R and Google Maps

Jim Thompson

August 2014

Creating Google Map in R

This paper demonstrates use of R packages **ggmap**[1] and related packages for visualizing and analyzing spatial data. **ggmap** provides functions to visualize spatial data on top of maps built on top of Google Maps, OpenStreetMaps, Stamen Maps, or CloudMade Maps.

Following code demonstrates geocoding an addresse using the **ggmap** function **geocode()**.

```r
###
# Geocode and map address
###
library(ggmap)

address.of.interest <- "8250 Jones Branch Dr., McLean, VA"

# call Google web service API to geocode the address
location <- geocode(address.of.interest,output="more")

# show resulting geocoded address
cat("property is located at lon=",location[1,1],", lat=",location[1,2],"\n")

## property is located at lon= -77.23 , lat= 38.93
```

Following code fragment draws and annotates a Google map centered on the specified location.

```r
# create Google map centered on address
this.map <- get_map(address.of.interest,16)

# draw map and annotate
png("./figures/sample_map.png")
ggmap(this.map) +
    # plot plot
    geom_point(aes(x=lon, y=lat), data=location, shape=4, size=5, color="red") +

    # label point
    geom_text(aes(x=lon, y=lat), data= location, label=address.of.interest,
              size=4,
              hjust=0.5, vjust=1.5,
              color="red", fontface="bold")
dev.off()
```

[1]Kahle, D. and Wickham, H., **ggmap: Spatial Visualization with ggplot2**, *The R Journal*, Vol. 5/1, http://journal.r-project.org/archive/2013-1/kahle-wickham.pdf

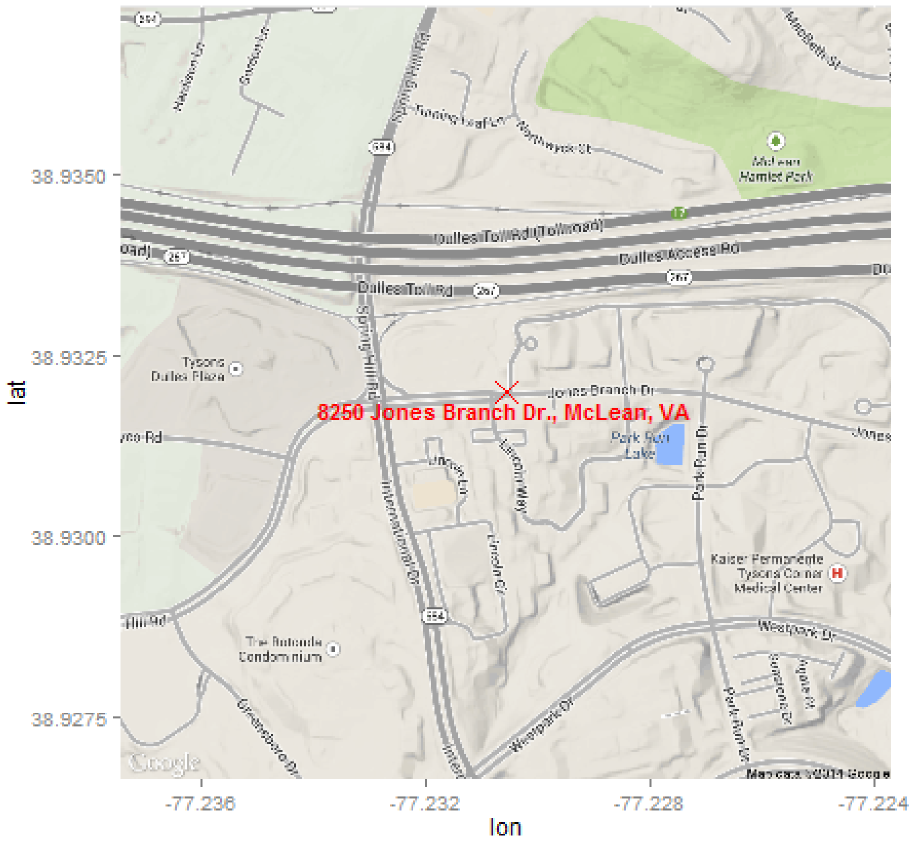

Figure 1: Sample Google Map

Use Case

Using hurricane forecast geospatial data from the National Hurrican Center, this use case determines properties within forecasted path of the storm. For this example, the October 28, 2012 forecast data for Hurricane Sandy is used.

Simulated Property Data

85 simulated properties were randomly placed in 10 Virginia administrative areas. Property values for the simulated properties are uniform psuedo-random values between $50,000 and $200,000. The total value of the simulate properties is $9,381,251. Figure 2 show locations of the simulated properties.

Storm Path Analysis

R packages used for this analysis

- **ggmap**
- **maptools**

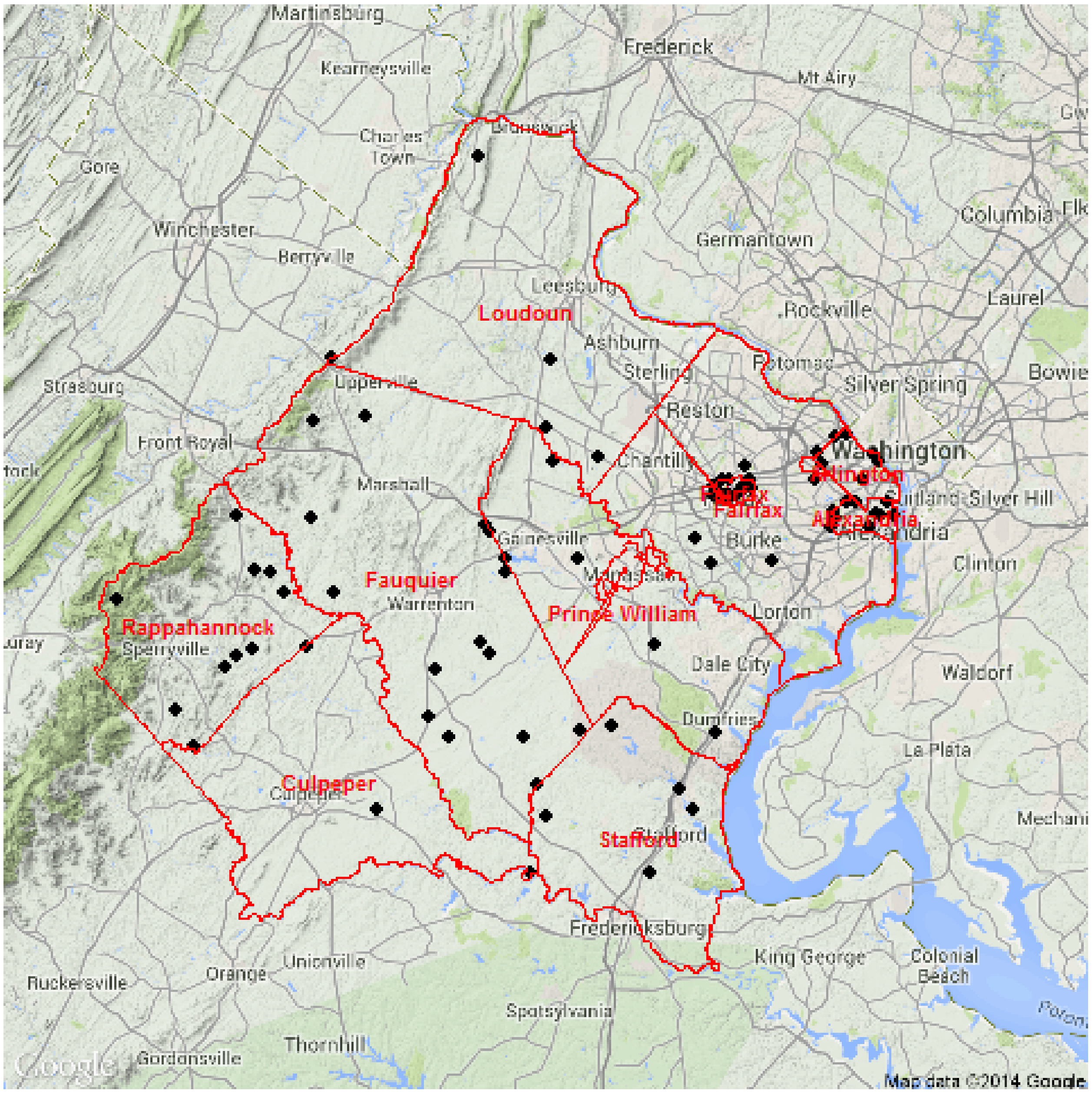

Figure 2: Simulated Property Locations

- **rgeos**
- **raster**

Steps taken for the storm are analysis.

- Simulated property data loaded in **SpatialPointsDataFrame** structure.
- State and County boundaries from Census Bureau (http://www.census.gov/geo/maps-data/data/tiger-line.html) loaded into **SpatialPolygonsDataFrame** structures.
- Storm path data, which are provided as shapefile, from the National Hurricane Center (http://www.nhc.noaa.gov/gis/) loaded in **SpatialPolygonsDataFrame** structure. Yellow area in Figure 3 shows the 72-hour forecast for the possible areas that will be impacted by Hurricane Sandy. Figure 4 shows possible impacted areas within the mid-Atlantic region.
- Function **over()** from the package **sp** determines which properties lie within the storm path polygon structures. Figure 5 shows areas in the Washington, D.C. area potentially in the path of Sandy. Properties in the hurricane path are show in red. Out of the 85 properties in the study, we find that 48 properties are in the projected storm path. These properties account for \$4,026,768 out of the total \$9,381,251.

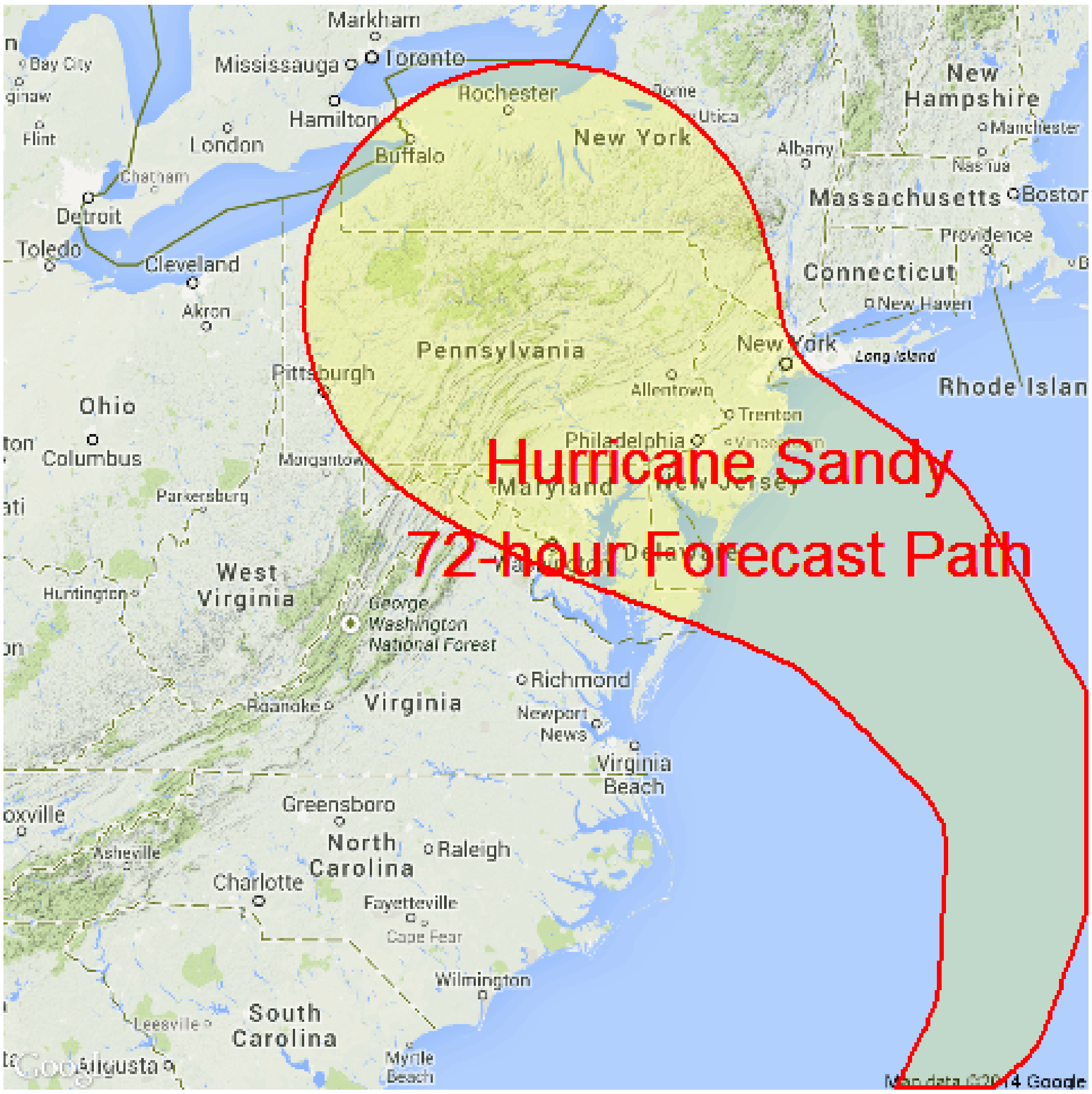

Figure 3: Hurricane Sandy Forecast Path (East Coast), October 28, 2012

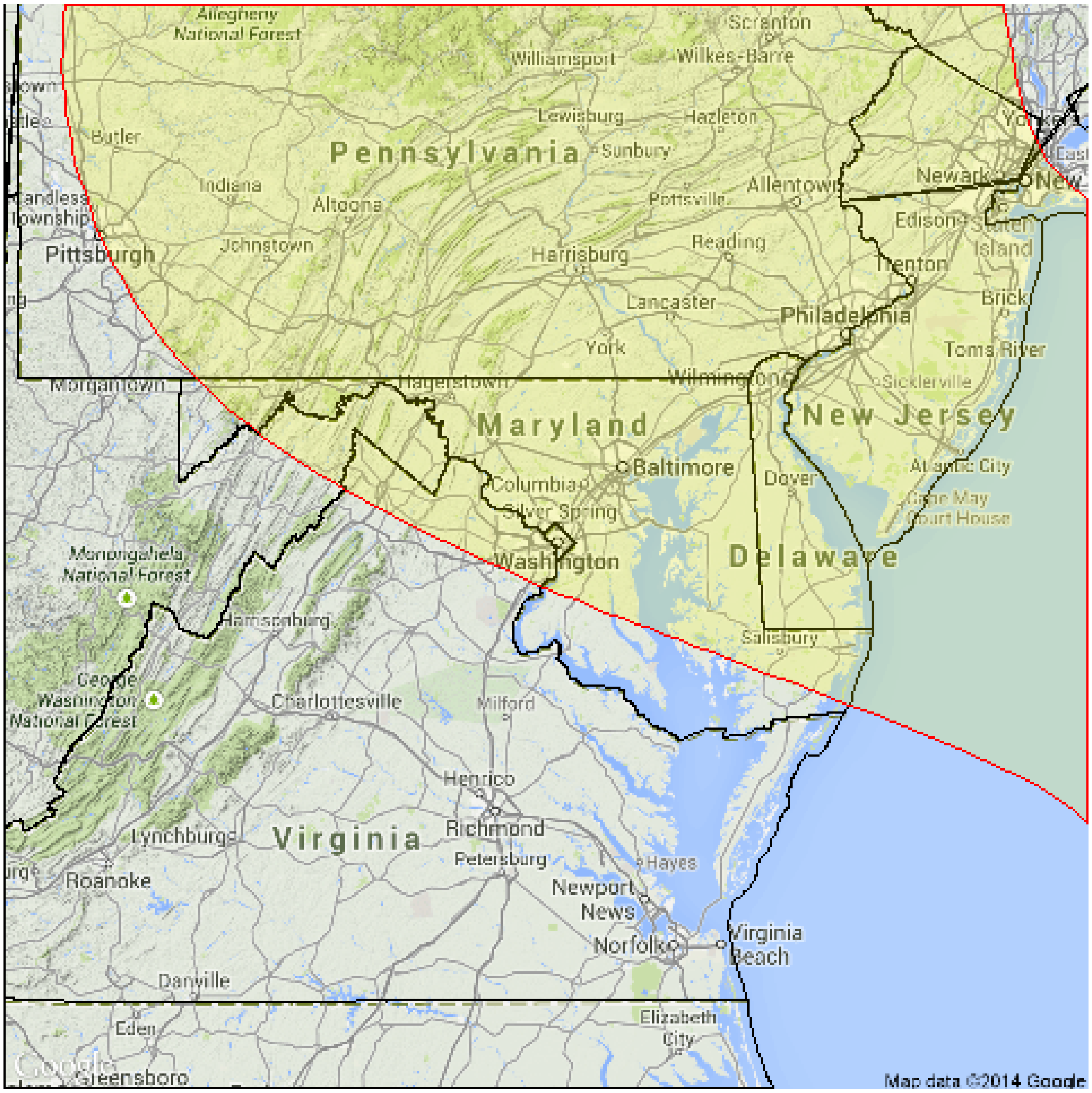

Figure 4: Hurricane Sandy Forecast Path (Mid-Atlantic), October 28, 2012

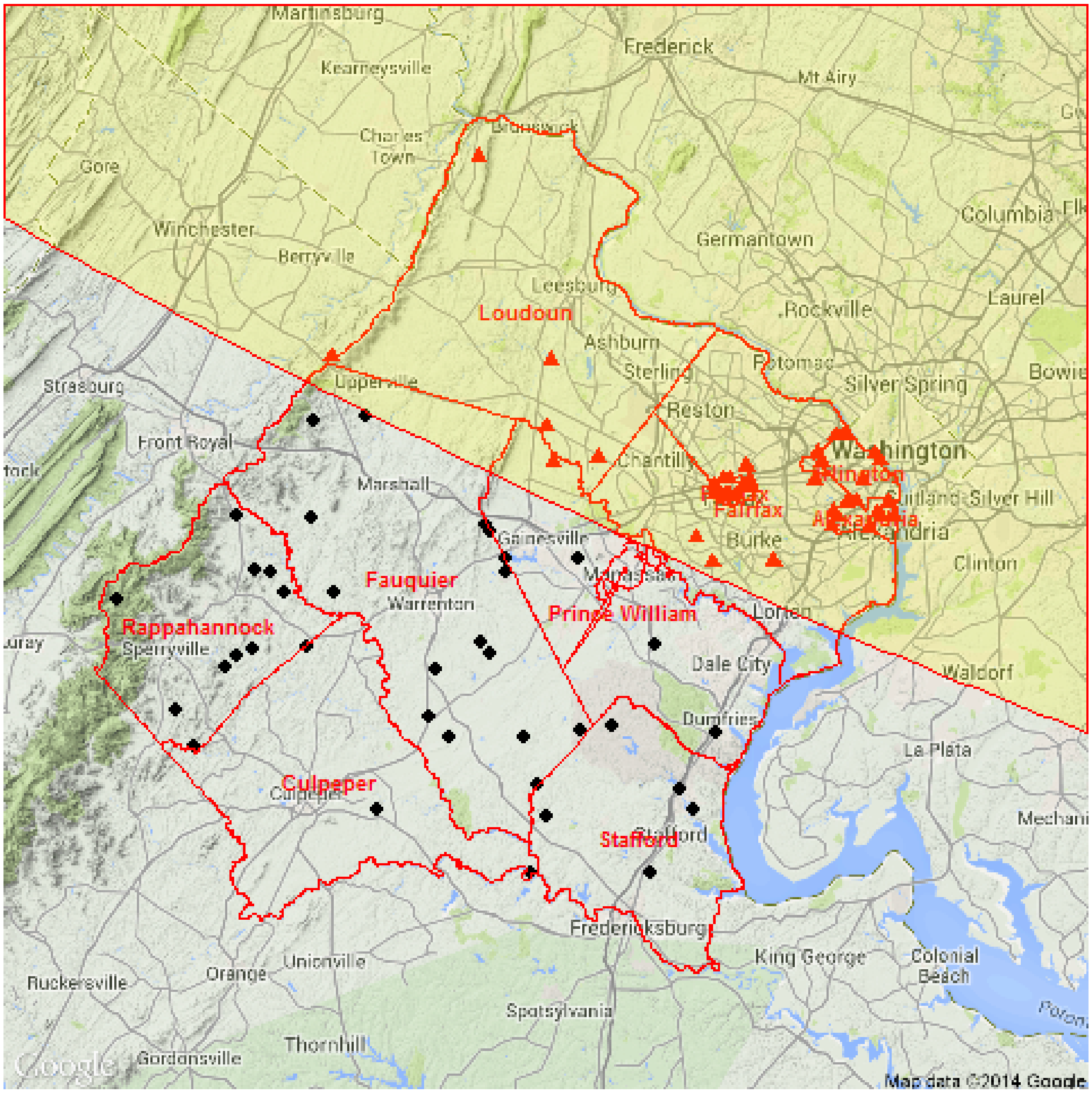

Figure 5: Properties Potentially Impacted by Hurricane Sandy

Appendix - Hurricane Path Analysis

```
###
# example code to draw Hurricane Storm Path
###

library(ggmap)
library(maptools)
library(rgeos)
library(raster)


source("CommonFunctions.R")

# retrieve simulated property location data
load("../data/property_locations.RData")

# convert property location to Spatial data for testing in or out of region
property.locations <- SpatialPoints(property.df[,1:2],
                                    proj4string=CRS("+proj=longlat +datum=WGS84"))

# read census.gov county shapefile data
us.counties <- readShapeSpatial("../data/tl_2014_us_county/tl_2014_us_county.shp",
                                proj4string = CRS("+proj=longlat +datum=WGS84"))

# read census.gov state shapefile data
us.states <-readShapeSpatial("../data/tl_2014_us_state/tl_2014_us_state.shp",
                             proj4string = CRS("+proj=longlat +datum=WGS84"))

# select only counties of interest
counties.of.interest <- subset(us.counties, STATEFP == 51 &
                                   NAME %in% c("Arlington", "Fairfax",
                                               "Alexandria",
                                               "Loudoun","Culpeper",
                                               "Rappahannock", "Fauquier",
                                               "Stafford","Prince William"))


# generate mape at requested location and zoom level
base.map <- get_map("prince william, va",9)
county.boundaries <- cropToMap(base.map,counties.of.interest)

# print map with property locations
storm.map <- ggmap(base.map) +
    geom_point(aes(x=lon, y=lat),
               data=property.df,
               shape=16, size=3) +
    geom_polygon(aes(x=long, y=lat, group=id),
                 data=county.boundaries,
                 color="red",alpha=0) +
    geom_text(aes(x=as.numeric(as.character(INTPTLON)),
                  y=as.numeric(as.character(INTPTLAT)), label=NAME),
              data=attr(counties.of.interest,"data"),
              fontface="bold", color="red", size=3) +
    theme_nothing()

png("../figures/base_property_locations.png")
print(storm.map)
dev.off()


# retieve storm path shapefile
storm.cone <- readShapeSpatial("../nhcdata/al182012_5day_025/al182012.025_5day_pgn.shp",
                               proj4string = CRS("+proj=longlat␣+datum=WGS84"))


# get only the 72-hour forecast
storm.path <- subset(storm.cone,FCSTPRD==72)

# display high-level map
ec <- get_map("arlington,␣va",6)
storm.path.to.display <- cropToMap(ec,storm.path)

labpt <- attr(attr(storm.path,"polygons")[[1]],"labpt")

storm.map <- ggmap(ec) +
    geom_polygon(aes(x=long, y=lat, group=id),
                 data=storm.path.to.display,
                 color="red",fill="yellow", alpha=0.2,size=1) +
    geom_text(aes(x=labpt[1], y=labpt[2]),
              label="Hurricane␣Sandy\n72-hour␣Forecast␣Path",
              size=10,
              color="red") +
    theme_nothing()
png("../figures/high-level_storm_path.png")
print(storm.map)
dev.off()

ec <- get_map("arlington,␣va",7)
storm.path.to.display <- cropToMap(ec,storm.path)
states.to.display <- cropToMap(ec,us.states)

png("../figures/mid-level_storm_path.png")
ggmap(ec) +
    geom_polygon(aes(x=long, y=lat, group=id),
                 data=states.to.display,
                 color="black",alpha=0,size=0.3) +
    geom_polygon(aes(x=long, y=lat, group=id),
                 data=storm.path.to.display,
                 color="red",fill="yellow", alpha=0.2,size=0.3) +
    #     ggtitle("Hurricane Sandy 3-Day Forecast Path as of 10/28/2012") +
    theme_nothing()
```

```
dev.off()

# determine the properties in the storm path region


# extract out storm path polygon data for testing
sp.storm <- SpatialPolygons(Srl=attr(storm.path,"polygons"))
proj4string(sp.storm) <- CRS(proj4string(storm.path))

# determine the properties in the storm path region
flag <- over(property.locations,sp.storm)
property.df$col <- factor(ifelse(!is.na(flag),"in","out"),levels=c("in","out"))
property.df$pch <- ifelse(!is.na(flag),"17","16")
property.count <- length(flag)
property.value<- sum(floor(property.df$value))
property.count.at.risk <- sum(!is.na(flag))
property.value.at.risk <- sum(floor(property.df$value[is.na(flag)]))


# plot property locations
# print map with property locations
storm.map <- ggmap(base.map) +
    geom_point(aes(x=lon, y=lat, color=col, shape=pch),
               data=property.df,
               size=3) +
    scale_color_manual(values=c("red","black"))+
    geom_polygon(aes(x=long, y=lat, group=id),
                 data=county.boundaries,
                 color="red",alpha=0) +
    geom_text(aes(x=as.numeric(as.character(INTPTLON)),
                  y=as.numeric(as.character(INTPTLAT)), label=NAME),
              data=attr(counties.of.interest,"data"),
              fontface="bold", color="red", size=3) +
    theme_nothing()


storm.path.to.display <- cropToMap(base.map,storm.path)
# generate map with storm path
storm.map <- storm.map +
    geom_polygon(aes(x=long, y=lat, group=id),
                 data=storm.path.to.display,
                 color="red",fill="yellow", alpha=0.2,size=0.3) +
    #     ggtitle("Hurricane Sandy Affected Areas") +
    theme_nothing()

png("../figures/affected_properties.png")
print(storm.map)
dev.off()
```

Appendix - Generate Sample Property Data

```
###
# Generate simulated property data
###

library(ggmap)
library(ggplot2)
library(maptools)
library(sp)
library(raster)
library(rgeos)

source("CommonFunctions.R")

# read census.gov county shapefile data
us.counties <- readShapeSpatial("../data/tl_2014_us_county/tl_2014_us_county.shp",
                                proj4string = CRS("+proj=longlat +datum=WGS84"))

# read census.gov state shapefile data
us.states <-readShapeSpatial("../data/tl_2014_us_state/tl_2014_us_state.shp",
                             proj4string = CRS("+proj=longlat +datum=WGS84"))

# select only states of interest
states.of.interest <- subset(us.states,STUSPS %in% c("VA"), select=GEOID)
counties.of.interest <- subset(us.counties,STATEFP == 51 &
                               NAME %in% c("Arlington", "Fairfax",
                                           "Alexandria",
                                           "Loudoun","Culpeper",
                                           "Rappahannock", "Fauquier",
                                           "Stafford","Prince William"))


###
# generate simulate property locations in the counties of interest
###

# generate long/lat coordinates and property value
generatePropertyData <- function(sp, num.pts=5) {
    # get Polygon definition for a county
    polygon <- attr(sp,"Polygons")[[1]]

    # get coordinates for the polygon defintion
    coords <- attr(polygon,"coords")
    colnames(coords) <- c("long","lat")

    # compute bounding box for the region
    bb <- c(min(coords[,"long"]),min(coords[,"lat"]),
            max(coords[,"long"]),max(coords[,"lat"]))
    names(bb) <- c("ll.lon","ll.lat","ur.lon","ur.lat")
    # randomly "place" points in the region, this is not perfect some will be out of regior
    set.seed(13)
    lon.pts <- runif(num.pts, bb["ll.lon"], bb["ur.lon"])
    lat.pts <- runif(num.pts, bb["ll.lat"], bb["ur.lat"])
    value <- runif(num.pts,50000,200000)

    invisible(cbind(lon=lon.pts,lat=lat.pts, value=value))
}

ll <- lapply(attr(counties.of.interest,"polygons"), generatePropertyData,10)

df <- data.frame(do.call(rbind,ll))

property.locations <- SpatialPointsDataFrame(df[,1:2],data=data.frame(value=df[,3]),
                                               proj4string=CRS("+proj=longlat +datum=WGS84"))
# makes points are in the counties of interest
sp.polygons <- SpatialPolygons(Srl=attr(counties.of.interest,"polygons"))
proj4string(sp.polygons) <- CRS(proj4string(counties.of.interest))
flag <- over(property.locations,sp.polygons)
df$col <- factor(ifelse(!is.na(flag),"in","out"),levels=c("in","out"))
df <- df[df$col=="in",]

# this.map <- get_map("arlington, virginia",9)
# county.boundaries <- cropToMap(this.map,counties.of.interest)
#
# ggmap(this.map) +
#     geom_point(aes(x=lon, y=lat),
#                data=df,
#                color="black", shape=16, size=3) +
#     geom_polygon(aes(x=long, y=lat, group=id),
#                  data=county.boundaries,
#                  color="red",alpha=0) +
#     geom_text(aes(x=as.numeric(as.character(INTPTLON)),
#                   y=as.numeric(as.character(INTPTLAT)), label=NAME),
#               data=attr(counties.of.interest,"data"),
#               size=3) +
#     theme_nothing()

# Save property locations for analysis
property.df <- subset(df,select=-col)
property.count <- nrow(property.df)
property.value<- sum(floor(property.df$value))
save(property.df,file="../data/property_locations.RData")
```

Appendix - CommonFunctions.R

```
# function to crop spatial data to bounding box of a Google Map
cropToMap <- function(the.map,spatial.data) {
    # the.map - Google map to crop the storm path to
    # spatial.data - Spatial data to display on map


    # calculate bounding box for displaying storm path
    bb <- attr(the.map,"bb")

    # adjust bounding box to make it slightly smaller than map
    epsilon <- 1e-6
    bb <- bb + c(epsilon, epsilon, -epsilon, -epsilon)

    # create cropping bounding box for the requested map
    CP <- as(extent(bb$ll.lon, bb$ur.lon, bb$ll.lat, bb$ur.lat), "SpatialPolygons")
    proj4string(CP) <- CRS("+proj=longlat +datum=WGS84")  # project string for Google Maps

    # apply cropping to spatial data
    crop.spatial.data <- gIntersection(spatial.data, CP, byid=TRUE)
    crop.spatial.data <- fortify(crop.spatial.data)

    # return the cropped spatial data
    invisible(crop.spatial.data)


}
```