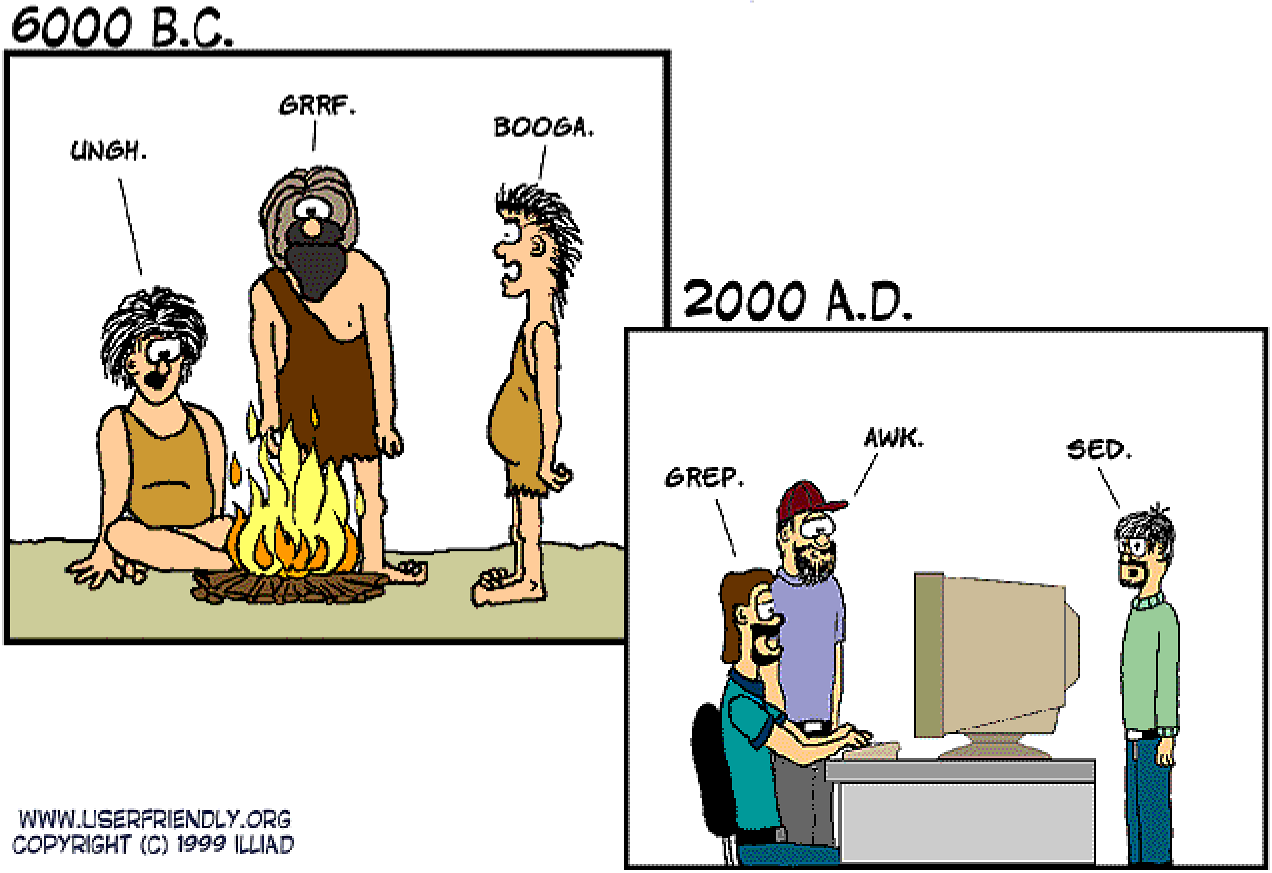
This presentation will not teach you how to become an expert programmer in R in under 45min

So What Will This Teach Me?

- How we can use data analytics to speed up our response and for post lessons learned
- How we should leverage programming languages more often in incident response
- How we can develop our own tools and analytics
- This is not trying to replace your current practices. Just simply giving you another tool in your toolbox, it’s really up to you on how you use it.

Frequently Asked Questions

Evolution



WWW.USERFRIENDLY.ORG
COPYRIGHT (C) 1999 ILLIAD

Issues…

- Incident response has been very *nix focused for years. This is not a bad thing, *nix rocks!
- The problem is that we are just not that good at detecting incidents
- So how can we get better?
  - Do we need to speed up response times?
  - Do we need better tools?
  - Do we need better talent?
  - Do we need more skills?
- So for IR there must be a different way, right?
- We must change our ways of thinking and try something new!

The good news

- Often times we are dealing with the same data sets
  - We see a lot of the same log files, config files, data sets, etc…
  - Shouldn’t we be able to streamline these?

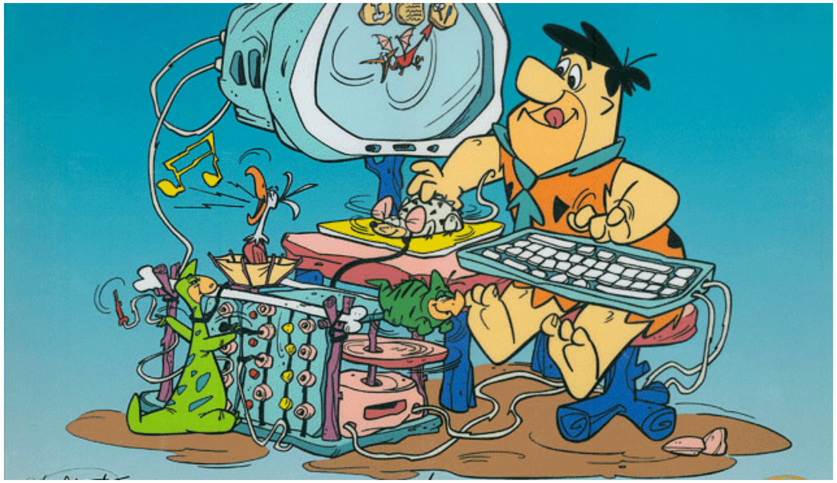
- What if we take more time to understand the data so future responses can be faster!
  - Think post incident work!!

Incident Response

Neanderthal method

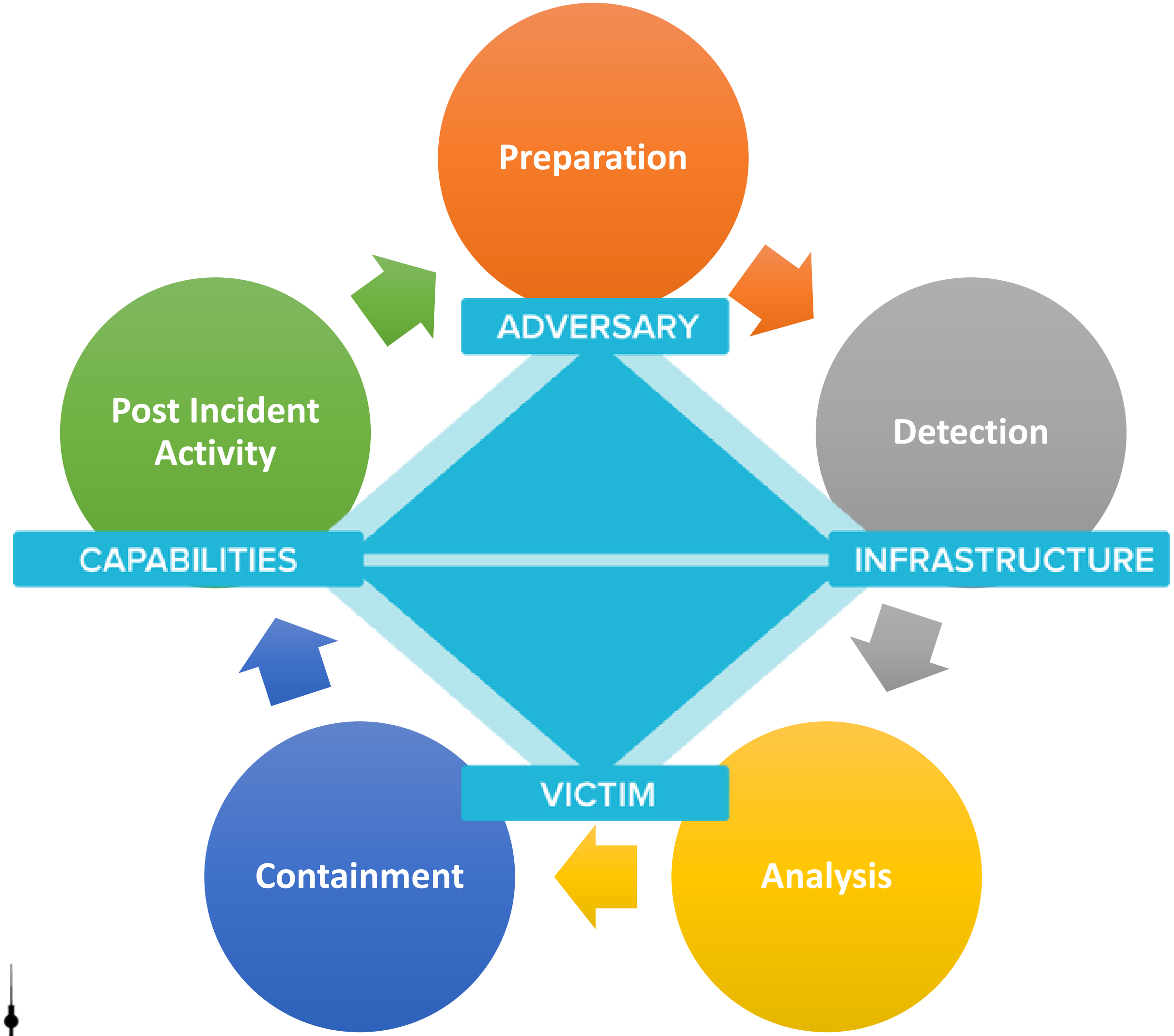
Bang on keyboard and mouse until you find something

Sophisticated Neanderthal Method



Let the data work for you, organize your data, combine, analyze, and respond

Lifecycle


Preparation
ADVERSARY
Post Incident Activity
Detection
CAPABILITIES
INFRASTRUCTURE
VICTIM
Containment
Analysis

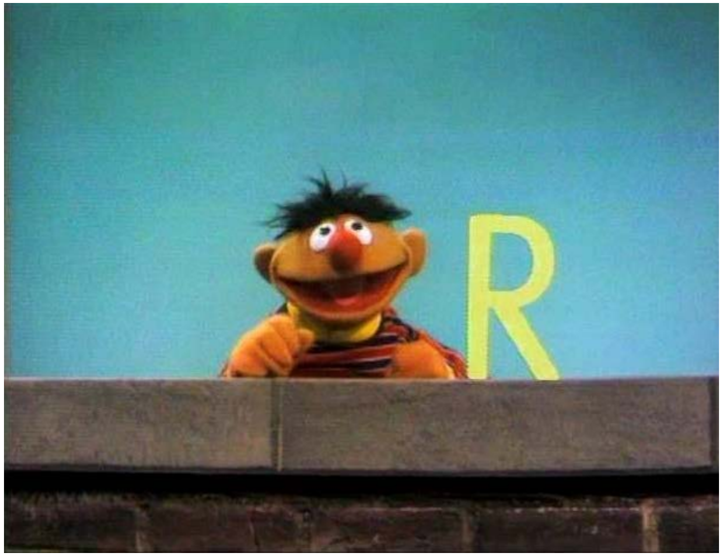
Post Incident Evolution

- Analysts often spend over 80% of their time preparing and exploring data sets before they begin more formal analysis work

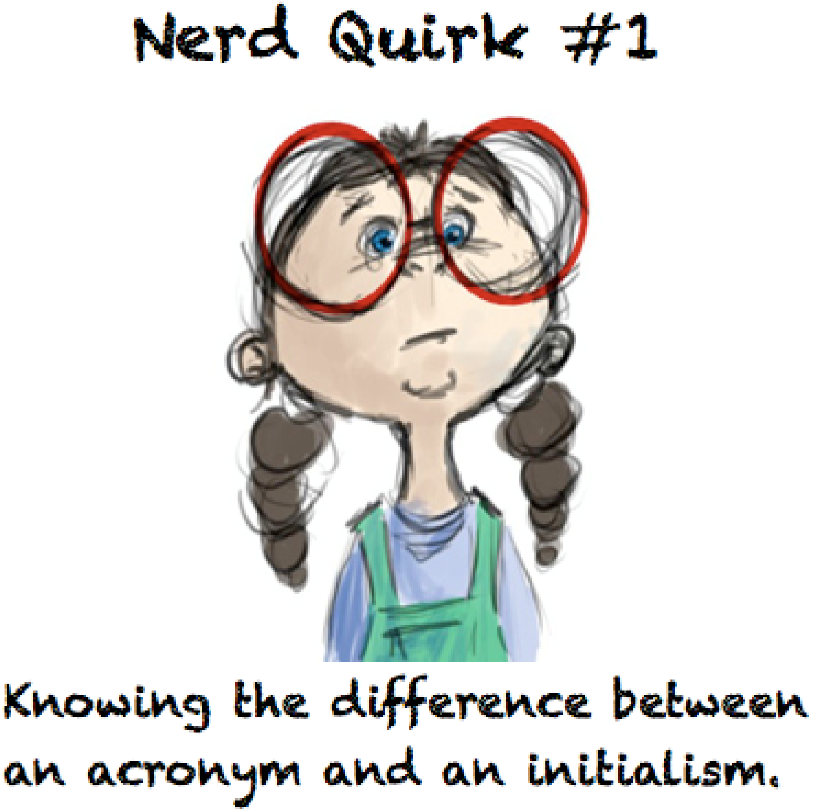
e436064 [RM] (c) www.visualphotos.com

Why R?

- R runs quickly
- It’s intuitive
- Vectorized programming
- It’s interactive!
  - View(Logs)



Quirks


Nerd Quirk #1
Knowing the difference between an acronym and an initialism.

Where to begin?

- Installing R

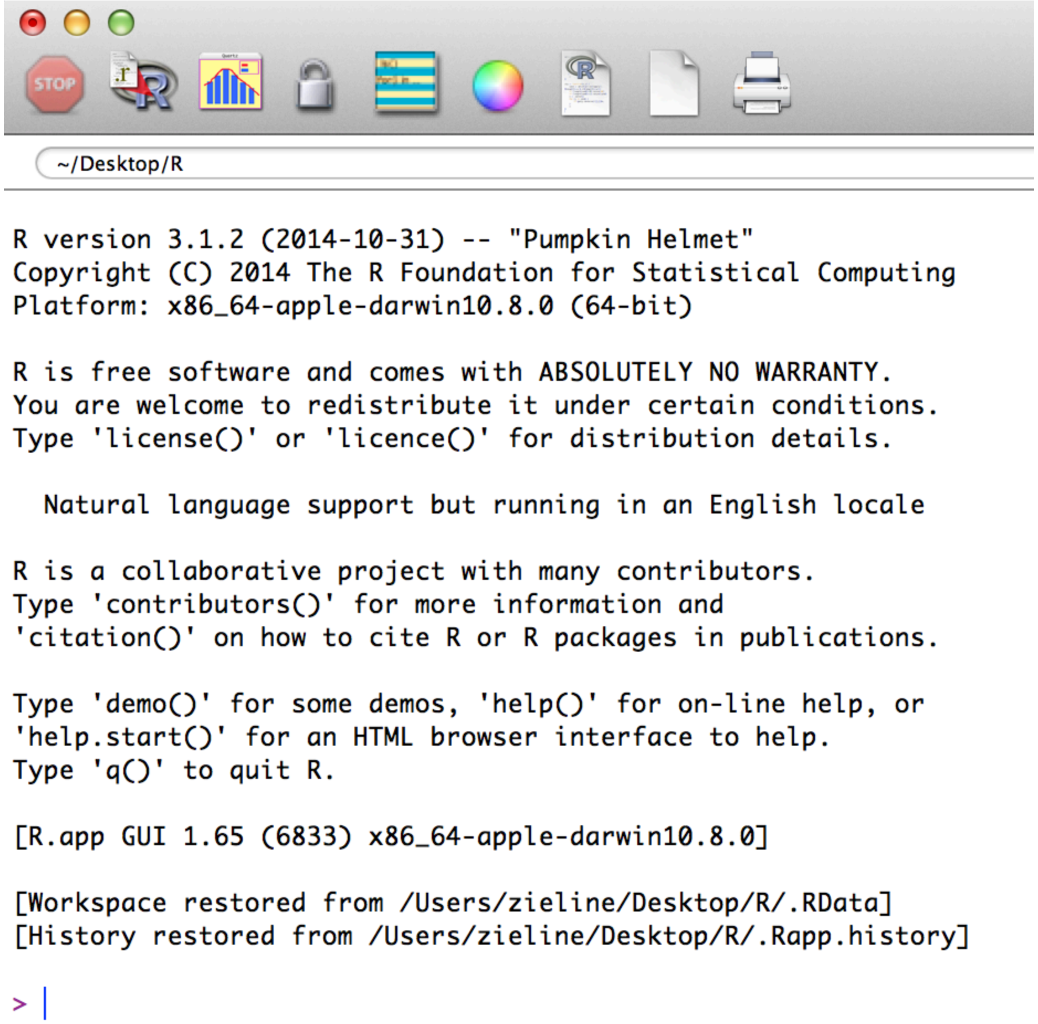
- R Studio vs R Project

  R Project - http://cran.r-project.org/

  R Studio - http://www.rstudio.com/

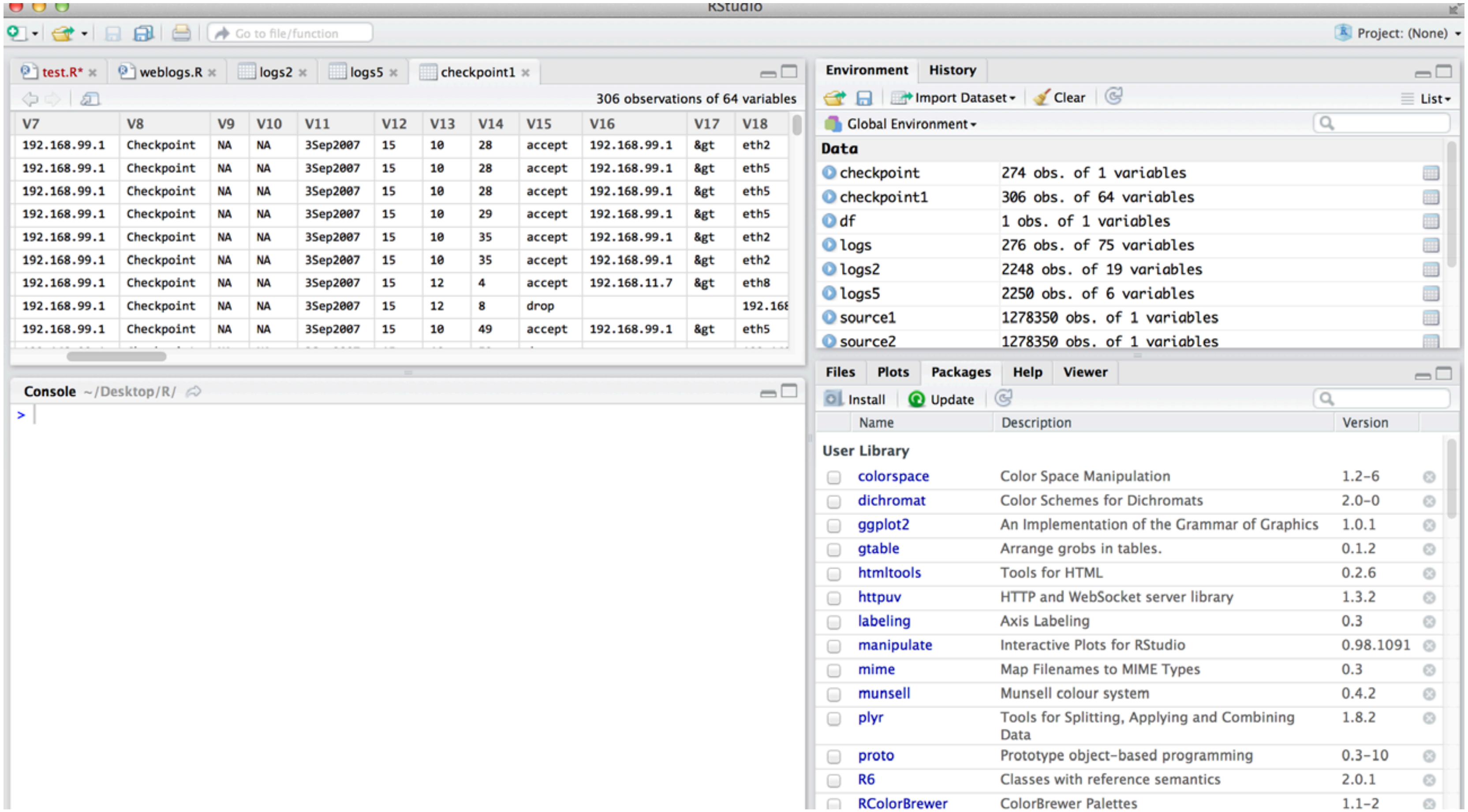
- There are thousands of packages!

R Project



```
R version 3.1.2 (2014-10-31) -- "Pumpkin Helmet"
Copyright (C) 2014 The R Foundation for Statistical Computing
Platform: x86_64-apple-darwin10.8.0 (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

  Natural language support but running in an English locale

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

[R.app GUI 1.65 (6833) x86_64-apple-darwin10.8.0]

[Workspace restored from /Users/zieline/Desktop/R/.RData]
[History restored from /Users/zieline/Desktop/R/.Rapp.history]

> |
```

RStudio



306 observations of 64 variables

V7	V8	V9	V10	V11	V12	V13	V14	V15	V16	V17	V18
192.168.99.1	Checkpoint	NA	NA	3Sep2007	15	10	28	accept	192.168.99.1	&gt	eth2
192.168.99.1	Checkpoint	NA	NA	3Sep2007	15	10	28	accept	192.168.99.1	&gt	eth5
192.168.99.1	Checkpoint	NA	NA	3Sep2007	15	10	28	accept	192.168.99.1	&gt	eth5
192.168.99.1	Checkpoint	NA	NA	3Sep2007	15	10	29	accept	192.168.99.1	&gt	eth5
192.168.99.1	Checkpoint	NA	NA	3Sep2007	15	10	35	accept	192.168.99.1	&gt	eth2
192.168.99.1	Checkpoint	NA	NA	3Sep2007	15	10	35	accept	192.168.99.1	&gt	eth2
192.168.99.1	Checkpoint	NA	NA	3Sep2007	15	12	4	accept	192.168.11.7	&gt	eth8
192.168.99.1	Checkpoint	NA	NA	3Sep2007	15	12	8	drop			192.168
192.168.99.1	Checkpoint	NA	NA	3Sep2007	15	10	49	accept	192.168.99.1	&gt	eth5

```
>
```

Data

checkpoint	274 obs. of 1 variables
checkpoint1	306 obs. of 64 variables
df	1 obs. of 1 variables
logs	276 obs. of 75 variables
logs2	2248 obs. of 19 variables
logs5	2250 obs. of 6 variables
source1	1278350 obs. of 1 variables
source2	1278350 obs. of 1 variables

	Name	Description	Version
	User Library		
□	colorspace	Color Space Manipulation	1.2-6
□	dichromat	Color Schemes for Dichromats	2.0-0
□	ggplot2	An Implementation of the Grammar of Graphics	1.0.1
□	gtable	Arrange grobs in tables.	0.1.2
□	htmltools	Tools for HTML	0.2.6
□	httpuv	HTTP and WebSocket server library	1.3.2
□	labeling	Axis Labeling	0.3
□	manipulate	Interactive Plots for RStudio	0.98.1091
□	mime	Map Filenames to MIME Types	0.3
□	munsell	Munsell colour system	0.4.2
□	plyr	Tools for Splitting, Applying and Combining Data	1.8.2
□	proto	Prototype object-based programming	0.3-10
□	R6	Classes with reference semantics	2.0.1
□	RColorBrewer	ColorBrewer Palettes	1.1-2

Memory

- How much memory is required to store data set in memory?

- How many rows and columns does your dataset contain?
  - 1,500,000 rows & 120 columns (all numeric data)

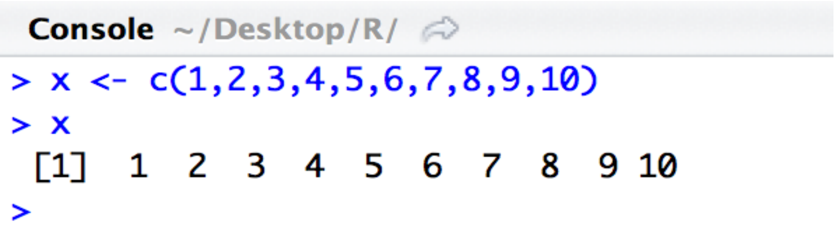
  - each number requires 8bytes of memory
  - numbers are stored using 64bit numbers
  - 8bits per byte, so 8 bytes of memory per numeric object
    - 1,500,000x120x8 bytes/numeric
    - 144000000 bytes
    - 1373.29 MB
    - 1.34GB Memory required.
  - Need a lil more than this to run, but not much more.

Up and Running

- Set your path for R to read your data sets from
- Installing packages (thousands of packages)
- Swirl - http://swirlstats.com/
- Lets see some commands!

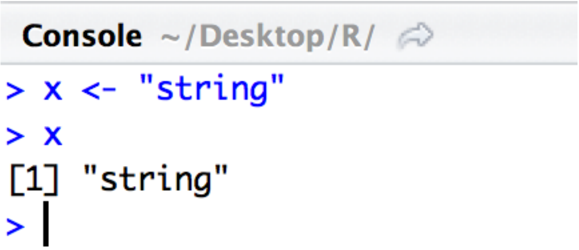
Overview of R

- Syntax example (storing numbers)
  - X <- c(10.4, 5.6, 2.3, 4.5 or whatever)



```
> x <- c(1,2,3,4,5,6,7,8,9,10)
> x
 [1]  1  2  3  4  5  6  7  8  9 10
>
```

- Syntax example (storing strings)
  - X <- “string”



```
> x <- "string"
> x
[1] "string"
> 
```

Quick Overview of R

- Data Types
- Objects
- Control structures – uses standard control structures
  - If else
  - For
  - While
  - Switch
- Functions
  - Fundamental building blocks of R
  - Functions are objects
  - 3 main objectives
    - Body ()
    - Formals ()
    - Environment ()

Getting started on reading Data

- Multiple ways to read data into R
  - Read.table, read.csv
  - readLines
  - Source
  - Dget
  - Load
  - Unserialize

Reading Data

Import an entire log file into a variable

```
data <- read.table(“logfile.txt”)
```

- File – where to get the data
- Header – indicates header line
- Sep – how columns are separated
- StringsAsFactors
- colNames – Names of the columns

Connections

- File – opens connection to file
- Gzfile – opens connection to gzip
- Bzfile – opens connection to bzip2
- url – opens connection to webpage

Cleaning up the memory mess

Your friends:

rm(list=ls()) – removes everything from memory

ctrl + L – clears the console

Now it’s time to dance!

Case Study

Web logs

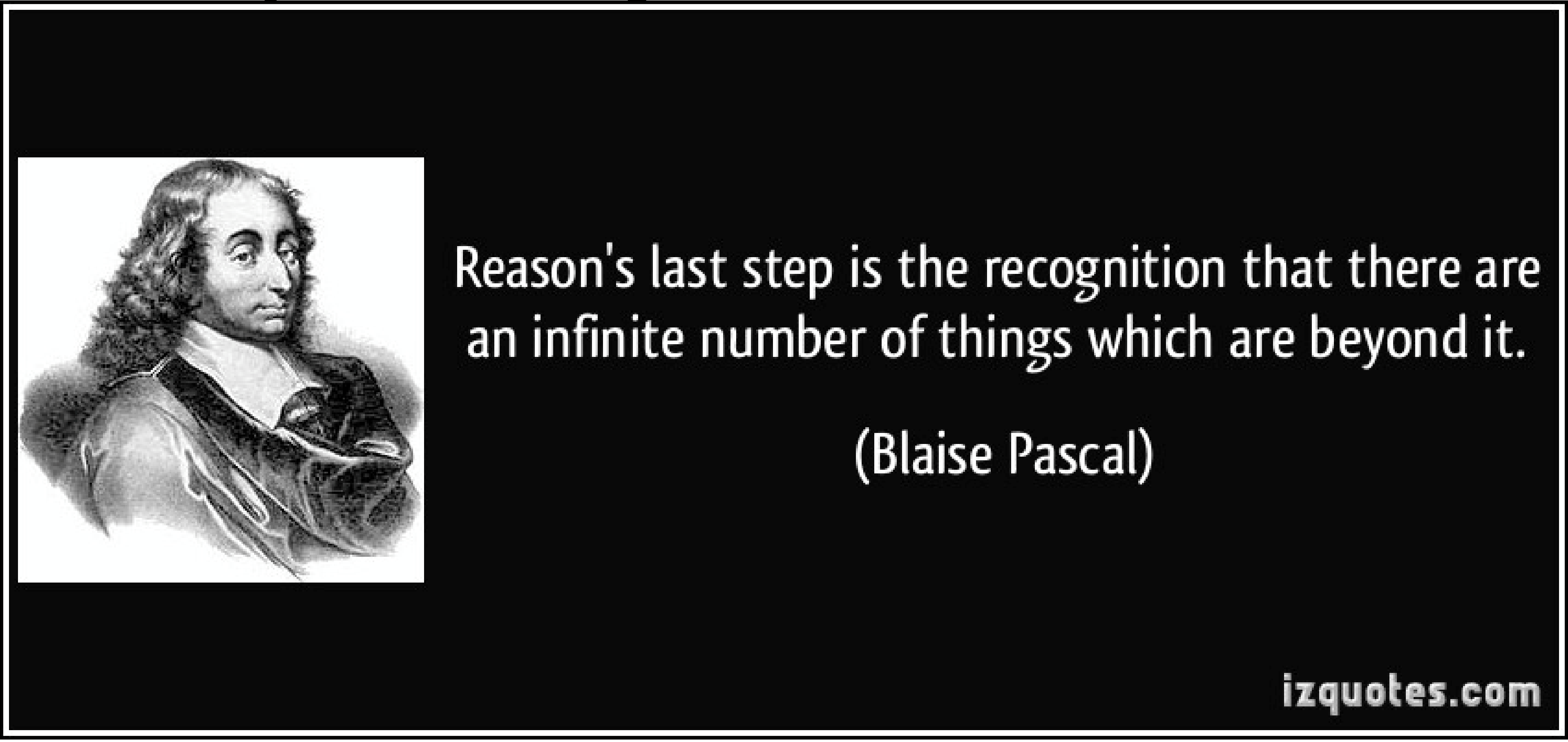
Step 1: Gather the logs

Step 2: Parse the logs

Ambition is the first step to success. The second step is action.

kushandwizdom.tumblr.com

Step 3: Analyze the data in R



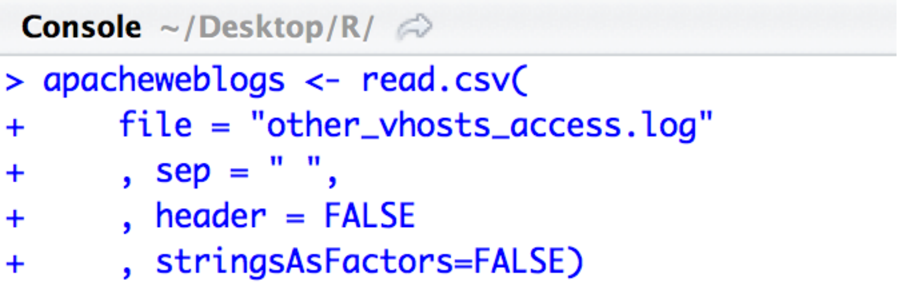
Reason's last step is the recognition that there are an infinite number of things which are beyond it.

(Blaise Pascal)

izquotes.com

Case Study: Reading the data

```
apachelogs <- read.csv(
    file = "other_vhosts_access.log"
    , sep = " “
    , header = FALSE
    , stringsAsFactors=FALSE)
```

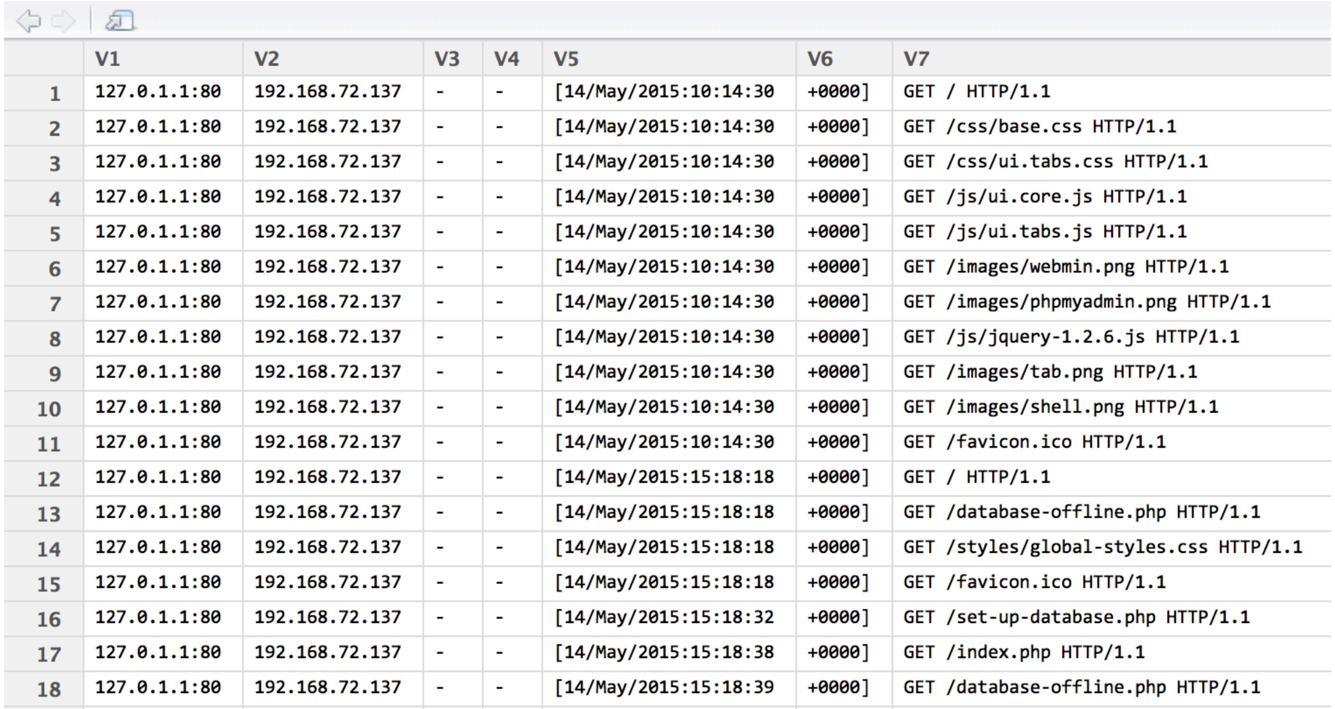


```
> apacheweblogs <- read.csv(
+     file = "other_vhosts_access.log"
+     , sep = " ",
+     , header = FALSE
+     , stringsAsFactors=FALSE)
```

https://stat.ethz.ch/R-manual/R-devel/library/utils/html/read.table.html

Example of log files

- Apache weblogs without column names



	V1	V2	V3	V4	V5	V6	V7
1	127.0.1.1:80	192.168.72.137	-	-	[14/May/2015:10:14:30	+0000]	GET / HTTP/1.1
2	127.0.1.1:80	192.168.72.137	-	-	[14/May/2015:10:14:30	+0000]	GET /css/base.css HTTP/1.1
3	127.0.1.1:80	192.168.72.137	-	-	[14/May/2015:10:14:30	+0000]	GET /css/ui.tabs.css HTTP/1.1
4	127.0.1.1:80	192.168.72.137	-	-	[14/May/2015:10:14:30	+0000]	GET /js/ui.core.js HTTP/1.1
5	127.0.1.1:80	192.168.72.137	-	-	[14/May/2015:10:14:30	+0000]	GET /js/ui.tabs.js HTTP/1.1
6	127.0.1.1:80	192.168.72.137	-	-	[14/May/2015:10:14:30	+0000]	GET /images/webmin.png HTTP/1.1
7	127.0.1.1:80	192.168.72.137	-	-	[14/May/2015:10:14:30	+0000]	GET /images/phpmyadmin.png HTTP/1.1
8	127.0.1.1:80	192.168.72.137	-	-	[14/May/2015:10:14:30	+0000]	GET /js/jquery-1.2.6.js HTTP/1.1
9	127.0.1.1:80	192.168.72.137	-	-	[14/May/2015:10:14:30	+0000]	GET /images/tab.png HTTP/1.1
10	127.0.1.1:80	192.168.72.137	-	-	[14/May/2015:10:14:30	+0000]	GET /images/shell.png HTTP/1.1
11	127.0.1.1:80	192.168.72.137	-	-	[14/May/2015:10:14:30	+0000]	GET /favicon.ico HTTP/1.1
12	127.0.1.1:80	192.168.72.137	-	-	[14/May/2015:15:18:18	+0000]	GET / HTTP/1.1
13	127.0.1.1:80	192.168.72.137	-	-	[14/May/2015:15:18:18	+0000]	GET /database-offline.php HTTP/1.1
14	127.0.1.1:80	192.168.72.137	-	-	[14/May/2015:15:18:18	+0000]	GET /styles/global-styles.css HTTP/1.1
15	127.0.1.1:80	192.168.72.137	-	-	[14/May/2015:15:18:18	+0000]	GET /favicon.ico HTTP/1.1
16	127.0.1.1:80	192.168.72.137	-	-	[14/May/2015:15:18:32	+0000]	GET /set-up-database.php HTTP/1.1
17	127.0.1.1:80	192.168.72.137	-	-	[14/May/2015:15:18:38	+0000]	GET /index.php HTTP/1.1
18	127.0.1.1:80	192.168.72.137	-	-	[14/May/2015:15:18:39	+0000]	GET /database-offline.php HTTP/1.1

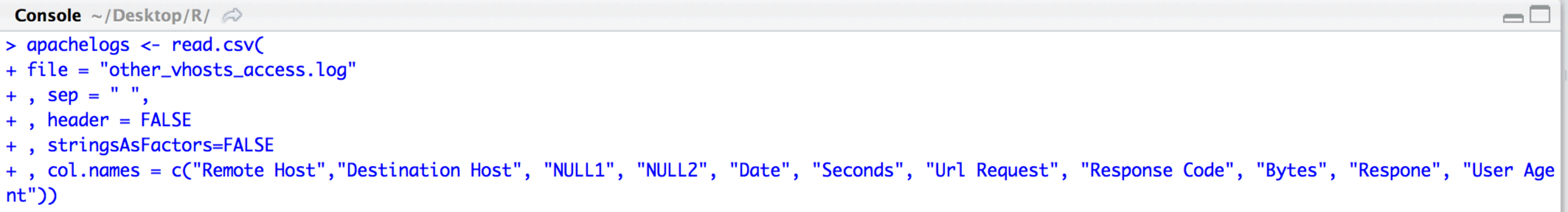
Diving deeper

- Understand your log format
- Apache log format

```
127.0.1.1:443 192.168.72.1 - - [17/May/2015:17:41:02 +0000] "GET
/images/cage.png HTTP/1.1" 200 4792
"https://192.168.72.151/index.php?page=capture-data.php"
"Mozilla/5.0 (Macintosh; Intel Mac OS X 10_9_5)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/42.0.2311.152
Safari/537.36"
```

Case Study: Reading the data

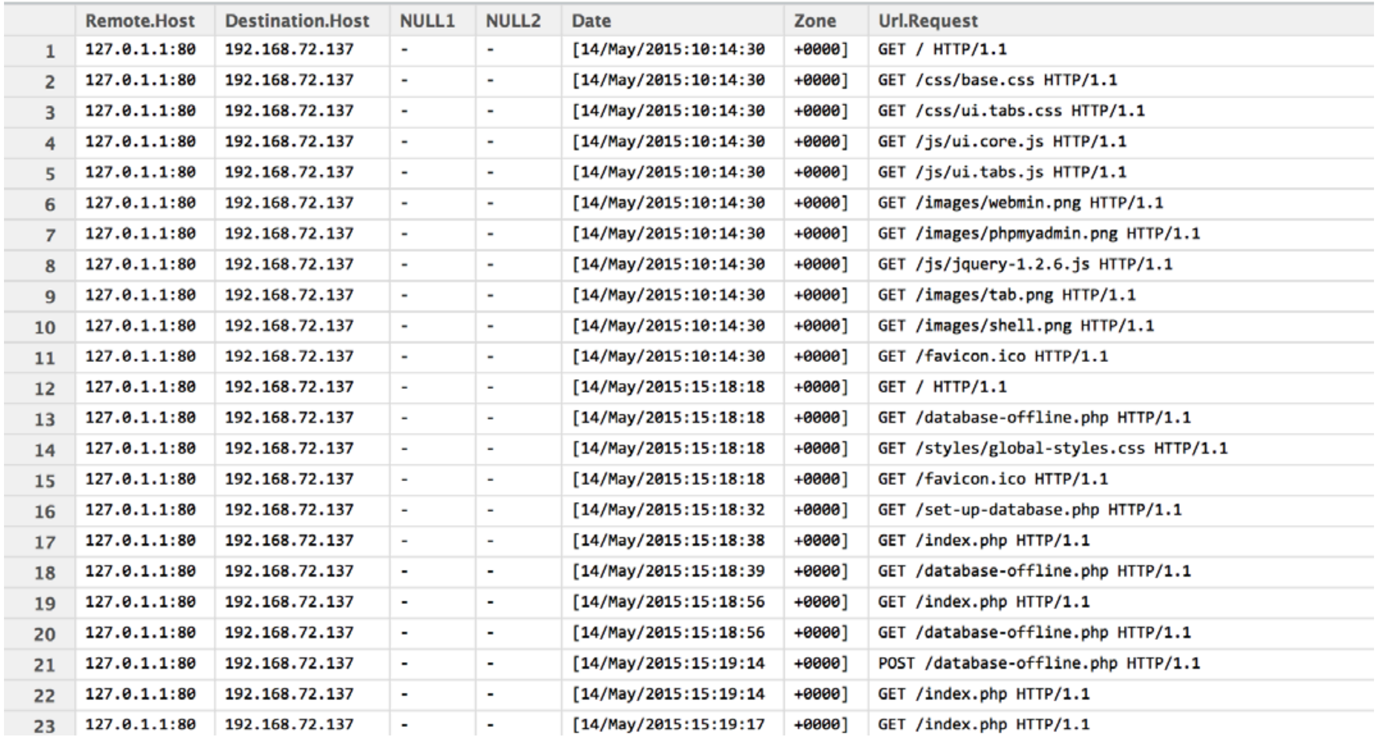
```
apachelogs <- read.csv(
    file = "other_vhosts_access.log”
    , sep = ” “
    , header = FALSE
    , stringsAsFactors=FALSE
    , col.names = c("Remote Host","Destination
Host", "NULL1", "NULL2", "Date", ”Zone", "Url
Request", "Response Code", "Bytes",
"Response", "User Agent"))
```



```
> apachelogs <- read.csv(
+ file = "other_vhosts_access.log"
+ , sep = " ",
+ , header = FALSE
+ , stringsAsFactors=FALSE
+ , col.names = c("Remote Host","Destination Host", "NULL1", "NULL2", "Date", "Seconds", "Url Request", "Response Code", "Bytes", "Respone", "User Age
nt"))
```

Example of log files

- Apache weblogs without column names

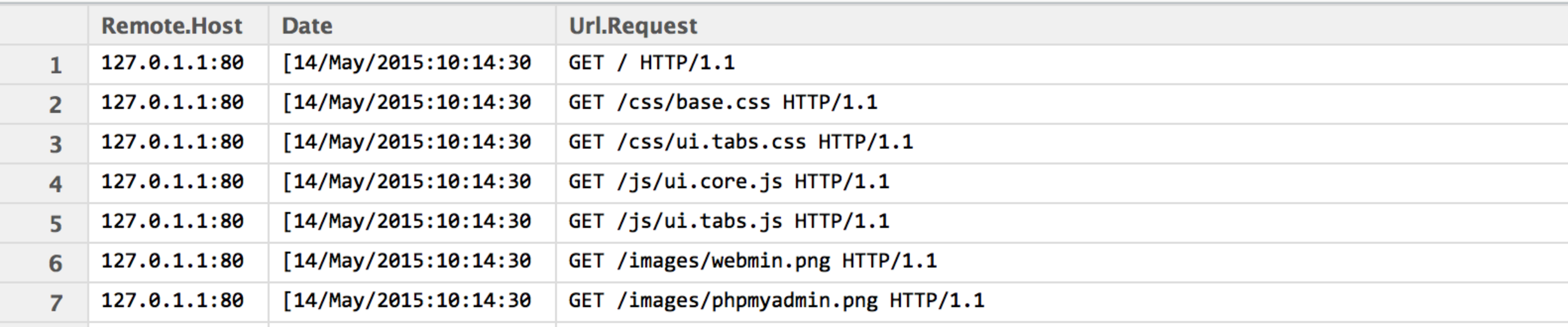


	Remote.Host	Destination.Host	NULL1	NULL2	Date	Zone	Url.Request
1	127.0.1.1:80	192.168.72.137	-	-	[14/May/2015:10:14:30	+0000]	GET / HTTP/1.1
2	127.0.1.1:80	192.168.72.137	-	-	[14/May/2015:10:14:30	+0000]	GET /css/base.css HTTP/1.1
3	127.0.1.1:80	192.168.72.137	-	-	[14/May/2015:10:14:30	+0000]	GET /css/ui.tabs.css HTTP/1.1
4	127.0.1.1:80	192.168.72.137	-	-	[14/May/2015:10:14:30	+0000]	GET /js/ui.core.js HTTP/1.1
5	127.0.1.1:80	192.168.72.137	-	-	[14/May/2015:10:14:30	+0000]	GET /js/ui.tabs.js HTTP/1.1
6	127.0.1.1:80	192.168.72.137	-	-	[14/May/2015:10:14:30	+0000]	GET /images/webmin.png HTTP/1.1
7	127.0.1.1:80	192.168.72.137	-	-	[14/May/2015:10:14:30	+0000]	GET /images/phpmyadmin.png HTTP/1.1
8	127.0.1.1:80	192.168.72.137	-	-	[14/May/2015:10:14:30	+0000]	GET /js/jquery-1.2.6.js HTTP/1.1
9	127.0.1.1:80	192.168.72.137	-	-	[14/May/2015:10:14:30	+0000]	GET /images/tab.png HTTP/1.1
10	127.0.1.1:80	192.168.72.137	-	-	[14/May/2015:10:14:30	+0000]	GET /images/shell.png HTTP/1.1
11	127.0.1.1:80	192.168.72.137	-	-	[14/May/2015:10:14:30	+0000]	GET /favicon.ico HTTP/1.1
12	127.0.1.1:80	192.168.72.137	-	-	[14/May/2015:15:18:18	+0000]	GET / HTTP/1.1
13	127.0.1.1:80	192.168.72.137	-	-	[14/May/2015:15:18:18	+0000]	GET /database-offline.php HTTP/1.1
14	127.0.1.1:80	192.168.72.137	-	-	[14/May/2015:15:18:18	+0000]	GET /styles/global-styles.css HTTP/1.1
15	127.0.1.1:80	192.168.72.137	-	-	[14/May/2015:15:18:18	+0000]	GET /favicon.ico HTTP/1.1
16	127.0.1.1:80	192.168.72.137	-	-	[14/May/2015:15:18:32	+0000]	GET /set-up-database.php HTTP/1.1
17	127.0.1.1:80	192.168.72.137	-	-	[14/May/2015:15:18:38	+0000]	GET /index.php HTTP/1.1
18	127.0.1.1:80	192.168.72.137	-	-	[14/May/2015:15:18:39	+0000]	GET /database-offline.php HTTP/1.1
19	127.0.1.1:80	192.168.72.137	-	-	[14/May/2015:15:18:56	+0000]	GET /index.php HTTP/1.1
20	127.0.1.1:80	192.168.72.137	-	-	[14/May/2015:15:18:56	+0000]	GET /database-offline.php HTTP/1.1
21	127.0.1.1:80	192.168.72.137	-	-	[14/May/2015:15:19:14	+0000]	POST /database-offline.php HTTP/1.1
22	127.0.1.1:80	192.168.72.137	-	-	[14/May/2015:15:19:14	+0000]	GET /index.php HTTP/1.1
23	127.0.1.1:80	192.168.72.137	-	-	[14/May/2015:15:19:17	+0000]	GET /index.php HTTP/1.1

Clean up

- Remove the columns

apachelogs$Zone <- NULL



	Remote.Host	Date	Url.Request
1	127.0.1.1:80	[14/May/2015:10:14:30	GET / HTTP/1.1
2	127.0.1.1:80	[14/May/2015:10:14:30	GET /css/base.css HTTP/1.1
3	127.0.1.1:80	[14/May/2015:10:14:30	GET /css/ui.tabs.css HTTP/1.1
4	127.0.1.1:80	[14/May/2015:10:14:30	GET /js/ui.core.js HTTP/1.1
5	127.0.1.1:80	[14/May/2015:10:14:30	GET /js/ui.tabs.js HTTP/1.1
6	127.0.1.1:80	[14/May/2015:10:14:30	GET /images/webmin.png HTTP/1.1
7	127.0.1.1:80	[14/May/2015:10:14:30	GET /images/phpmyadmin.png HTTP/1.1

Packages

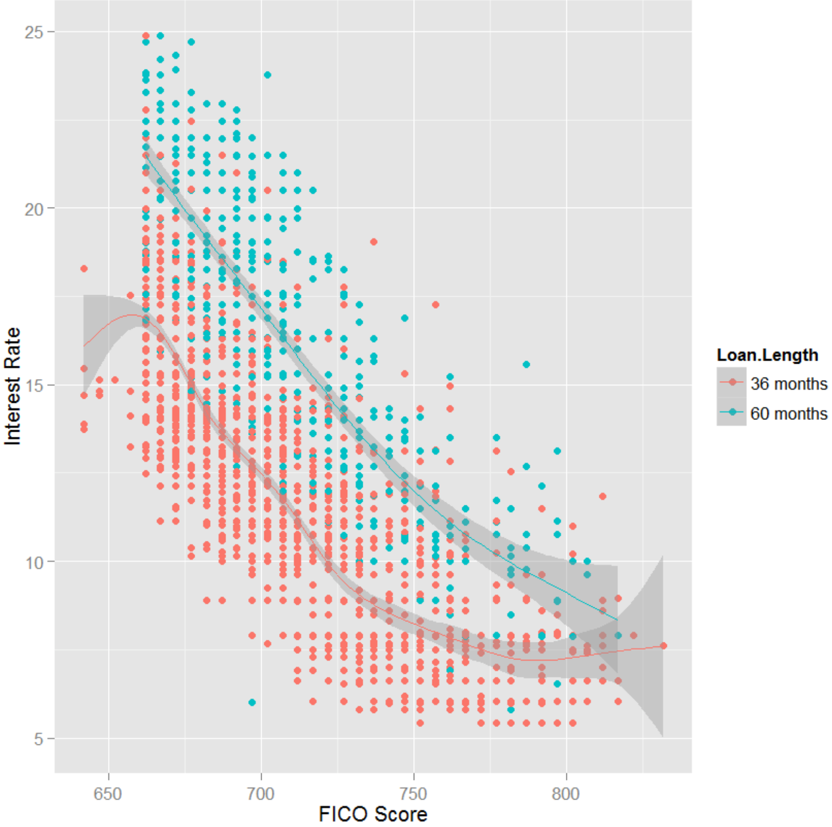
- Lots of functionality not delivered in the basic R install
- Bring on the packages
- Where can I find packages?
  - R Cran or Bioformatics or Github

  - install.packages("ggplot2")
  - library(ggplot2)

Is that all?

Visualize

- Ggplot2 allows for plotting information in a graph



Let’s try it!

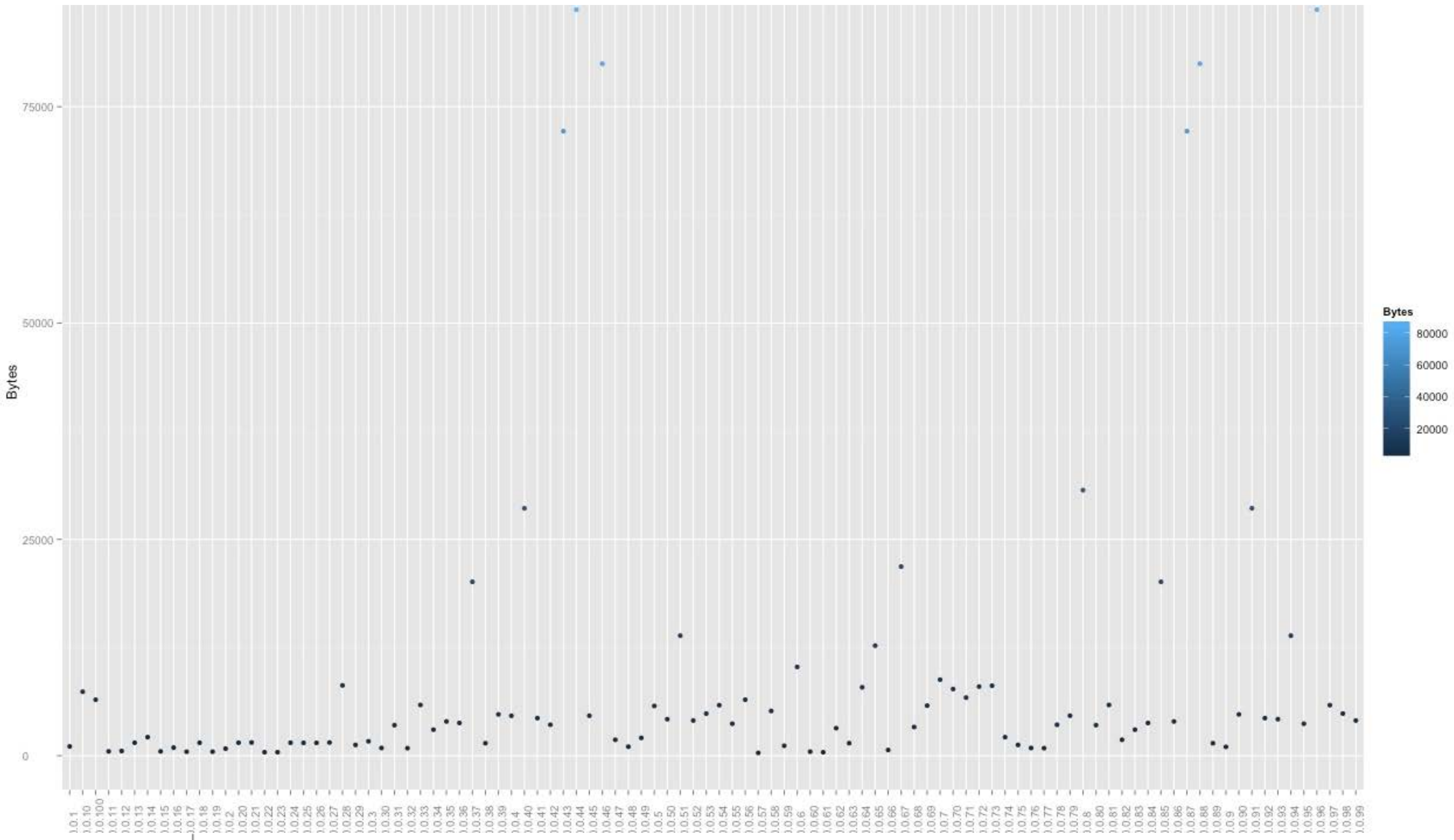
- Back to our web logs
- What would be interesting to graph?
  - How about Remote Hosts and Bytes? Why Not?

```
p <- qplot (Remote.Host, Bytes, color = Bytes, data = apachelogs)
```

But we need to clean it up a bit as always:

```
p + theme(axis.text.y=element_text(hjust=0, angle=0), axis.text.x = element_text(hjust=0, angle=90))
```

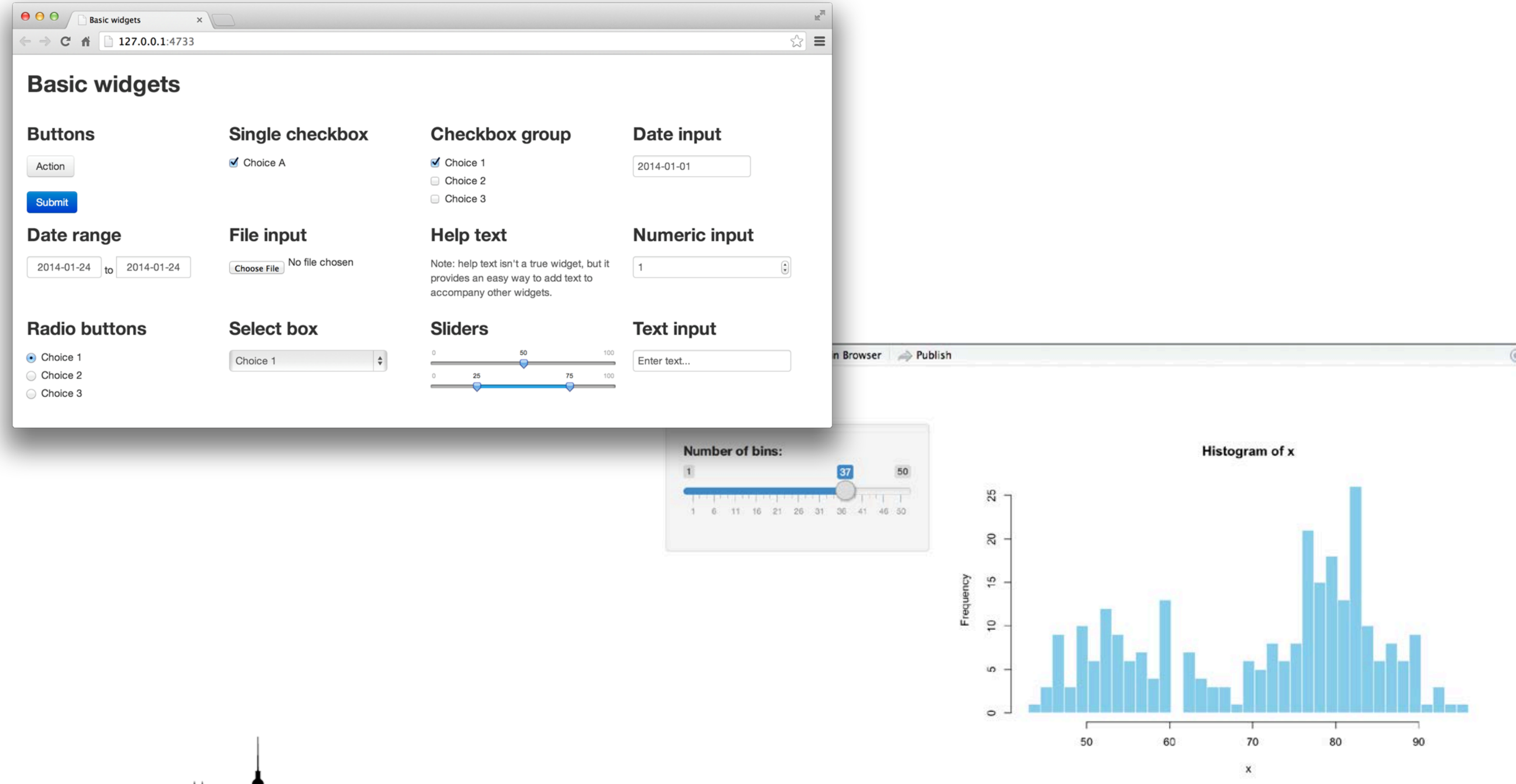
And we now have value



Shiny

- Let’s get creative!
- Shiny allows us to build our own dashboard
- R programs embedded into a web page
- Prediction algorithms – Shiny can call your algorhithm and display the results
- Uses bootstrap (looks nice and mobile friendly)

We can build Web Apps!



Basic widgets

127.0.0.1:4733

Basic widgets

Buttons

Action

Submit

Single checkbox

☑ Choice A

Checkbox group

☑ Choice 1
□ Choice 2
□ Choice 3

Date input

2014-01-01

Date range

2014-01-24 to 2014-01-24

File input

Choose File No file chosen

Help text

Note: help text isn't a true widget, but it provides an easy way to add text to accompany other widgets.

Numeric input

1

Radio buttons

● Choice 1
○ Choice 2
○ Choice 3

Select box

Choice 1

Sliders

Text input

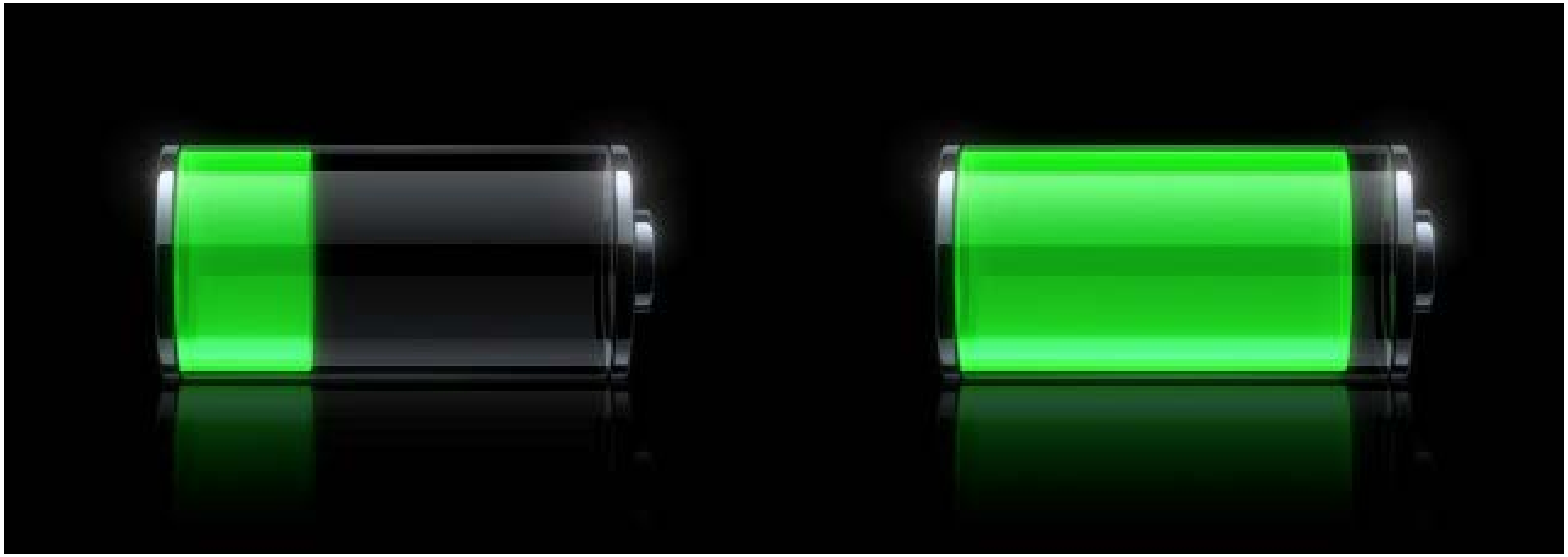
Enter text...

Case Study with R

- Again understand what your log format is
  - Know how you want to organize your data
  - Know what field headers they contain
  - Cleaning up your data can be teadious but worth it
  - There is much more to cleaning up the data than time allows
  - R allows for RegEx’s,

Now Lets Maximize!

- Merge multiple data sets into one
- Clean out the garbage data



Tidy it up!

- How about this scenario?
  - Web application is suspected of being compromised?
    - What do we need to investigate?
      - Web Application Logs
      - Web Server Logs
      - Firewall Logs
      - Server Logs
      - What other logs are available?

Tidyr & dplyr

- The tidyr package makes it easy to reshape the layout of your data sets while retaining the relationships embedded in the data
- Makes your data “Tidy”
- Group your data with dplyr

Putting it all together

R allows us to pull the data directly from the sources

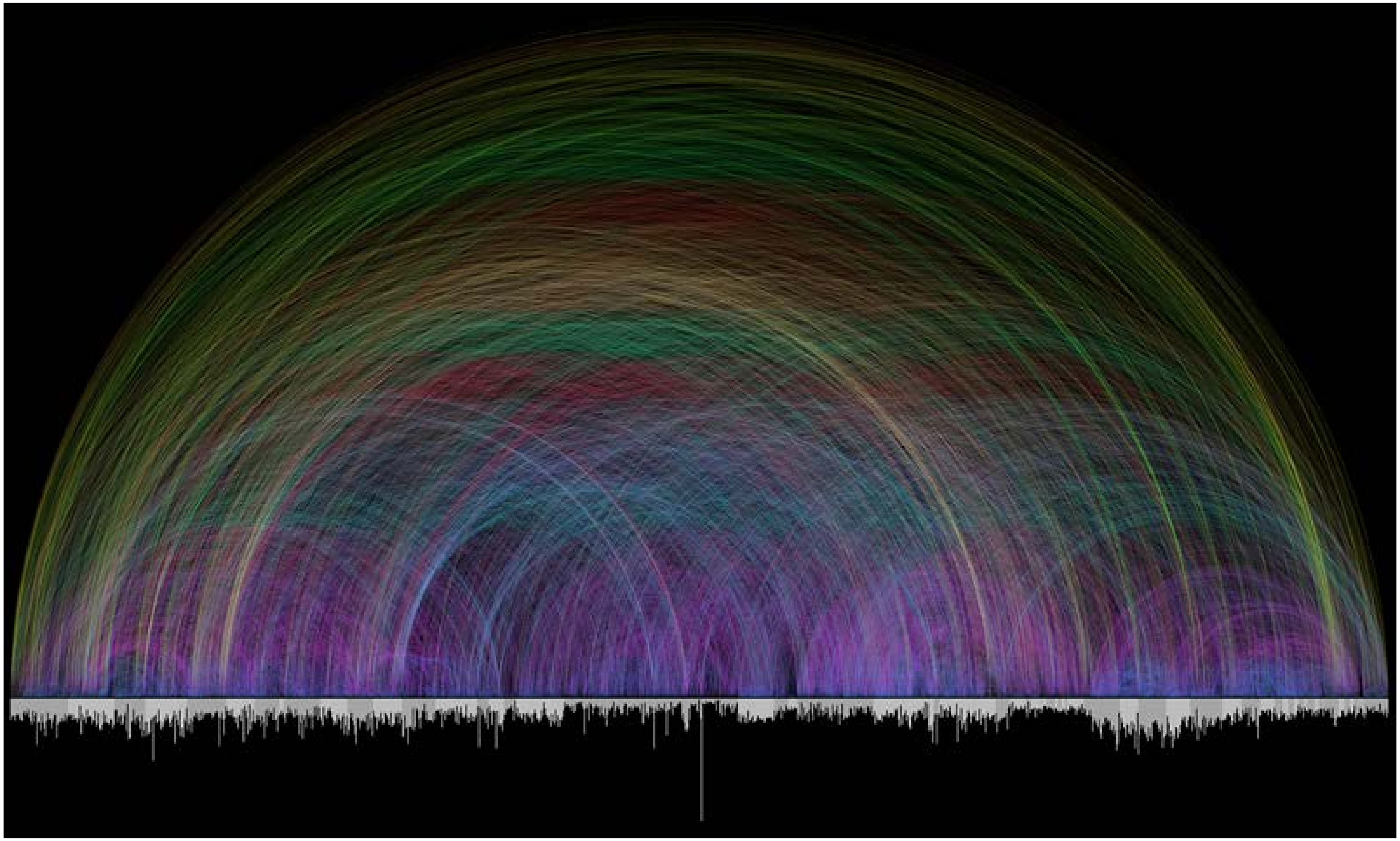
- pull out the interesting information
- create a script for the following:
    - reading the logfiles
    - pulling data (website, web crawling of data)
- once data sets are pulled we need to clean them (remove columns, null data, unnecessary fields)
- next script them to merge into one giant data set

- Factors to consider
    - many to many relationships
    - need to understand data to validate the merges and joins
    - multiple sets of code for graphics and visualization

End Results

- One massive data set that can be scripted, searched, and visualized
- Create algorithms to determine normal
- Show us the outliers, strange data, things not expected
- Activity of certain data sets

Data Exploration

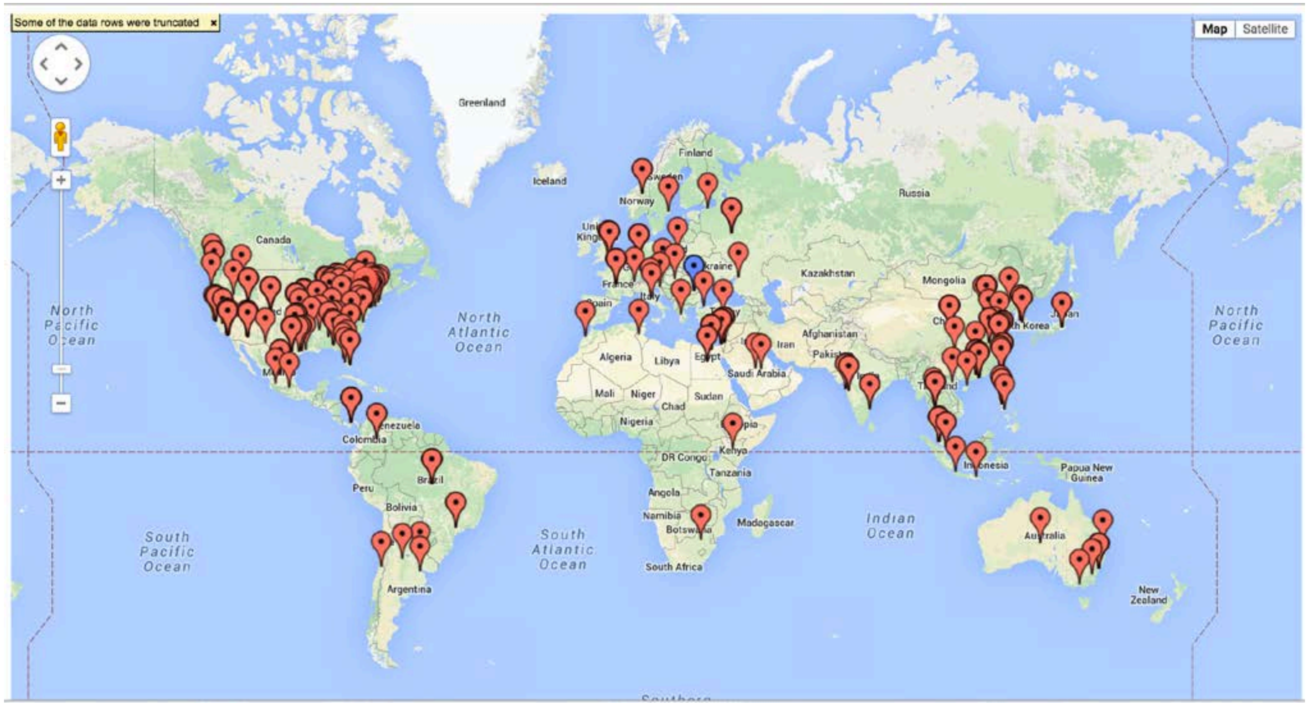


How to Apply in Real World?

What if we were to take data from a bad reputation IP list and map it?

Yes We Can!

By using libraries such as ggplot, lattice, googleVis, ggmap and calling the URL we can download a reputation list and plot the locations on the map!



Baselines

- Baseline 7 days Database logs
- - Take 1 hr of SQL Queries or 1 Day or 1 Week
    - TimeStamps
    - Server Type (which servers accessed the most)
    - Client IP / Server IP
    - DB Usernames
    - Source Program (to help identify client source)
    - SQL query

How about Netflow data

Top Talkers

- Who is talking to whom?
- what date/time
- volume

Bottom Talkers

Can we build our own SIEM?

- Live Data vs Archived Data issues

Feed your animal

Behavior based analysis

Recon analysis

Indicators of Compromise

Vulnerability Scanning

Unlimited Possibilities

How does this scale?

- It won’t always scale on your desktop
- Good for incident response analysis
- Long term need to move to big data Hadoop type solution
- Big R runs on Hadoop

cheatsheets

- Plenty of cheatsheets available from Rstudio
- http://www.rstudio.com/resources/cheatsheets/
- R Dir
- http://r-dir.com/reference/crib-sheets.html
- R Bloggers
- http://www.r-bloggers.com/the-data-table-cheat-sheet/

Thank you!

Questions?