

Severe Weather Impact in the United States

Major health and economic impacts pulled from NOAA Storm Database

Charles Tilford

Contents

Summary	1
Data Processing	2
Simplifying the input data set	2
Loading the simplified data	3
Normalizing the Event Type	4
Aggregating the data	5
Results	6
Older data (Prior to 1983)	6
More recent data (1983 to 2011)	6
Alignment with known major weather events.	11
Major Omissions	11
Thoughts on R / ggplot / etc	12
Acknowledgements	12
System Information	12

Summary

This analysis is the [second assignment](#) of the Coursera [Reproducible Research](#) class. The data are from the [NOAA Storm Data Publication](#) service, provided from a [cache on CloudFront](#) (!47Mb file!). The assignment asks that the following two questions be addressed:

1. Across the United States, which types of events (as indicated in the EVTYPE variable) are most harmful with respect to population health?
 - *Nutshell:* **Heat** and **tornadoes** pose the greatest threat to health, with flooding causing sporadic significant events
2. Across the United States, which types of events have the greatest economic consequences?
 - *Nutshell:* **Tornados** are a constant, significant destroyer of property, but **hurricanes** and **floodings** sporadically cause immense property damage. Crops are most affected by **drought**, **ice** and **floodings**, with **hurricanes** also wreaking occasional major impact.

As this is an educational exercise, I am keeping the vast majority of R code visible. In normal circumstances I would suppress much of it with `echo = FALSE`

Data Processing

Simplifying the input data set

The raw data set is large, and includes many fields that are not relevant for this analysis. For efficiency while exploring the data a smaller derivative file is first made:

```
simpleFile <- "StormDataSimple.tsv"

if (!file.exists(simpleFile)) {
  message("Generating simplified data file...")
  ## Much of the primary file is not of interest to us. Simplify the
  ## file to just the fields we will use
  sourceFile <- "repdata_data_StormData.csv.bz2"
  if (!file.exists(sourceFile)) {
    url <- "https://d396qusza40orc.cloudfront.net/repdata%2Fdata%2FStormData.csv.bz2"
    stop(paste("Source data not found. Please download from:",
              url, collapse = "\n"))
  }
  full <- read.csv(sourceFile)
  ## Filter the results to just the 50 "standard" states:
  validState <- full$STATE %in% state.abb
  ## Will reduce 902k observations to 883k
  state50 <- full[ validState, ]
  ## Do not care about the time of day, most seem bogus anyway:
  state50$Days <- format(strptime(state50$BGN_DATE,
                                "%m/%d/%Y %H:%M:%S"), "%Y-%m-%d")

  ## Need to handle the "exponent" for the damage values
  exponentParser <- function(token) {
    if (is.null(token)) return(1)
    token <- tolower(token)
    if (token == 'h') {
      100 # I presume "hundred"?
    } else if (token == 'k') {
      1000
    } else if (token == 'm') {
      1000000 # million
    } else if (token == 'b') {
      1000000000 # billion
    } else if (grepl('[1-9]$', token)) {
      # Presume it is an actual exponent??
      10 ^ as.integer(token)
    } else {
      # No idea. Leave it alone. Stuff like "+" and "-"
      0
    }
  }

  numrows <- nrow(state50)
  propDamage <- state50$PROPDGMG
  cropDamage <- state50$CROPDGMG
  for (r in 1:numrows) {
    mp <- exponentParser(state50$PROPDGMGEXP[r])
    if (mp != 0) propDamage[r] <- propDamage[r] * mp
    ## Ugh. There's a mis-coded entry for a 2006 Napa Valley flood
```

```

## that claims "115B" ([B]illion) in property damage. REMARKS
## = "The City of Napa had 600 homes with moderate damage, 150
## damaged businesses with costs of at least $70 million."
if (state50$REFNUM[r] == 605943 & propDamage[r]) {
  ## REFNUM is a unique accessor for the event, but I am
  ## checking the value as well in case it later gets
  ## corrected.
  propDamage[r] <- 70e6
}
mc <- exponentParser(state50$CROPDMGEXP[r])
if (mc != 0) cropDamage[r] <- cropDamage[r] * mc
}
state50$Property <- propDamage
state50$Crop <- cropDamage
## The reference number is very handy for tracing weird data
## points back to their source in the original giant CSV
simp <- data.frame(Date = state50$Days,
  RawEvent = state50$EVTYPE,
  Deaths = state50$FATALITIES,
  Injuries = state50$INJURIES,
  Property = state50$Property,
  Crop = state50$Crop,
  State = state50$STATE,
  RefNum = state50$REFNUM )
write.table( simp, file = simpleFile, sep = "\t", row.names = FALSE,
  quote = FALSE)

## I am finding browsing the Remarks tedious. Copy the big ones to
## their own file.
majorEvents <- state50[ propDamage > 100e6 | cropDamage > 100e6 |
  state50$FATALITIES > 100 | state50$INJURIES > 500,
  c("REFNUM", "Days", "EVTYPE", "STATE",
    "FATALITIES", "INJURIES", "Property", "Crop",
    "REMARKS")]
## Woah. Unescaped newlines in a single CSV record.
majorEvents$REMARKS <- gsub("\n\r|+", " ", majorEvents$REMARKS)
remFile <- "MajorEventRemarks.tsv"
write.table( majorEvents, file = remFile, sep = "\t", row.names = FALSE,
  quote = FALSE)

## Clean up memory - I think?
full <- NULL
days <- NULL
state50 <- NULL
simp <- NULL
invisible(gc())
}

```

Loading the simplified data

```

## Read the simplified data set
colCls <- c(rep("character", 2), rep("numeric", 4), "character", "numeric")

```

```
data <- read.table(simpleFile, header = TRUE, sep = "\t",
                  colClasses = colCls )
data$Date <- strptime(data$Date, "%Y-%m-%d")
data$Year <- as.integer(format(data$Date, "%Y"))
data$State <- as.factor( data$State )
```

Normalizing the Event Type

The raw data are also extremely poorly normalized; There appear to be no constraints on event naming, and many events are listed under different names or with abbreviation or spelling differences. I have made an attempt to normalize the event types programatically, dealing with capitalization (Flood vs FLOOD), whitespace insanity (" WIND" vs "WIND") and some simple [stemming](#) (FLOOD, FLOODS, FLOODING). I also generated an [alias file](#) that manually collects “the same” events under a sensible parent event type.

While tedious, these manipulations are extremely helpful in concentrating the major impacts from many modest effects into a handful of major ones.

```
## Pull in the aliases - SEE THIS FILE FOR THE ASSIGNMENTS I'VE MADE:
source("StormDataAliases.R")
normalEvent <- list()
stemmedKeys <- list()
eventRaw <- unique(data$RawEvent)
for (e in eventRaw) {
  ## Nice-case the events. Lowercase and remove non alphanumeric from end:
  norm <- gsub("[^a-z0-9]$", tolower(e), rep = "")
  ## Wow. So many spaces
  norm <- gsub("^+", norm, rep = "")
  norm <- gsub(" +$", norm, rep = "")
  norm <- gsub(" +", norm, rep = " ")
  ## Uppercase first letter:
  substr(norm, 1, 1) <- toupper(substr(norm, 1, 1))
  ## Map over aliases
  alias <- aliases[[norm]]
  if (!is.null(alias)) norm <- alias

  ## Some special-case common classes. Some false positives here
  if (grepl('tornado', norm, ignore.case = T)) norm <- 'Tornado'
  if (grepl('hurricane', norm, ignore.case = T)) norm <- 'Hurricane / TS'
  if (grepl('hail', norm, ignore.case = T)) norm <- 'Ice'
  if (grepl('\\\\bice\\\\b', norm, ignore.case = T)) norm <- 'Ice'
  if (grepl('snow', norm, ignore.case = T)) norm <- 'Snow'
  if (grepl('flood', norm, ignore.case = T)) norm <- 'Flood'
  if (grepl('wind', norm, ignore.case = T)) norm <- 'Wind'

  ## Deal (crudely) with plurals:
  key <- gsub("i?e?s+$", norm, rep = "")
  ## "Costal flooding" -> "Costal flood"
  key <- gsub("ing$", key, rep = "")
  if (key == "") key <- "Unknown"
  ## Keep the first instance of a key as the value to use:
  if (is.null(stemmedKeys[[ key ]])) stemmedKeys[ key ] <- norm
  normalEvent[ e ] <- stemmedKeys[[ key ]]
  ## SOOO much more could be done ... this field is a nightmare
```

```

}

data$Event <- vapply(data$RawEvent, function(x) { normalEvent[[x]] }, "")
data$Event <- as.factor( data$Event)
allEvents <- sort(levels(data$Event))

## Make a melted frame with data aggregated by year and event
melted <- melt(data, id = c("Year", "Event"),
               measure.vars = c("Deaths", "Injuries", "Property", "Crop"))
annualEvents <- aggregate(value ~ Year + Event + variable,
                           melted, FUN = sum)

## How many recent years should we consider for the 'top' events?
numRecentYears <- 10
## How many major events should we consider for each category?
numEventsPerType <- 5

```

Aggregating the data

There are four casualty categories: Fatalities, Injuries, Property Damage and Crop Damage. For each category, find the top 5 events, considering the last 10 years.

```

maxYear <- max(annualEvents$Year) ## What is the most recent year in data?
minRecent <- maxYear - numRecentYears + 1
mostRecent <- annualEvents[ annualEvents$Year >= minRecent, ]
recentSum <- aggregate( value ~ Event + variable, mostRecent, FUN = sum,
                        na.rm = TRUE)

## Eh. Not sure this is an efficient way to go about this...
recentWide <- reshape(recentSum, idvar = "Event", direction = "wide",
                      timevar = "variable")
numEvents <- nrow(recentWide) ## Total distinct events in full dataset
## Add rank columns for each of the four categories:
recentWide <- mutate(recentWide,
                     rd = numEvents - rank(recentWide$value.Deaths),
                     ri = numEvents - rank(recentWide$value.Injuries),
                     rp = numEvents - rank(recentWide$value.Property),
                     rc = numEvents - rank(recentWide$value.Crop))

## Select just those events that fall into the top ranks
topEvents <- with( recentWide, {
  recentWide[ rd < numEventsPerType | ri < numEventsPerType |
              rp < numEventsPerType | rc < numEventsPerType, ]
})

## All the top events from the 4 categories:
topNames <- topEvents$Event
## Order the events with maximal property damage at top:
topNames <- as.character(topNames[ order(topEvents$rp, decreasing = FALSE) ])

majorRows <- annualEvents$Event %in% topNames
majorEvents <- annualEvents[ majorRows, ]

```

```
## What about everything else? How much are we leaving out?
otherLabel <- "All Other"
minorEvents <- annualEvents[ !majorRows, ]
allMinor <- aggregate(value ~ Year + variable, minorEvents,
                      FUN = sum, na.rm = TRUE)
allMinor$Event <- rep(otherLabel, nrow(allMinor))
## Add the "other" category to our summary data:
majorEvents <- rbind(majorEvents, allMinor)
majorEvents$Event <- factor(as.character(majorEvents$Event),
                          levels = c(topNames, otherLabel))

numMajEvents <- length(levels(majorEvents$Event));
```

Results

```
## Summarize older data before more detailed reporting occurs.
## What year do crop damage results start? -> 1993
cropsSeen <- min(annualEvents[ annualEvents$variable == "Crop" &
                              annualEvents$value > 0, "Year"])
## limited data before 1982, just tornado reporting - get mean values
## for that time.
oldYear <- 1983 # From looking at the data
maxYear <- max(annualEvents$Year) ## What is the most recent year in data?
olderData <- with(majorEvents, {
  majorEvents[ Year < oldYear & Event == "Tornado" & variable != "Crop", ]
})
oldTornado <- aggregate( value ~ Event + variable, olderData, mean)
oldTornado <- reshape(oldTornado, idvar = "Event", direction = "wide",
                     timevar = "variable")
recentEvents <- majorEvents[ majorEvents$Year >= oldYear, ]
```

Older data (Prior to 1983)

Crop data are not available until 1993. Limited data are available prior to 1980, primarily showing the impact from tornadoes, which hold a fairly steady annual average of 105 deaths, 1715 injuries and US\$534 million in property damage.

More recent data (1983 to 2011)

```
## Since they use different scales, we will break out injury and
## damages as separate data structures:
recInj <- with(majorEvents,
              majorEvents[Year >= oldYear &
                          (variable == "Deaths" | variable == "Injuries"), ])
recDmg <- with(majorEvents,
              majorEvents[Year >= oldYear &
                          (variable == "Property" | variable == "Crop"), ])

majTickX <- seq(from = 1950, to = 2010, by = 5) # 5-year major ticks
```

```

minTickX <- 1950:2011

maxInj    <- 2500
injTickY  <- seq(from = 0, to = maxInj, by = 500)
maxDmg    <- 1e10
dmgTickY  <- seq(from = 0, to = maxDmg, by = 1e9) # Billion dollar ticks
billions <-function(x) {
  ## Format dollar amounts as billions
  ifelse(x == 0, "", sprintf("%$%dbn", x / 1e9))
}

## Out-of-bounds logic:
oobFunc <- function(points,lim) {
  ## Huh. I can just pass back the OOB values, and it does what I
  ## want. There's probably a string value (eg "as-is" or "keep") I
  ## can pass, but I got lost in the documentation and gave up looking.
  points
  # max <- lim[2]
  # vapply(points, function(x) { ifelse(x > max, max * 1.1, x) }, 0 )
}

## First build deaths and injuries:
injPlot <- ggplot( recInj ) +
  geom_line( aes(x = Year, y = value, color = Event, linetype = Event)) +
  scale_linetype_manual(values = rep(c('solid','dashed', "dotted"),
                                     length.out = numMajEvents),
                       guide = guide_legend( ncol = 4)) +
  facet_wrap( "variable", nrow = 1) +
  scale_x_continuous(breaks = majTickX, minor_breaks = minTickX) +
  scale_y_continuous(limits = c(0,maxInj), breaks = injTickY,
                     oob = oobFunc) +
  ylab("Impacted Individuals") +
  theme(legend.position=c(0,1), legend.justification=c(0, 1))

## Manually annotate the out-of-bounds values
injOOB <- recInj[ recInj$value > maxInj, ]
injPlot <- injPlot +
  geom_point( data = injOOB, aes(x = Year, y = maxInj ), shape = 17) +
  geom_text_repel(data = injOOB, show.legend = FALSE,
                 aes(x = Year, y = maxInj, label = value),
                 box.padding = unit(0.45, "lines"))

## Now monetary damages:
dmgPlot <- ggplot( recDmg ) +
  geom_line( aes(x = Year, y = value, color = Event, linetype = Event)) +
  scale_color_discrete(guide = FALSE) + # Needed to suppress extra legend
  scale_linetype_manual(values = rep(c('solid','dashed', "dotted"),
                                     length.out = numMajEvents),
                       guide = FALSE ) +
  facet_wrap( "variable", nrow = 1) +
  scale_x_continuous(breaks = majTickX, minor_breaks = minTickX) +
  scale_y_continuous(limits = c(0,maxDmg), breaks = dmgTickY,

```

```

                                label = billions, oob = oobFunc ) +
  ylab("Damage (US Dollars)")

## Manually annotate the out-of-bounds values
dmgOOB <- recDmg[ recDmg$value > maxDmg, ]
## Make the OOB damage values nice:
bigDmg <- vapply(dmgOOB$value, function(x) sprintf("%dbn", round(x / 1e9)), "")
dmgPlot <- dmgPlot +
  geom_point( data = dmgOOB, aes(x = Year, y = maxDmg ), shape = 17) +
  geom_text_repel(data = dmgOOB, show.legend = FALSE,
    aes(x = Year, y = maxDmg, label = bigDmg),
    box.padding = unit(0.45, "lines"))

```

Tornadoes remain a constant threat to both lives and property, even when other weather factors are considered, and are the leading cause of weather-related injury in a “typical” year. For fatalities, **Heat** appears to be a major contributor, and also is responsible for a large share of injury as well.

Flooding (which likely encompass hurricane-related events like storm surge) is a sporadic event that does follow a clear trend, but can be devastating to property and cause extensive injury. 1998 was a particularly bad year for flooding, with over 6400 injuries and \$7bn in damages. **Hurricanes** and **Tropical Storms** (TS) also have the potential to inflict billions in damages.

For crops, **Drought** is unsurprisingly a major, consistent threat. Similarly, **Ice** and **Floods** are significant concerns. **Hurricanes** can impact crop losses as well. Interestingly, while tornados appear prominently in all other categories, crops are relatively unaffected, presumably due to the highly localized nature of the damage.

```

grid.arrange(injPlot, dmgPlot, nrow=2)

```




```
## Organize the information by state, event and impact type for the
## most recent data.
stateMelt <- melt(data[data$Year >= minRecent,], id = c("State", "Event"),
  measure.vars = c("Deaths", "Injuries", "Property", "Crop"))

## Collapse
byState <- aggregate(value ~ State + Event + variable, stateMelt, sum)
names(byState)[3] <- "Impact"
## Not really needed, but makes exploring easier. Eliminate zeros:
byState <- byState[byState$value > 0, ]
## For each state and each impact type, what is the major event in recent years?
allStates <- unique(byState$State) # should be 50
allTypes <- unique(byState$Impact) # should be 4
aa <- length(allStates) * length(allTypes)
## Build a blank data frame with all State/Impact combinations:
maxState <- data.frame(State = rep(allStates, each = length(allTypes)),
  Impact = rep(allTypes, times = length(allStates)),
  Event = vector("character", aa),
  value = vector("numeric", aa))
for (i in seq_len(nrow(maxState))) {
  ## What are the observed events for this state/impact
  all <- byState[byState$State == maxState$State[i] &
    byState$Impact == maxState$Impact[i], c("Event", "value")]
  ## What's the maximal value?
```

```

m <- ifelse(nrow(all) == 0, 0, max(all$value, na.rm = TRUE))
maxState$value[i] <- m
## What event(s) does that represent?
top <- as.character(all[ all$value == m, ][[1]])
## Which of those events were previously reported as a "major" event?
tLogi <- top %in% topNames
top <- sort(top[ tLogi ])
numTop <- length(top)
if (numTop == 0) {
  ## The major impact for this state is not a major category
  maxState$Event[i] <- otherLabel
} else {
  maxState$Event[i] <- top[1]
  ## Warn if there were more than one top event. We just pick one
  if (numTop > 1) message
    (paste(c("Multiple maximal events",
             as.character(maxState$State[i]),
             as.character(maxState$Impact[i]), top),
           collapse = ' | '))
}
}

## For coordinating colors make sure that the levels match those used before:
maxState$Event <- factor(maxState$Event, levels = levels(majorEvents$Event))

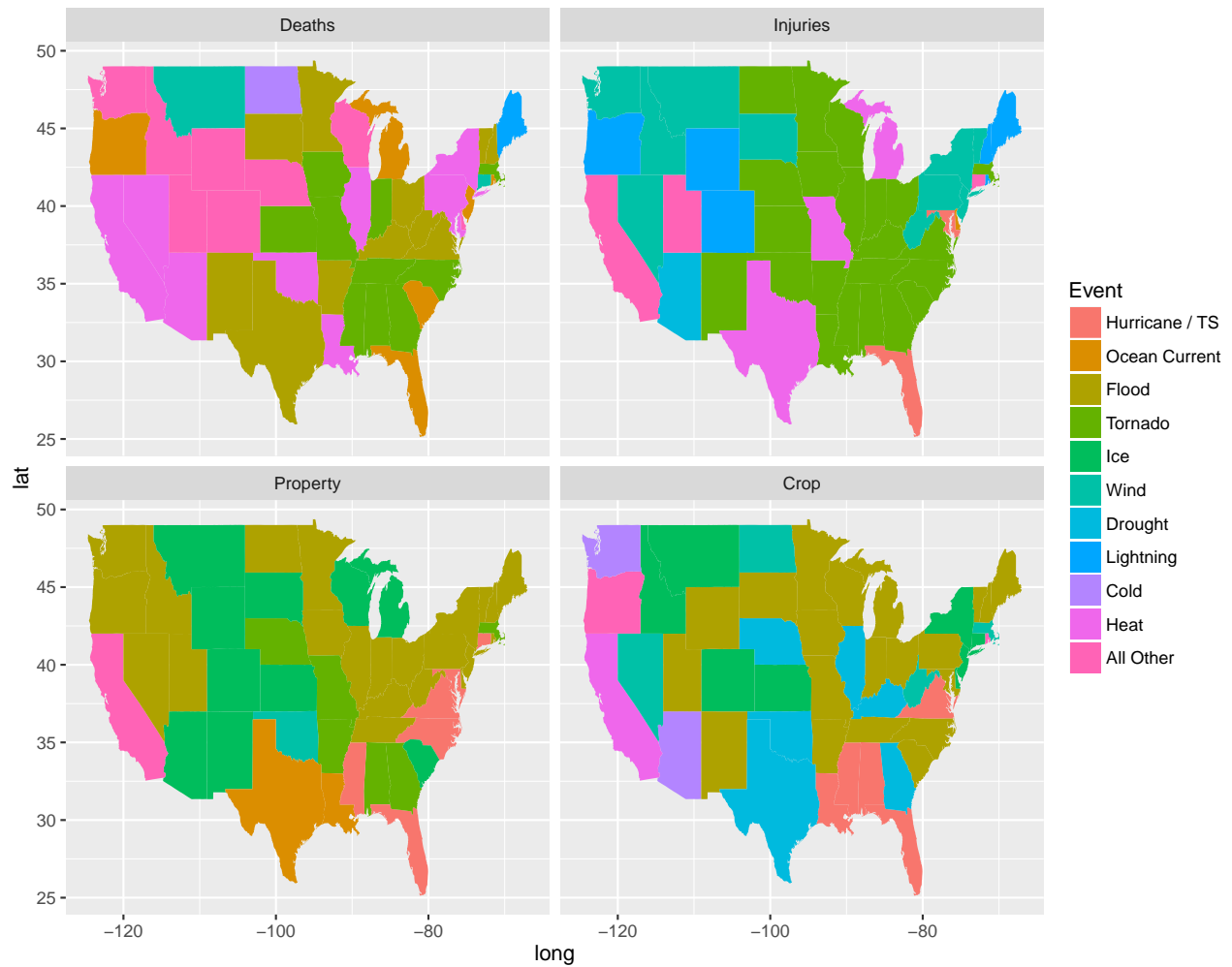
## Need to be able to lookup mapping data by 2-letter abbreviation:
name2abbr <- data.frame( region = tolower(state.name), State = state.abb)
states_map <- inner_join(map_data("state"), name2abbr, by = "region")
## inner_join is needed to keep polygons intact (SO@Richard Careaga)
maxState$State <- as.character(maxState$State)
stateImpact <- inner_join(states_map, maxState, by = "State")

us <- ggplot(data = stateImpact,
             aes(x = long, y = lat, fill = Event, group = group)) +
  geom_polygon() + facet_wrap( "Impact", nrow = 2)

```

Unsurprisingly, each state is exposed to slightly different weather risks. For example, it would be surprising to see blizzards playing a major role in Florida fatalities. The maps below highlight the top threats in each of the four casualty categories in each of the continental 48 states. Only the main national risks are specifically highlighted; “All Other” indicates that the primary risk for that state falls outside those categories.

us



Alignment with known major weather events.

- The [Great Flood of 1993](#) saw the Mississippi inflict 32 fatalities and over \$10bn in losses in the Midwest.
- A major [ice storm in 1994](#) caused hundreds of injuries and billions in losses, including a quarter of Arkansas' pecan industry.
- The [1995 heat wave](#) is visible as a major spike in fatalities.
- In 2005 [Hurricane Katrina](#) wreaked havoc on the eastern seaboard. This is reflected in Hurricane and “Ocean Current” losses of US\$93Bn. These values are likely under-estimates of the actual impact.
- 2011 was an [atypically destructive year for tornadoes](#), with nearly US\$10bn in property damage. This includes the [Super Outbreak](#) in late April that claimed 363 lives, representing over half the fatalities shown in these data.

Major Omissions

- Katrina caused over 1000 fatalities in the U.S. alone, but the data reflect only 150 deaths and few injuries for that time. It is possible that delayed casualties (those occurring days after the causative event) are being left out, even though the NOAA document indicates they should be included (eg injuries from using a kerosene heater due to a power failure)

Thoughts on R / ggplot / etc

- I am dissatisfied with ggplot's handling of outlier information. I would have liked to use a broken axis, but this is apparently [frowned upon by ggplot](#).
- Possible in other packages, like [plotrix](#).
- Wow. The input data have records with un-escaped newlines. The good news is that `read.csv` appears to read these properly if they are properly quoted. The bad news is that it makes grepping in bash effectively impossible.
- Scientific notation (eg `1e+6`) appears to always be considered numeric by R, even if there is no decimal (stops on `read.table` if it occurs in an "integer" column).

Acknowledgements

- The knitr template is based on one devised by [Ron Ammar](#), particularly the `rmarkdown` header and System Information section.
- Ron was generally helpful, particularly for plaintive requests for advice on ggplot.
- StackOverflow:
 - [ggrepel: Automatic handling of nearby geom_text](#) - will "repel" text labels so they don't overlap
 - [Using theme\(\) to place legends inside plot](#)
 - [Using gridExtra::grid.arrange\(\) to organize ggplots](#) - like `par(mfrow = c(1, 2))` but ggplot-friendly
 - [Properly merging data with US map](#) - `merge` disrupts the polygon order, need to use `inner_join` instead

System Information

Time required to process this report: 23.43908 secs

R session information:

```
sessionInfo()
```

```
## R version 3.2.3 (2015-12-10)
## Platform: x86_64-pc-linux-gnu (64-bit)
## Running under: Ubuntu 14.04.4 LTS
##
## locale:
##  [1] LC_CTYPE=en_US.UTF-8      LC_NUMERIC=C
##  [3] LC_TIME=en_US.UTF-8      LC_COLLATE=en_US.UTF-8
##  [5] LC_MONETARY=en_US.UTF-8  LC_MESSAGES=en_US.UTF-8
##  [7] LC_PAPER=en_US.UTF-8     LC_NAME=C
##  [9] LC_ADDRESS=C             LC_TELEPHONE=C
## [11] LC_MEASUREMENT=en_US.UTF-8 LC_IDENTIFICATION=C
##
## attached base packages:
## [1] stats      graphics  grDevices  utils      datasets  methods    base
##
## other attached packages:
## [1] maps_3.1.0      ggrepel_0.5      gridExtra_2.0.0 reshape2_1.4.1
## [5] dplyr_0.4.3     ggplot2_2.0.0    printr_0.0.5     rmarkdown_0.9.2
##
## loaded via a namespace (and not attached):
```

```
## [1] Rcpp_0.12.1      knitr_1.12.3      magrittr_1.5      munsell_0.4.2
## [5] colorspace_1.2-6 R6_2.1.1          stringr_1.0.0     plyr_1.8.3
## [9] tools_3.2.3      parallel_3.2.3    grid_3.2.3        gtable_0.1.2
## [13] DBI_0.3.1        htmltools_0.2.6   lazyeval_0.1.10   yaml_2.1.13
## [17] digest_0.6.8     assertthat_0.1    formatR_1.2.1     evaluate_0.8
## [21] labeling_0.3      stringi_0.5-5     scales_0.3.0
```