Tornadoes vs Hurricanes: Exploratory analysis of the health and economic effect of storm in the United States

Dylan Tweed

June 19, 2016

Contents

1 Synopsis

The U.S. National Oceanic and Atmospheric Administration's (NOAA) storm database provide a large collection of events spanning 1950 to 2011, as well as the import of such events in terms of human life and economical damage. We provide a preliminary analysis in order to distinguish between human life related consequences and economic consequences. We first proceed extracting form the original data set, as many

usable data as possible, and classify the event into 8 main categories. Then we study, for each category, the evolution of the health impact (injuries and fatalities), and economical impact (property and crop damage). We found that Tornadoes have the greatest impact on the population health. Storms cause greater damage, while still causing a large number of fatalities and injuries. Floods causes great damage, but tend to be a lesser risk to the health of the population.

2 Data Processing

The data, we process here contain the following relevant information

- Type of event
- Location
- Date and time at the event start and and
- Impact in human life, fatalities, injuries
- Economical impact

The line on inquiry, we interested in is the distinction of two types of impact human and economical and the correlation with specific type of events.

Interestingly, the kind of weather described here should vary in terms of effects. For example, Tornado should have low duration, and strong wind cause extreme damage locally to the infrastructure. Rain and Snow storm however would be characterized by lower wind speed but last longer, with long term consequence to the infrastructure, floods, increased accidents, lower response time for rescue operations.

In this section we proceed to read the original file, and create a new `data.frame` for the subsequent analysis. Our aim is to extract the relevant information, in the tidiest form possible. The information we gather is the following

1. Starting date of the event
2. Ending date of the event
3. Type of event ideally a factor variable with little number of levels
4. Health impact, including number of injuries and fatality
5. Economical impact, including cost estimate to the damage to the crops and properties.

2.1 Source

Original source located in Storm data

```
sourcedata <- "repdata%2Fdata%2FStormData.csv.bz2"
# reading data inserting NAs for blanks entries
stormData <- read.csv(sourcedata,na.strings = c("", " "))
```

Removing '_' from column names

```
names(stormData) <- gsub("_","",names(stormData))
```

2.2 Processing: Dates

The simplest method is to convert only `BGNDATE` and `ENDDATE` entries. These are well formatted but contains a time set to midnight.

```
# Load lubridate package.
library(lubridate,quietly=TRUE,warn.conflicts=FALSE)
# Look at the data before correction
bgnexist <- !is.na(stormData$BGNDATE)
endexist <- !is.na(stormData$ENDDATE)
# first delete midnight
bgndate <- sub(" 0:00:00","",stormData$BGNDATE,fixed=T)
enddate <- sub(" 0:00:00","",stormData$ENDDATE,fixed=T)
# convert into POSIXct object
bgndate <- mdy(bgndate)
enddate <- mdy(enddate)
```

For this particular analysis we are not interested in the time or duration of specific events. However we present in the Appendix, our method to combine the time and the dates column, taking into account the various format used for the times, and the time zones.

2.3 Processing: Event Type

We plan to compare the impact of different event accordingly to their type. This variable is contained in columns `EVTPE`. We aim to create a factor variable to distinguish the different types of events. Before that, we have a look at the current values.

This entry contains 985 possible factors, some are duplicated, or misspelled. First we shall capitalize everything and correct for spelling within a new variable.

```
# correct evtype by captilizing
unsortedev <- toupper(stormData$EVTYPE)
# correct for spelling
unsortedev <- sub("AVALANCE","AVALANCHE",unsortedev)
unsortedev <- sub("COSTAL","COASTAL",unsortedev)
unsortedev <- sub("DEVEL","DEVIL",unsortedev)
unsortedev <- sub("DRIEST","DRYEST",unsortedev)
unsortedev <- sub("FLOOODING","FLOODING",unsortedev)
unsortedev <- sub("LIGNTNING","LIGHTNING",unsortedev)
unsortedev <- sub("LIGTNING","LIGHTNING",unsortedev)
unsortedev <- sub("LIGHTING","LIGHTNING",unsortedev)
unsortedev <- sub("TEMPERATURES","TEMPERATURE",unsortedev)
unsortedev <- sub("RECORD TEMPERATURE","TEMPERATURE RECORD",unsortedev)
unsortedev <- sub("TORNDO","TORNADO",unsortedev)
unsortedev <- sub("TORNDAO","TORNADO",unsortedev)
unsortedev <- sub("REMNANTS OF FLOYD","HURRICANE FLOYD",unsortedev)
unsortedev <- sub("VOG","FOG",unsortedev)
unsortedev <- sub("WAYTER","WATER",unsortedev)
unsortedev <- sub("WND","WIND",unsortedev)
unsortedev <- sub("WINTRY","WINTER",unsortedev)
unsortedev <- sub("WINTERY","WINTER",unsortedev)
unsortedev <- sub("/"," ",unsortedev)
unsortedev <- sub(" AND "," ",unsortedev)
unsortedev[grep("SUMMARY",unsortedev)] <- "SUMMARY"
```

We have 779 possible factor to consider.

Since we cannot expect to create relevant figure with such a large amount of factor variable, we proceed to re-categorize them. We proceeded by creating a dictionary of keywords, and assign them to a new category.

This list of keywords was incremented, while the spelling corrections updated to minimize the number of non categorized keywords.

```
# set new variable
evsorted <- rep(NA,length(unsortedev))
```

We converged on a non exhaustive list of categories.

1. `Storm/Hurricane/Wind` related to storm, typhoons, hurricanes and high winds
2. `Thunder` related to thunder and thunderstorms
3. `Tornado` related to tornadoes
4. `Flood/Slide/Avalanche` related in floods, land slides, avalanches
5. `Coastal` related to tides, waves, currents
6. `Precipitations` related to precipitations, rain, snow and ice
7. `Temparature Record` related to temperatures variations both hot and cold
8. `Volcanic/Fire` related to volcanic eruptions and forest fires

2.3.1 Storms, Hurricanes and Wind

```
evsorted[grepl("TROPICAL",unsortedev) & is.na(evsorted)] <- "Storm/Hurricane/Wind"
evsorted[grepl("HURRICANE",unsortedev) & is.na(evsorted)] <- "Storm/Hurricane/Wind"
evsorted[grepl("TYPHOON",unsortedev) & is.na(evsorted)] <- "Storm/Hurricane/Wind"
evsorted[grepl("STORM",unsortedev) & is.na(evsorted)] <- "Storm/Hurricane/Wind"
evsorted[grepl("WIND",unsortedev) & is.na(evsorted)] <- "Storm/Hurricane/Wind"
evsorted[grepl("TURBULENCE",unsortedev) & is.na(evsorted)] <- "Storm/Hurricane/Wind"
```

2.3.2 Thunder

```
evsorted[grepl("THUNDER",unsortedev) & is.na(evsorted)] <- "Thunder"
evsorted[grepl("LIGHTNING",unsortedev) & is.na(evsorted)] <- "Thunder"
evsorted[grepl("TSTM",unsortedev) & is.na(evsorted)] <- "Thunder"
```

2.3.3 Tornado

```
evsorted[grepl("WALL CLOUD",unsortedev) & is.na(evsorted)] <- "Tornado"
evsorted[grepl("TORNADO",unsortedev) & is.na(evsorted)] <- "Tornado"
evsorted[grepl("FUNNEL",unsortedev) & is.na(evsorted)] <- "Tornado"
evsorted[grepl("ROTATING",unsortedev) & is.na(evsorted)] <- "Tornado"
evsorted[grepl("GUSTNADO",unsortedev) & is.na(evsorted)] <- "Tornado"
evsorted[grepl("LANDSPOUT",unsortedev) & is.na(evsorted)] <- "Tornado"
evsorted[grepl("WATERSPOUT",unsortedev) & is.na(evsorted)] <- "Tornado"
evsorted[grepl("DUST DEVIL",unsortedev) & is.na(evsorted)] <- "Tornado"
evsorted[grepl("MICROBURST",unsortedev) & is.na(evsorted)] <- "Tornado"
```

2.3.4 Flood, Avalanche, Mud/Rock/Land slides

```
evsorted[grepl("FLOOD",unsortedev) & is.na(evsorted)] <- "Flood/Slide/Avalanche"
evsorted[grepl("FLD",unsortedev) & is.na(evsorted)] <- "Flood/Slide/Avalanche"
evsorted[grepl("DAM",unsortedev) & is.na(evsorted)] <- "Flood/Slide/Avalanche"
evsorted[grepl("STREAM",unsortedev) & is.na(evsorted)] <- "Flood/Slide/Avalanche"
evsorted[grepl("MUD",unsortedev) & is.na(evsorted)] <- "Flood/Slide/Avalanche"
evsorted[grepl("SLIDE",unsortedev) & is.na(evsorted)] <- "Flood/Slide/Avalanche"
evsorted[grepl("SLUMP",unsortedev) & is.na(evsorted)] <- "Flood/Slide/Avalanche"
evsorted[grepl("AVALANCHE",unsortedev) & is.na(evsorted)] <- "Flood/Slide/Avalanche"
```

2.3.5 Volcanic eruptions and Fires

```
evsorted[grepl("VOLCANIC",unsortedev) & is.na(evsorted)] <- "Volcanic/Fire"
evsorted[grepl("SMOKE",unsortedev) & is.na(evsorted)] <- "Volcanic/Fire"
evsorted[grepl("FIRE",unsortedev) & is.na(evsorted)] <- "Volcanic/Fire"
evsorted[grepl("RED FLAG CRITERIA",unsortedev) & is.na(evsorted)] <- "Volcanic/Fire"
```

2.3.6 Coastal

```
evsorted[grepl("MARINE",unsortedev) & is.na(evsorted)] <- "Coastal"
evsorted[grepl("COASTAL",unsortedev) & is.na(evsorted)] <- "Coastal"
evsorted[grepl("WAVE",unsortedev) & is.na(evsorted)] <- "Coastal"
evsorted[grepl("CURRENT",unsortedev) & is.na(evsorted)] <- "Coastal"
evsorted[grepl("TIDE",unsortedev) & is.na(evsorted)] <- "Coastal"
evsorted[grepl("SEAS",unsortedev) & is.na(evsorted)] <- "Coastal"
evsorted[grepl("SURF",unsortedev) & is.na(evsorted)] <- "Coastal"
evsorted[grepl("BEACH",unsortedev) & is.na(evsorted)] <- "Coastal"
evsorted[grepl("SWELLS",unsortedev) & is.na(evsorted)] <- "Coastal"
evsorted[grepl("DROWNING",unsortedev) & is.na(evsorted)] <- "Coastal"
evsorted[grepl("TSUNAMI",unsortedev) & is.na(evsorted)] <- "Coastal"
```

2.3.7 Precipitations: Rain, Snow and Ice

```
evsorted[grepl("RAIN",unsortedev) & is.na(evsorted)] <- "Precipitations"
evsorted[grepl("DOWNBURST",unsortedev) & is.na(evsorted)] <- "Precipitations"
evsorted[grepl("WATER",unsortedev) & is.na(evsorted)] <- "Precipitations"
evsorted[grepl("PRECIP",unsortedev) & is.na(evsorted)] <- "Precipitations"
evsorted[grepl("WET",unsortedev) & is.na(evsorted)] <- "Precipitations"
evsorted[grepl("FOG",unsortedev) & is.na(evsorted)] <- "Precipitations"
evsorted[grepl("HEAVY",unsortedev) & is.na(evsorted)] <- "Precipitations"
evsorted[grepl("SLEET",unsortedev) & is.na(evsorted)] <- "Precipitations"
evsorted[grepl("BLIZZARD",unsortedev) & is.na(evsorted)] <- "Precipitations"
evsorted[grepl("SNOW",unsortedev) & is.na(evsorted)] <- "Precipitations"
evsorted[grepl("ICE",unsortedev) & is.na(evsorted)] <- "Precipitations"
evsorted[grepl("HAIL",unsortedev) & is.na(evsorted)] <- "Precipitations"
evsorted[grepl("FROST",unsortedev) & is.na(evsorted)] <- "Precipitations"
evsorted[grepl("GLAZE",unsortedev) & is.na(evsorted)] <- "Precipitations"
```

```
evsorted[grepl("ICY",unsortedev) & is.na(evsorted)] <- "Precipitations"
evsorted[grepl("MIX",unsortedev) & is.na(evsorted)] <- "Precipitations"
```

2.3.8 Temperature variations

```
evsorted[grepl("TEMPERATURE RECORD",unsortedev) & is.na(evsorted)] <- "Temperature Record"
evsorted[grepl("HYPOTHERMIA",unsortedev) & is.na(evsorted)] <- "Temperature Record"
evsorted[grepl("LOW",unsortedev) & is.na(evsorted)] <- "Temperature Record"
evsorted[grepl("HOT",unsortedev) & is.na(evsorted)] <- "Temperature Record"
evsorted[grepl("DRY",unsortedev) & is.na(evsorted)] <- "Temperature Record"
evsorted[grepl("HEAT",unsortedev) & is.na(evsorted)] <- "Temperature Record"
evsorted[grepl("WARM",unsortedev) & is.na(evsorted)] <- "Temperature Record"
evsorted[grepl("DRYEST",unsortedev) & is.na(evsorted)] <- "Temperature Record"
evsorted[grepl("DUST",unsortedev) & is.na(evsorted)] <- "Temperature Record"
evsorted[grepl("HYPERTHERMIA",unsortedev) & is.na(evsorted)] <- "Temperature Record"
evsorted[grepl("DROUGH",unsortedev) & is.na(evsorted)] <- "Temperature Record"
evsorted[grepl("SEICHE",unsortedev) & is.na(evsorted)] <- "Temperature Record"
evsorted[grepl("HIGH",unsortedev) & is.na(evsorted)] <- "Temperature Record"
evsorted[grepl("WINTER",unsortedev) & is.na(evsorted)] <- "Temperature Record"
evsorted[grepl("COLD",unsortedev) & is.na(evsorted)] <- "Temperature Record"
evsorted[grepl("COOL",unsortedev) & is.na(evsorted)] <- "Temperature Record"
evsorted[grepl("FREEZE",unsortedev) & is.na(evsorted)] <- "Temperature Record"
evsorted[grepl("FREEZING",unsortedev) & is.na(evsorted)] <- "Temperature Record"
```

2.3.9 Unclassified entries

```
unsortlist<- as.data.frame(table(as.factor(unsortedev[is.na(evsorted)])))
names(unsortlist)<-c("EVTYPE","nb")
unsortlist
```

```
##                 EVTYPE nb
## 1                    ?  1
## 2        APACHE COUNTY  1
## 3            EXCESSIVE  1
## 4         MILD PATTERN  1
## 5  MONTHLY TEMPERATURE  4
## 6    NO SEVERE WEATHER  1
## 7                 NONE  2
## 8      NORTHERN LIGHTS  1
## 9                OTHER 52
## 10           SOUTHEAST  1
## 11             SUMMARY 76
## 12         URBAN SMALL  5
```

As far as we can determine, the remaining entries from the `EVTYPE` column are either unclear or irrelevant. We are confident that our keyword search is accurate enough to classify most of the entries.

2.3.10 Resulting classification

```
sortlist<- as.data.frame(table(as.factor(evsorted)))
names(sortlist)<-c("New type","nb of events")
sortlist
```

```
##                New type nb of events
## 1               Coastal         2976
## 2 Flood/Slide/Avalanche        87174
## 3        Precipitations       326194
## 4  Storm/Hurricane/Wind       380395
## 5    Temperature Record        13435
## 6               Thunder        15774
## 7               Tornado        71912
## 8         Volcanic/Fire         4291
```

We see the result of the classification as the number of events. Arguably, our classification is not homogeneous. It could be improved by creating further categories for the `Precipitations` (separating snow and ice from rain) and `Storm\Hurricane\Winds` entries (separating Hurricane and Typhoons).

We note that this sorted may depend on the order in which the selection is made. These categories can overlap with one another, but this classification has the merit of being fairly illustrative. It is also relevant that this categories applies to certain US states and not others.

2.4 Processing: Property damage estimate

The damage costs estimates are given in columns `PROPDMG` and `PROPDMGEXP` for property damages and `CROPDMG` and `CROPDMGEXP`for crop damages. According to the National weather Service Storm Data Documentation , the damages cost units are specified with letters `B` for billion (10^9) dollars damage, `M` for million (10^6) dollars damage and `K` for thousand (10^3) dollars damage. We found these letters in the `PROPDMGEXP` for `CROPDMGEXP` entries.

As for the other columns this is not true for the earliest records.

2.4.1 Property damage `PROPDMG` in units of `PROPDMGEXP`

```
# init the unit aray
unitsexp <- rep(NA,nrow(stormData))
# fetch from stromData the units
unitsexp[grep("[bB]",as.character(stormData$PROPDMGEXP))] <- 1E9
unitsexp[grep("[mM]",as.character(stormData$PROPDMGEXP))] <- 1E6
unitsexp[grep("[kK]",as.character(stormData$PROPDMGEXP))] <- 1E3
# search values with units but no value
noval <- !is.na(unitsexp)& (is.na(stormData$PROPDMG))
# set unit with no values to NA
if(sum(noval) > 0){
    # set to NA
    unitsexp[noval] <- NA
}
# Multiply units by values, if value if 0 result is zero
unitsexp[!is.na(unitsexp)] <- unitsexp[!is.na(unitsexp)] *
```

```
    stormData$PROPDMG[!is.na(unitsexp)]
# Copy the result to a new variable
PropDam <- unitsexp
```

The values of PROPDMGEXP we neglect are:

```
unitsexp[!is.na(unitsexp)] <- unitsexp[!is.na(unitsexp)] * stormData$PROPDMG[!is.na(unitsexp)]
nounits <- as.data.frame(table(as.character(
    stormData$PROPDMGEXP[is.na(unitsexp) & !is.na(stormData$PROPDMG)]
    )))
names(nounits) <- c("PROPDMGEXP","nb")
nounits
```

```
##    PROPDMGEXP  nb
## 1           -   1
## 2           ?   8
## 3           +   5
## 4           0 216
## 5           1  25
## 6           2  13
## 7           3   4
## 8           4   4
## 9           5  28
## 10          6   4
## 11          7   5
## 12          8   1
## 13          h   1
## 14          H   6
```

We could suppose that the numerical values 1-8 could be the base 10 log. It would be simple to add them to the damage entry we create, but we prefer not to do so.

2.4.2 Property damage CROPDMG in units of CROPDMGEXP

```
# init the unit aray
unitsexp <- rep(NA,nrow(stormData))
# fetch from stromData the units
unitsexp[grep("[bB]",as.character(stormData$CROPDMGEXP))] <- 1E9
unitsexp[grep("[mM]",as.character(stormData$CROPDMGEXP))] <- 1E6
unitsexp[grep("[kK]",as.character(stormData$CROPDMGEXP))] <- 1E3
# search values with units but no value
noval <- !is.na(unitsexp)& (is.na(stormData$CROPDMG))
if(sum(noval) > 0){
    # set unit with no values to NA
    unitsexp[noval] <- NA
}
# Multiply units by values, if value if 0 result is zero
unitsexp[!is.na(unitsexp)] <- unitsexp[!is.na(unitsexp)] *
    stormData$CROPDMG[!is.na(unitsexp)]

addval <- !is.na(PropDam) & !is.na(unitsexp)
```

```
newval <- is.na(PropDam) & !is.na(unitsexp)
PropDam[newval] <- unitsexp[newval]
PropDam[addval] <- PropDam[addval] + unitsexp[addval]
rm(list=c("noval","newval","addval"))
```

The values of `CROPDMGEXP` we neglect are:

```
nounits <- as.data.frame(table(as.character(
    stormData$CROPDMGEXP[is.na(unitsexp) & !is.na(stormData$CROPDMG)]
    )))
names(nounits) <- c("CROPDMGEXP","nb")
nounits
```

```
##   CROPDMGEXP nb
## 1          ?  7
## 2          0 19
## 3          2  1
```

2.5 Combining the clean variable into a new tidy `data.frame`

Since we cleaned, the most relevant entries for this study. We now need to combine the results of our calculations into a new `data.frame`. We mainly focus on the `type` of the event, and the date at which the event occurred. For these reason, we select only events with an associated `bgndate` and `type` entry.

```
## First we need the date (time)
## Check for entries with enddate but no bgndate
misbgn <- is.na(bgndate) & !is.na(enddate)
if(sum(misbgn)>0){
    message(paste("Missing start date ",sum(misbgn)))
    # set the begin date as the end date
    bgndate[misbgn] <- enddate[misbgn]
}
## Second, among those we want only events we could classify
mistype    <- is.na(evsorted) & !is.na(bgndate)
outtype    <- !is.na(evsorted) & is.na(bgndate)
fullselect <- !is.na(bgndate) & !is.na(evsorted)
```

Before using this selection for the new `data.frame`, we check the it is as complete as possible. We count the number of recorded entries where victims and properties damage have not assigned values in our selection. We also count the number of event that despite being left out of our selection, have recorded values for these variables.

```
## Check we don't have any issue with fatalisties or injuries
misvictim  <- fullselect & (is.na(stormData$FATALITIES) | is.na(stormData$INJURIES))
outvictim  <- !fullselect & (stormData$FATALITIES > 0 | stormData$INJURIES >0)
## Check we don't have any issue with Property damage
mispropdam <- fullselect & is.na(PropDam)
outpropdam <- !fullselect & !is.na(PropDam)
## prepare a matrix to print out errors
selectcount <- c(
    nrow(stormData),
```

```
    sum(fullselect),
    sum(mistype),
    sum(outtype),
    sum(misvictim) ,
    sum(outvictim) ,
    sum(mispropdam) ,
    sum(outpropdam)
    )
selectcount <- as.matrix(selectcount,ncol=1)
rownames(selectcount) <- c(
    "total",
    "select",
    "na.type",
    "out.type",
    "na.victim",
    "out.victim",
    "na.propdam",
    "out.propdam"
    )
colnames(selectcount) <- "nb of entries"
selectcount
```

```
##             nb of entries
## total              902297
## select             902151
## na.type               146
## out.type                0
## na.victim               0
## out.victim              1
## na.propdam         461826
## out.propdam            37
```

Our selection contains 902151 events out of 902297 entries. Few events 461826 in our selection do not have associated values for the economical impact.

Most worrying however is that 1 entry with recorded with and health impact is left out, and 37 entries with economical impact.

We check that these entries have not been neglected due to our incomplete selection.

```
## Check the original EVTYPE for discarded entries
listtypeerr<-as.data.frame(table(as.character(stormData$EVTYPE[outvictim|outpropdam])))
names(listtypeerr) <-c("EVTYPE","nb of entries")
listtypeerr
```

```
##            EVTYPE nb of entries
## 1               ?             1
## 2   APACHE COUNTY             1
## 3           Other             1
## 4           OTHER            33
## 5 URBAN AND SMALL             1
## 6     URBAN SMALL             1
```

The entries we are missing were indeed mostly classified as `OTHER`. We can now proceed to implement this selection into our new `data.frame`.

```
## Prepare a clean data.frame
cleanStorm <- data.frame(
    state   = stormData$STATE[fullselect],
    county  = stormData$COUNTYNAME[fullselect],
    bgndate = bgndate[fullselect],
    enddate = enddate[fullselect],
    type    = evsorted[fullselect],
    propDam = PropDam[fullselect],
    victim  = stormData$FATALITIES[fullselect]+stormData$INJURIES[fullselect]
)
## Clean up
rm(list=c("mistype","outtype","misvictim","outvictim","mispropdam","outpropdam"))
```

3 Results

3.1 Events harmful to the population health

```
# Load lubridate package.
library(lubridate,quietly=TRUE,warn.conflicts=FALSE)
## compute the total number of victims per type and year
cleanStormvic <- subset(cleanStorm,!is.na(victim))
victimYear <- aggregate(cleanStorm$victim,
    by=list(cleanStormvic$type,year(cleanStormvic$bgndate)),"sum")
names(victimYear) <- c("type","year","victims")
## use the lattice system to plot the result
library(lattice)
xyplot(victims~year|type,data=victimYear,
    layout=c(2,4),
    aspect = "fill",
    ylab = "nb of fatalities or injuries",
    xlab = "year",
    pch=19,
    col="black",
    grid = TRUE,
    alpha=0.5,
    scales = list(y = list(log = 10, equispaced.log = FALSE)),
    ylim =c(1,2e4)
)
```

1. Tornadoes: This type of events seems to cause the largest number of injuries and fatalities. Except for the outliers above 4000 victims/years, there seems the be a trend across the years, with a smaller dispersion and lower number of victims. This might be interpreted as improvement of warning system, or better shelters.
2. Hurricane and Storms also cause a large number of injuries especially from 1990, this may be due to incomplete records. Alternatively, it could suggest that the storms are either stronger and causes more death and injuries, or that the infrastructure is less adapted than in the past for these type of events.

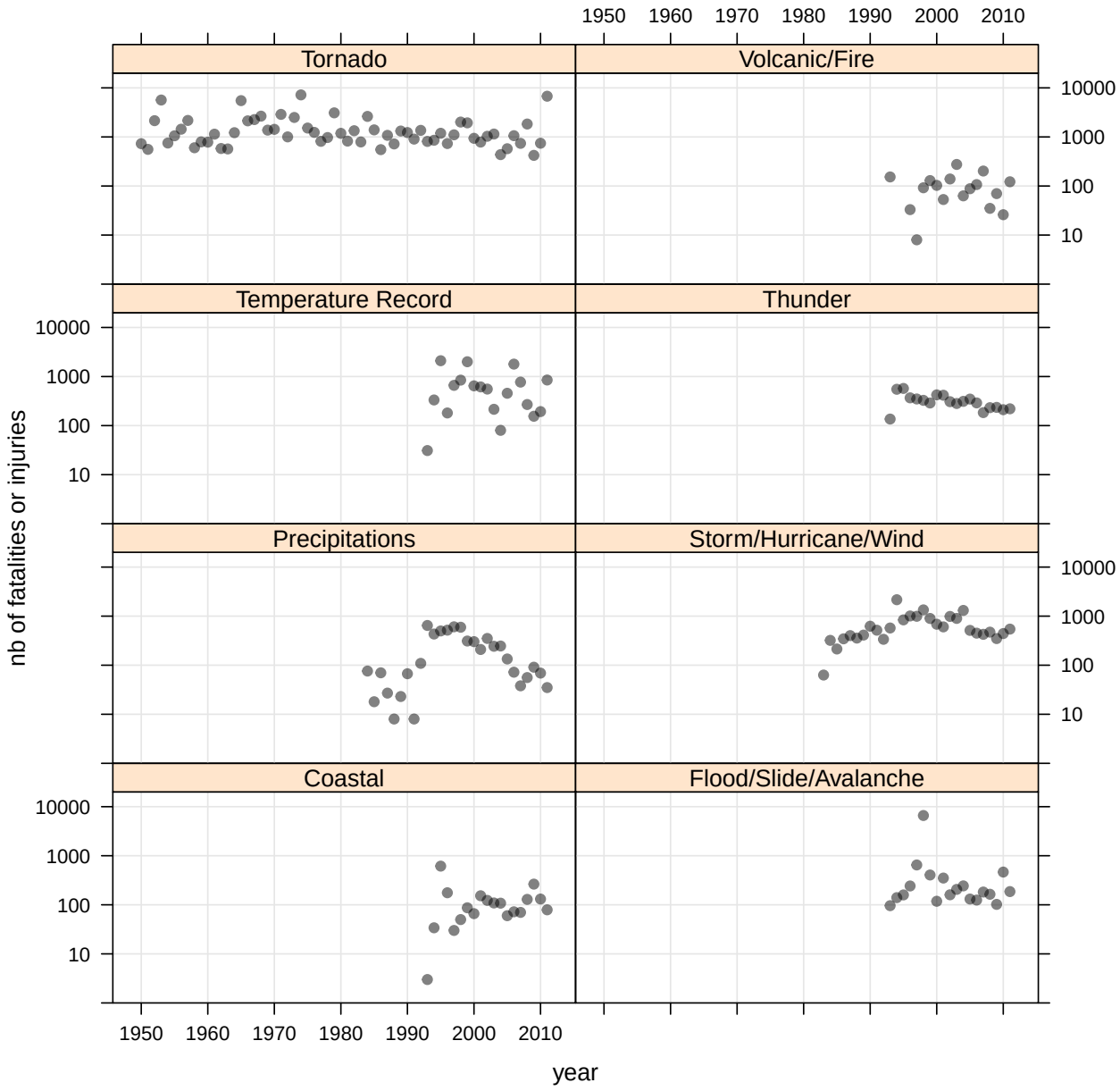

Figure 1: Heath impact of storm events recorded in the United States from 1950 to 2011. We see that Tronados have the greatest impact on the health of the population.

3. The variations in temperatures causes has a less but still significant impact on the population health. A more detailed study should help interpret this result with further distinction between hot and cold periods.
4. On the exception of one year in the `Flood/Slide/Avalanche` category, other type of events appear to have a lesser impact on the health of the population.

3.2 Events harmful to the economy

```
# Load lubridate package.
library(lubridate,quietly=TRUE,warn.conflicts=FALSE)
cleanStormDam <- subset(cleanStorm,!is.na(propDam))
damageYear <- aggregate(cleanStormDam$propDam,
    by=list(cleanStormDam$type,year(cleanStormDam$bgndate)),"sum")
names(damageYear) <- c("type","year","damage")
## use the lattice system to plot the result
library(lattice)
xyplot((damage/1e9)~year|type,data=damageYear,
    layout=c(2,4),
    aspect = "fill",
    ylab = "Damage [Billion $]",
    xlab = "year",
    pch=19,
    col="black",
    alpha=0.5,
    grid = TRUE,
    scales = list(y = list(log = 10, equispaced.log = FALSE)),
    ylim =c(1e-3,2e3)
)
```

1. The recorded Floods, slides and avalanches have the greatest impact as they certainly causes greater damage to the habitation, crops and infrastructures.
2. For similar reason Tornadoes also have a great if slightly lesser economical impact. Over the years, Tornadoes have greater economical impact, but it is not clear whether it is related to the rate of destruction or simply due to monetary inflation or higher damage from Insurance companies.
3. Precipitation and Storm have also a significant impact.
4. Temperature variations, thunderstorms and coastal events (tides currents) have minimal economic impact.

3.3 Cross comparison property against lives

We first select events which had an impact of human lives (fatalities or injuries) and an impact on the economy (property or crop damage).

```
## subsetting data.frame with event having non zero impact on health and economy
cleanStormbad <- subset(
    cleanStorm,
    !is.na(cleanStorm$victim) &
    !is.na(cleanStorm$propDam) &
    cleanStorm$victim> 1 &
    cleanStorm$propDam>1e6
```

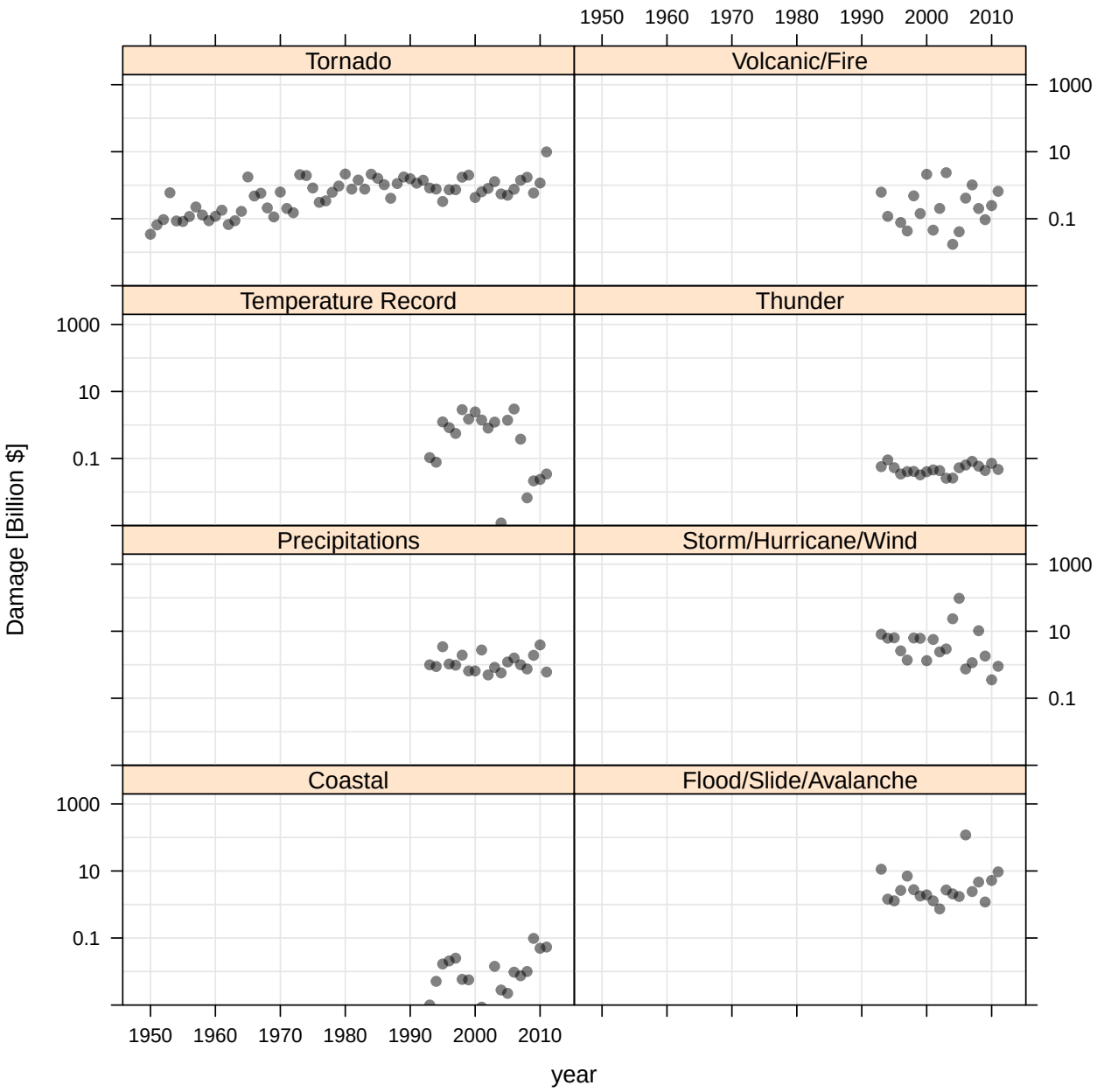

Figure 2: Economic impact of storm events recorded in the United States from 1950 to 2011. Flood, Slides and Avalanches seem to have the largest economical impact.

```
    )
## changing units to million $
cleanStormbad$propDam = cleanStormbad$propDam/1e6
```

We compute the corresponding means for each type of event and each year, into a `data.frame`

```
## Computing mean per type and year
meandamage <- aggregate(cleanStormbad$propDam,
    by=list(cleanStormbad$type,year(cleanStormbad$bgndate)),"mean")
names(meandamage) <- c("type","year","mean.damage")
meanvictim <- aggregate(cleanStormbad$victim,
    by=list(cleanStormbad$type,year(cleanStormbad$bgndate)),"mean")
names(meanvictim) <- c("type","year","mean.victim")
## Keep count of the number of events
count <- aggregate(rep(1,nrow(cleanStormbad)),
    by=list(cleanStormbad$type,year(cleanStormbad$bgndate)),"sum")
names(count) <- c("type","year","count")
## Merging data.frames
meantypeyear <- merge(meanvictim,meandamage,by=c("year","type"))
meantypeyear <- merge(meantypeyear,count,by=c("year","type"))
meantypeyear$count <- log10(meantypeyear$count)
rm(list=c("meanvictim","meandamage","count","cleanStormbad"))
```

Finally we show the scatter of the relation between the health impact on the population and the impact on the economy.

```
library(ggplot2)
ggp <- ggplot(data=meantypeyear,aes(mean.victim,mean.damage)) +
    geom_point(aes(color=type,size=count),alpha=0.6)+
    scale_x_log10()+
    scale_y_log10()+
    labs(
        x="Nb of fatalities or injuries",
        y="Damage [Million $]",
        size="Nb of events [log]"
    ) +
    scale_color_manual(values=c(
        "#707070", # Coastal
        "#404000", # Flood Slides...
        "#207070", # Precipitations
        "#000080", # Storm Hurricanes
        "#900090", # Temperature
        "#009000", # Tornadoes
        "#C00000"  # Volcanic Fire
        ))+
    ## change theme
    theme_bw(base_family="Times")+
    ## Remove box around legend symbols
    theme(legend.key = element_blank())
print(ggp)
```

This figure shows that on average storm and Hurricanes tend to cause the greatest economical impact while having a large impact on the population health. The floods, mud/rock slides and avalanches cause significant

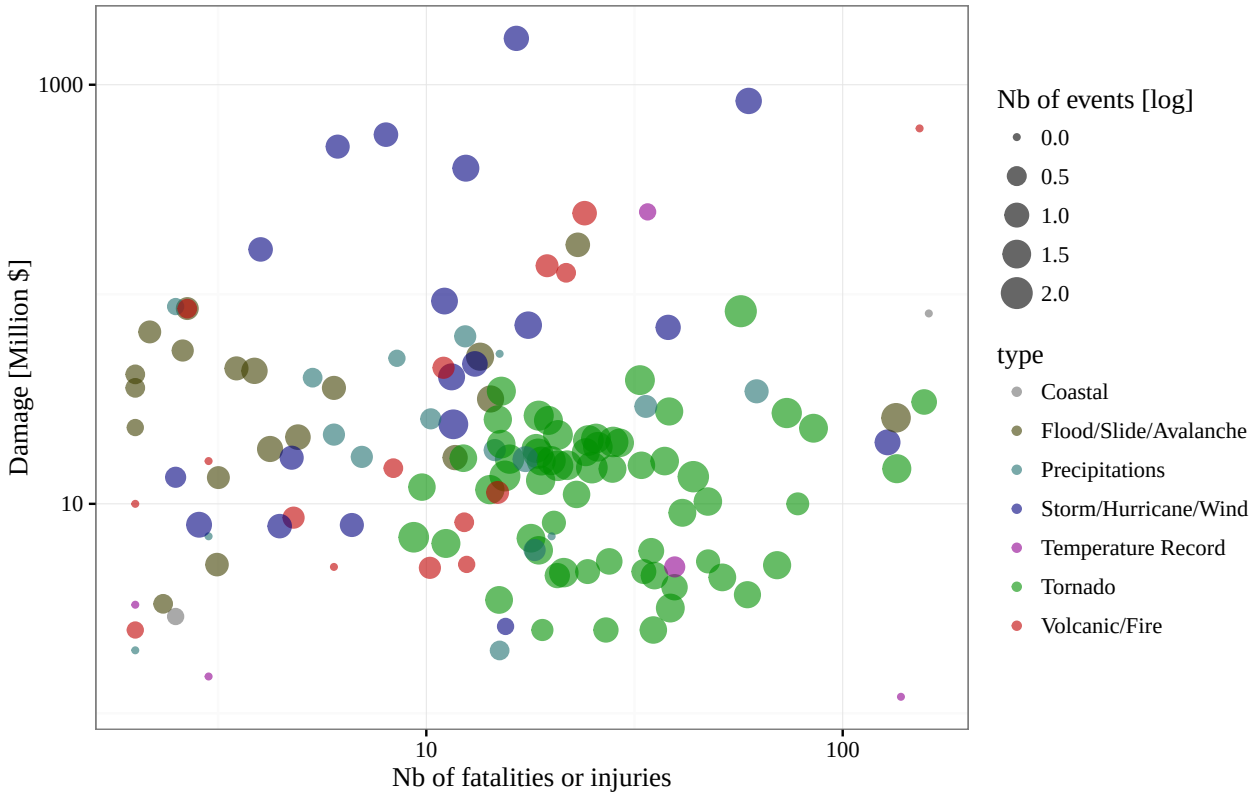

Figure 3: Economical impact againt health impact. Each point corresponds to the means taken over the year of specific types of events indicated by color. The size of the symbol corresponds to the number of events recorded that year.

damage with less impact on the population. We confirm here, that tornadoes tend to cause the largest number of victims, with similar economical impact to the floods. More importantly, tornadoes occurs at a large frequency.

4 Conclusions

We analysed a data set extracted from The U.S. National Oceanic and Atmospheric Administration's (NOAA) storm database. This database dating form 1950, lacks completeness but after careful processing we could produce a new data set, including a classification of the recorded events into 8 categories.

After exploratory analysis of this clean data set we could draw the following conclusion:

- Tornadoes are the most harmful to the health of the population as each event causes a large number of fatalities and injuries and their frequency is higher.
- Storm and Hurricane have great consequence on the economy and the public health.
- Flood Slides and Avalanche follow a similar pattern to the storm with lesser impact on economy and human lives per event.

A Appendix: Processing Dates and Times

Dates of events referred as columns `BGNDATE`, and `ENDDATE` are tidy, they contains a time set to midnight to all, since the times are supposedly contained in the extra columns `BGNTIME` and `ENDTIME` which are not tidy.

The problem are the following:

- Date are written with a time set to midnight, while the time is written as a separate entries. Dates seems to be always written in the `%m/%d/%Y %h:%m:%s` format, however in this entry the time is set to midnight.
- Times `BGNTIMES` are written in multiple formats
  - `%h%m`
  - `%h:%m:%s [AP]M`
- Times `ENDTIMES` are written in multiple formats
  - `%h%m`
  - `%h%m%tz`
  - `%h%m %tz`
  - `%h:%m:%s %p`
  - `%h:%m:%s %p %tz`
  - text

This is a relevant problem as only the end of the data set contains dates and time formatted in a practical manner. We aim to automatically correct for this issue, to increase the sample of usable dates and time, and potentially extract the duration of event containing both `BGNDATE, BGNTIME` and `ENDDATE, ENDTIME` entries.

A.1 Processing dates

```
# Load lubridate package.
library(lubridate,quietly=TRUE,warn.conflicts=FALSE)
# Look at the data before correction
bgnexist <- !is.na(stormData$BGNDATE)
endexist <- !is.na(stormData$ENDDATE)
# first delete midnight
bgndate <- sub(" 0:00:00","",stormData$BGNDATE,fixed=T)
enddate <- sub(" 0:00:00","",stormData$ENDDATE,fixed=T)
```

A.2 Processing time zones

```
# first put them upper case
stormData$TIMEZONE <- toupper(stormData$TIMEZONE)
# correct for mispelled or unknown timezones
stormData$TIMEZONE <- sub("CSC","CST",stormData$TIMEZONE)
stormData$TIMEZONE <- sub("UNK","CST",stormData$TIMEZONE)
stormData$TIMEZONE <- sub("SCT","CST",stormData$TIMEZONE)
stormData$TIMEZONE <- sub("ESY","EST",stormData$TIMEZONE)
```

The time zone abbreviations are not necessarily recognized by `R` with the `lubridate` package. We created a simple `data.frame` to correct for the timezone in `R`, that we save into csv file `UStimezones.csv`. Source of the data if Time Zone US.

```
# Load lubridate package.
library(lubridate,quietly=TRUE,warn.conflicts=FALSE)
UStz <- read.csv("UStimezones.csv")
row.names(UStz) <- UStz$Abbreviation
UStz$Abbreviation <- NULL
# convert into time
UStz$Offset.standard<-hms(UStz$Offset.standard)
UStz$Offset.daylight<-hms(UStz$Offset.daylight)
UStz
```

```
##                       Time.zone.name Offset.standard Offset.daylight
## ADT           Atlantic Daylight Time       -3H 0M 0S            <NA>
## HAA           Atlantic Daylight Time       -3H 0M 0S            <NA>
## AKDT            Alaska Daylight Time       -8H 0M 0S            <NA>
## AKST            Alaska Standard Time       -9H 0M 0S            <NA>
## AST           Atlantic Standard Time       -4H 0M 0S            <NA>
## HNA           Atlantic Standard Time       -4H 0M 0S            <NA>
## AT                     Atlantic Time       -4H 0M 0S       -3H 0M 0S
## CDT            Central Daylight Time       -5H 0M 0S            <NA>
## HAC            Central Daylight Time       -5H 0M 0S            <NA>
## CST            Central Standard Time       -6H 0M 0S            <NA>
## HNC            Central Standard Time       -6H 0M 0S            <NA>
## CT                      Central Time       -6H 0M 0S       -5H 0M 0S
## EDT            Eastern Daylight Time       -4H 0M 0S            <NA>
## HAE            Eastern Daylight Time       -4H 0M 0S            <NA>
## EGST   Eastern Greenland Summer Time              0S            <NA>
## EGT              East Greenland Time       -1H 0M 0S            <NA>
## EST            Eastern Standard Time       -5H 0M 0S            <NA>
```

```
## HNE            Eastern Standard Time       -5H 0M 0S            <NA>
## ET                     Eastern Time       -5H 0M 0S       -4H 0M 0S
## GMT              Greenwich Mean Time              0S            <NA>
## UTC              Greenwich Mean Time              0S            <NA>
## GST      South Georgia Standard Time       -2H 0M 0S            <NA>
## HADT   Hawaii-Aleutian Daylight Time       -9H 0M 0S            <NA>
## HAST   Hawaii-Aleutian Standard Time      -10H 0M 0S            <NA>
## HST    Hawaii-Aleutian Standard Time      -10H 0M 0S            <NA>
## MDT           Mountain Daylight Time       -6H 0M 0S            <NA>
## HAR           Mountain Daylight Time       -6H 0M 0S            <NA>
## MST           Mountain Standard Time       -7H 0M 0S            <NA>
## HNR           Mountain Standard Time       -7H 0M 0S            <NA>
## MT                     Mountain Time       -7H 0M 0S       -6H 0M 0S
## NDT       Newfoundland Daylight Time      -2H 30M 0S            <NA>
## HAT       Newfoundland Daylight Time      -2H 30M 0S            <NA>
## NST       Newfoundland Standard Time      -3H 30M 0S            <NA>
## HNT       Newfoundland Standard Time      -3H 30M 0S            <NA>
## PDT            Pacific Daylight Time       -7H 0M 0S            <NA>
## HAP            Pacific Daylight Time       -7H 0M 0S            <NA>
## PMDT Pierre & Miquelon Daylight Time       -2H 0M 0S            <NA>
## PMST Pierre & Miquelon Standard Time       -3H 0M 0S            <NA>
## PST            Pacific Standard Time       -8H 0M 0S            <NA>
## HNP            Pacific Standard Time       -8H 0M 0S            <NA>
## PT                     Pacific Time       -8H 0M 0S       -7H 0M 0S
## SST              Samoa Standard Time      -10H 0M 0S            <NA>
## WGST   Western Greenland Summer Time       -2H 0M 0S            <NA>
## WGT              West Greenland Time       -3H 0M 0S            <NA>
```

The resulting data frame is printed in the appendix, for reference.

A.3 Processing `BGNTIME`

```
# copy time
evtime <- stormData$BGNTIME
evtime[!bgnexist] <- NA
```

A.3.1 Computing time convertion into UTC

We use the `data.frame` we created earlier to compute the time conversion from the `TIMEZONE` entry.

```
# Init time convertion
timeshift <- hms(rep("0:00:00",nrow(stormData)))
# First deal with the timezone
for(tz_loc in unique(stormData$TIMEZONE)) {
    selecttz <- stormData$TIMEZONE==tz_loc
    timeshift[selecttz] <- -1*UStz[tz_loc,"Offset.standard"]
}
rm(list=c("tz_loc","selecttz"))
```

Additionally we take into account that some entries may use the AM/PM standard. We detect the times coded as PM format and add 12 hours to the time conversion.

```
# remove spaces
evtime <- gsub(" ","",evtime)
# now lets look at the PM/AM issue
isPM <- grepl("PM",evtime) & !grepl("^12",evtime)
# now let's remove the PM/AM
evtime <- sub("[PA]M","",evtime)
# correct timeshift
timeshift[isPM] <- timeshift[isPM] + hms("12:00:00")
rm(isPM)
```

We remove the AM/PM characters from the working variable.

A.3.2 Correcting for spelling

We find several issues, that would break down the parsing by `lubridate`

- Entries where midnight is written as 2400 (we didn't increment the date by 1 day)
- Character `0` written as `O`
- 4 entries with number of minutes greater than 59. Interpreted as a likely permutation of the last 2 digits.

```
# Replace 2400 by 0000
evtime <- sub("2400","0000",evtime,fixed=T)
# Replace O by 0
evtime <- sub("O","0",evtime)
# Check minute permutation and replace them
worsetime <- grep("^[0-9][0-9][7-9][0-5]",evtime)
sb1 <- substring(evtime[worsetime],4)
sb2 <- substring(evtime[worsetime],3)
substring(evtime[worsetime],4) <-sb2
substring(evtime[worsetime],3) <-sb1
rm(list=c("sb1","sb2"))
rm(worsetime)
```

A.3.3 Reformatting times

We find time written as `%h%m` and reformat them as `%h:%m:%s`

```
# find time not formatted properly
badform <-grep("^[0-2][0-9][0-5][0-9]",evtime)
if(length(badform) > 0){
    # Reformat times
    evtime[badform] <- paste(
        substring(evtime[badform],1,2),
        substring(evtime[badform],3,4),
        "00",
        sep=":"
    )
}
rm(badform)
```

A.3.4 Resetting remaining times

Still after these correction some time may not be formatted properly still, we set them to midnight UTC

```
# set time to midnight is none was provided or format could not be c
notime <- bgnexist & (is.na(evtime) | !grepl(":",evtime))
evtime[notime] <- "0:00:00"
timeshift[notime] <- hms("0:00:00")
toreset <-as.character(stormData$BGNTIME[notime])
resettime <- data.frame(table(as.factor(toreset)))
names(resettime) <- c("BGNTIME","nb")
resettime
```

```
##   BGNTIME nb
## 1     000  1
## 2    9999  1
```

The only entry we could not interpret was set to 9999. Since we combine both date and time into one variable to be converted into a POSIXct object , we prefer to use midnight as default (note: we also reset the time conversion).

A.3.5 Converting to POSIXct object

```
# now add our time to the date
bgndate[bgnexist] <- paste(bgndate[bgnexist],evtime[bgnexist])
# convert into POSIXct object
bgndate <- mdy_hms(bgndate,tz="UTC")
# take timezone into account and PM shif
bgndate[bgnexist] <- bgndate[bgnexist] + timeshift[bgnexist]
# Cleaning temporary vartaibles
rm(timeshift)
rm(evtime)
stormData$bgn <- bgndate
head(stormData[bgnexist,c("BGNDATE","BGNTIME","TIMEZONE","bgn")])
```

```
##              BGNDATE BGNTIME TIMEZONE                 bgn
## 1  4/18/1950 0:00:00    0130      CST 1950-04-18 07:30:00
## 2  4/18/1950 0:00:00    0145      CST 1950-04-18 07:45:00
## 3  2/20/1951 0:00:00    1600      CST 1951-02-20 22:00:00
## 4   6/8/1951 0:00:00    0900      CST 1951-06-08 15:00:00
## 5 11/15/1951 0:00:00    1500      CST 1951-11-15 21:00:00
## 6 11/15/1951 0:00:00    2000      CST 1951-11-16 02:00:00
```

```
tail(stormData[bgnexist,c("BGNDATE","BGNTIME","TIMEZONE","bgn")])
```

```
##                   BGNDATE     BGNTIME TIMEZONE                 bgn
## 902292 11/28/2011 0:00:00 03:00:00 PM      CST 2011-11-28 21:00:00
## 902293 11/30/2011 0:00:00 10:30:00 PM      MST 2011-12-01 05:30:00
## 902294 11/10/2011 0:00:00 02:48:00 PM      MST 2011-11-10 21:48:00
## 902295  11/8/2011 0:00:00 02:58:00 PM      AKS 2011-11-08 23:58:00
## 902296  11/9/2011 0:00:00 10:21:00 AM      AKS 2011-11-09 19:21:00
## 902297 11/28/2011 0:00:00 08:00:00 PM      CST 2011-11-29 02:00:00
```

We successfully, combined the date and time BGNDATE,BGNTIME into a single time variable set to the UTC timezone

A.4 Processing ENDTIME

```
# copy time make all letter upper case
evtime <- toupper(stormData$ENDTIME)
evtime[!endexist] <- NA
```

Since this entry contains multiples letter, PM/AM, or timezone we set them all to upper case strait from the start.

A.4.1 Computing time convertion into UTC

We first proceed as before

```
# Init time convertion
timeshift <- hms(rep("0:00:00",nrow(stormData)))
# First deal with the timezone
for(tz_loc in unique(stormData$TIMEZONE)) {
    selecttz <- stormData$TIMEZONE==tz_loc
    timeshift[selecttz] <- -1*UStz[tz_loc,"Offset.standard"]
}
```

For this column however the timezone is often encoded. It can also differ from the entry TIMEZONE, we detect those case and correct the time conversion accordingly. We also erase the corresponding character form our working variable

```
# Take into account the timezone written in the time
for(tz_loc in unique(stormData$TIMEZONE)) {
    selectbadtz <- grepl(tz_loc,evtime) & stormData$TIMEZONE!=tz_loc
    if(sum(selectbadtz) > 0 ) {
        timeshift[selectbadtz] <- -1*UStz[tz_loc,"Offset.standard"]
    }
    selecttz <- grepl(tz_loc,evtime)
    # Remove the timezone we find in order to avoid error
    evtime <- sub(tz_loc,"",evtime)
}
rm(list=c("tz_loc","selecttz"))
```

Finally, as before, we detect the times coded as PM format and add 12 hours to the time conversion.

```
# remove spaces
evtime <- gsub(" ","",evtime)
# now lets look at the PM/AM issue
isPM <- grepl("PM",evtime) & !grepl("^12",evtime)
# correct timeshift
timeshift[isPM] <- timeshift[isPM] + hms("12:00:00")
rm(isPM)
# now let's remove the PM/AM
evtime <- sub("[PA]M","",evtime)
```

We remove the AM/PM character from the working variable.

A.4.2 Correcting for spelling

We find several issues, that would break down the parsing by `lubridate`

- Entries where midnight is written as 2400 (we didn't increment the date by 1 day)
- Character `O` written as `O`
- Time written as 3 character instead of 4. Example: `300` is translated in `0300`
- 0 entries with number of minutes greater than 59. Interpreted as a likely permutation of the last 2 digits.
- 1 entries with number of minutes greater than 24. Interpreted as a likely permutation of the first 2 digits.

```
# Replace 2400 by 0000
evtime <- sub("2400","0000",evtime,fixed=T)
# Replace O by 0
evtime <- sub("O","0",evtime)
# Correct missing 0
badform <- grep("^[0-9][0-9][0-9]$",evtime)
evtime[badform] <- paste0("0",evtime[badform])
print(paste("Missing 0:",length(badform)))
```

```
## [1] "Missing 0: 3"
```

```
# Check minute permutation and replace them
worsetime <- grep("^[0-9][0-9][7-9][0-5]",evtime)
sb1 <- substring(evtime[worsetime],4)
sb2 <- substring(evtime[worsetime],3)
substring(evtime[worsetime],4) <-sb2
substring(evtime[worsetime],3) <-sb1
rm(list=c("sb1","sb2"))
rm(worsetime)
# Check hour digits permutation and replace them
worsetime <- grep("^[3-9][0-2][0-5][0-9]",evtime)
sb1 <- substring(evtime[worsetime],1)
sb2 <- substring(evtime[worsetime],2)
substring(evtime[worsetime],1) <-sb2
substring(evtime[worsetime],2) <-sb1
rm(list=c("sb1","sb2"))
rm(worsetime)
```

This code is similar to the previous version but include extra corrections, and should be used as default.

A.4.3 Reformatting times

Exactly as before, we find time written as `%h%m` and reformat them as `%h:%m:%s`, and set the default time to midnight.

```
# find time not formatted properly
badform <-grep("^[0-2][0-9][0-5][0-9]",evtime)
if(length(badform) > 0){
    # Reformat times
    evtime[badform] <- paste(
```

```
        substring(evtime[badform],1,2),
        substring(evtime[badform],3,4),
        "00",
        sep=":"
    )
}
rm(badform)
# set time to midnighst is none was provided
notime <- endexist & (is.na(evtime) | !grepl(":",evtime))
evtime[notime] <- "0:00:00"
timeshift[notime] <- hms("0:00:00")
toreset <-as.character(stormData$ENDTIME[notime])
resettime <- data.frame(table(as.factor(toreset)))
names(resettime) <- c("BGNTIME","nb")
resettime
```

```
##         BGNTIME nb
## 1             ?  7
## 2          ?CST  2
## 3          ?EST  2
## 4            05  1
## 5     Afternoon  1
## 6           All  1
## 7       All Day  4
## 8            AM  4
## 9        AM MST  1
## 10         even  1
## 11      Evening  1
## 12 Late Afterno  1
## 13        Month  1
## 14      Morning  6
## 15        night  1
## 16    Overnight  1
## 17           PM  6
## 18       PM MST  1
```

We see in that case that the time we could not interpret where either `?` or a description of the day period.

A.4.4 Converting to `POSIXct` object

```
# now add our time to the date
enddate[endexist] <- paste(enddate[endexist],evtime[endexist])
# convert into POSIXct object
enddate <- mdy_hms(enddate,tz="UTC")
# take timezone into account and PM shif
enddate[endexist] <- enddate[endexist] + timeshift[endexist]
rm(timeshift)
rm(evtime)
stormData$end <- enddate
print(head(stormData[endexist,c("ENDDATE","ENDTIME","TIMEZONE","end")]))
```

```
##                   ENDDATE ENDTIME TIMEZONE                 end
## 187561 1/23/1995 0:00:00 0800CST      CST 1995-01-23 14:00:00
## 187562 2/10/1994 0:00:00 2300CST      CST 1994-02-11 05:00:00
## 187563  2/8/1995 0:00:00 0500CST      CST 1995-02-08 11:00:00
## 187564 3/13/1993 0:00:00 1200CST      CST 1993-03-13 18:00:00
## 187565 2/12/1995 0:00:00 0400CST      CST 1995-02-12 10:00:00
## 187566 10/5/1995 0:00:00 1000CST      CST 1995-10-05 16:00:00
```

```
print(tail(stormData[endexist,c("ENDDATE","ENDTIME","TIMEZONE","end")]))
```

```
##                    ENDDATE     ENDTIME TIMEZONE                 end
## 902292 11/29/2011 0:00:00 12:00:00 PM      CST 2011-11-29 18:00:00
## 902293 11/30/2011 0:00:00 10:30:00 PM      MST 2011-12-01 05:30:00
## 902294 11/10/2011 0:00:00 02:48:00 PM      MST 2011-11-10 21:48:00
## 902295  11/9/2011 0:00:00 01:15:00 PM      AKS 2011-11-09 22:15:00
## 902296  11/9/2011 0:00:00 05:00:00 PM      AKS 2011-11-10 02:00:00
## 902297 11/29/2011 0:00:00 04:00:00 AM      CST 2011-11-29 10:00:00
```

Once again, we compensated for inconsistent formatting and successfully, combined the date and time `ENDDATE`,`ENDTIME` into a single time variable set to the UTC timezone

B Appendix: Available Scripts

The codes described in the document are publicly available in BreizhZut/RepData_PeerAssessment2

This directory contains the additional csv file created for this project: `UStimezones.csv`. This find associate for multiple time zones:

- its abbreviation (first column)
- its name (second columns),
- and the corresponding time different to the UTC timezone
  - for standard time (third column)
  - for daylight saving (fourth column) if relevant

B.1 Processing scripts

Script are the following:

- `readStormData.R` if the `StormData` `data.frame` has not been loaded read file repdata%2Fdata%2FStormData.csv.bz2 into `data.frame` `StormData`
- `readUStimezones.R` read `data.frame` `UStz` from file `UStimezones.csv` created for this project. Requires package `lubridate`
- `fixStormTimes.R` uses columns `BGNDATE`, `BGNTIME`,`ENDNDATE`, `ENDTIME` and `TIMEZONE` of `data.frame` `StormData` to create:
  - `POSIXct` vector `bgndate`
  - `POSIXct` vector `enddate`
  - Require `readUStimezones.R` and package `lubridate`
- `fixStormEventsType.R` uses column `EVTYPE` of `data.frame` `StormData` and re-categorizes events in new character vector `evsorted`

- `fixStormPropDamage.R` uses columns `PROPDMG`, `PROPDMGEXP`, `CROPDMG`, `CROPDMGEXP` of `data.frame StormData` to create a numeric vector `PropDam` corresponding to the damage estimate.
- `cleanStormData.R` the main script creates `data.frame cleanStorm`
  - calls `readStormData.R`
  - calls `fixStormTimes.R` if `bgndate` and `enddate` have not been created
  - calls `fixStormEventsType.R` if `evsorted` has not been created
  - calls `fixStormPropDamage.R` if `PropDam` has not been created
  - collect these vectors into a new `data.frame cleanStorm`
  - clear memory of vectors `bgndate`, `enddate`, `evsorted` and `PropDam`. If one of the previous script was use independently, this script won't rerun the full analysis, on the first call, but will on the second.

B.2 Analysis scripts

The figure can be regenerated using the following script

- `plot_victims_per_type.R` (figure 1)
  - calls `cleanStormData.R` if the tidy data set `cleanStorm` has not been created
  - displays a panel figure of the total number of victim per year as a function of time
  - requires packages `lubridate` and `lattice`
- `plot_damage_per_type.R` (figure 2)
  - calls `cleanStormData.R` if the tidy data set `cleanStorm` has not been created
  - displays a panel figure of the total number damage estimate per year as a function of time
  - requires packages `lubridate` and `lattice`
- `plot_victim_damage.R` (figure 3)
  - calls `cleanStormData.R` if the tidy data set `cleanStorm` has not been created
  - displays a scatter plot of the average number of damage opposed to the damage estimate
  - requires packages `ggplot2` and `lattice`