

R and Google Maps

Jim Thompson

August 2014

This paper demonstrates use of R packages **ggmap**¹ and related spatial packages for visualizing and analyzing spatial data. **ggmap** provides functions to visualize spatial data on top of maps built using Google Maps, OpenStreetMaps, Stamen Maps, or CloudMade Maps. In addition, this paper illustrates how these R packages can be used in conjunction with Census Bureau geographic data and National Hurricane forecast data to determine properties that may be impacted by a hurricane.

Creating Google Map in R

In this section, we show how to geocode an address to determine its location, i.e., longitude and latitude. We then generate a Google map and annotate the map with the location information of the geocoded address.

Following code geocodes an address using the **ggmap**'s function **geocode()**.

```
###  
# 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 longitude=",location[1,1],", latitude=",location[1,2],"\n")  
  
## property is located at longitude= -77.23 , latitude= 38.93
```

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

```
# 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()
```

Note: For the free web API Google imposes a limit of 2,500² addresses that can be geocoded in a 24-hour period.

¹Kahle, 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>

²<https://developers.google.com/maps/documentation/geocoding/#Limits>

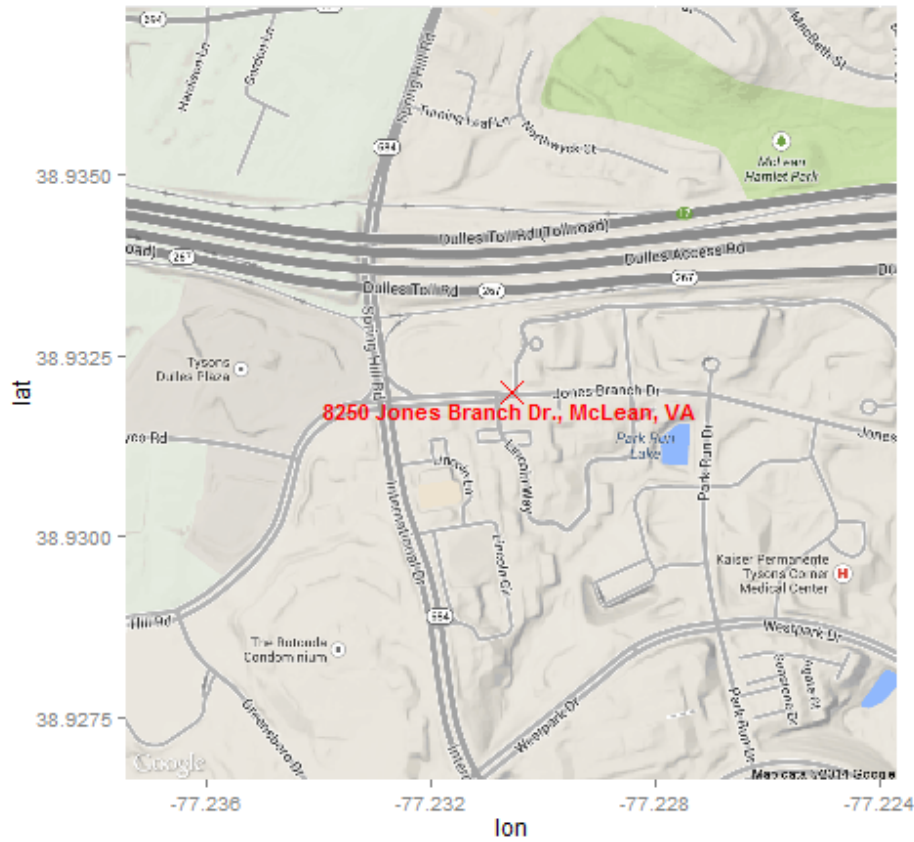


Figure 1: Sample Google Map

Hurricane Impact Analysis Use Case

This use case illustrates how features of **ggmap** and other R packages related to spatial data can be used in conjunction with hurricane forecast data from the National Hurricane Center to identify individual properties that may be affected by a hurricane. For this example, the October 28, 2012 forecast data for Hurricane Sandy is used.

All source code for this use case can be found in the appendices at the end and on GitHub at <https://github.com/jimthompson5802/GeoSpatial>.

Simulated Property Data

85 simulated properties were randomly placed in 10 Virginia administrative areas. Property values for the simulated properties are uniform pseudo-random values between \$50,000 and \$200,000. The total value of the simulated properties is \$9,381,251. Figure 2 shows locations of the simulated properties. Table 1 shows data for the first six simulated properties.

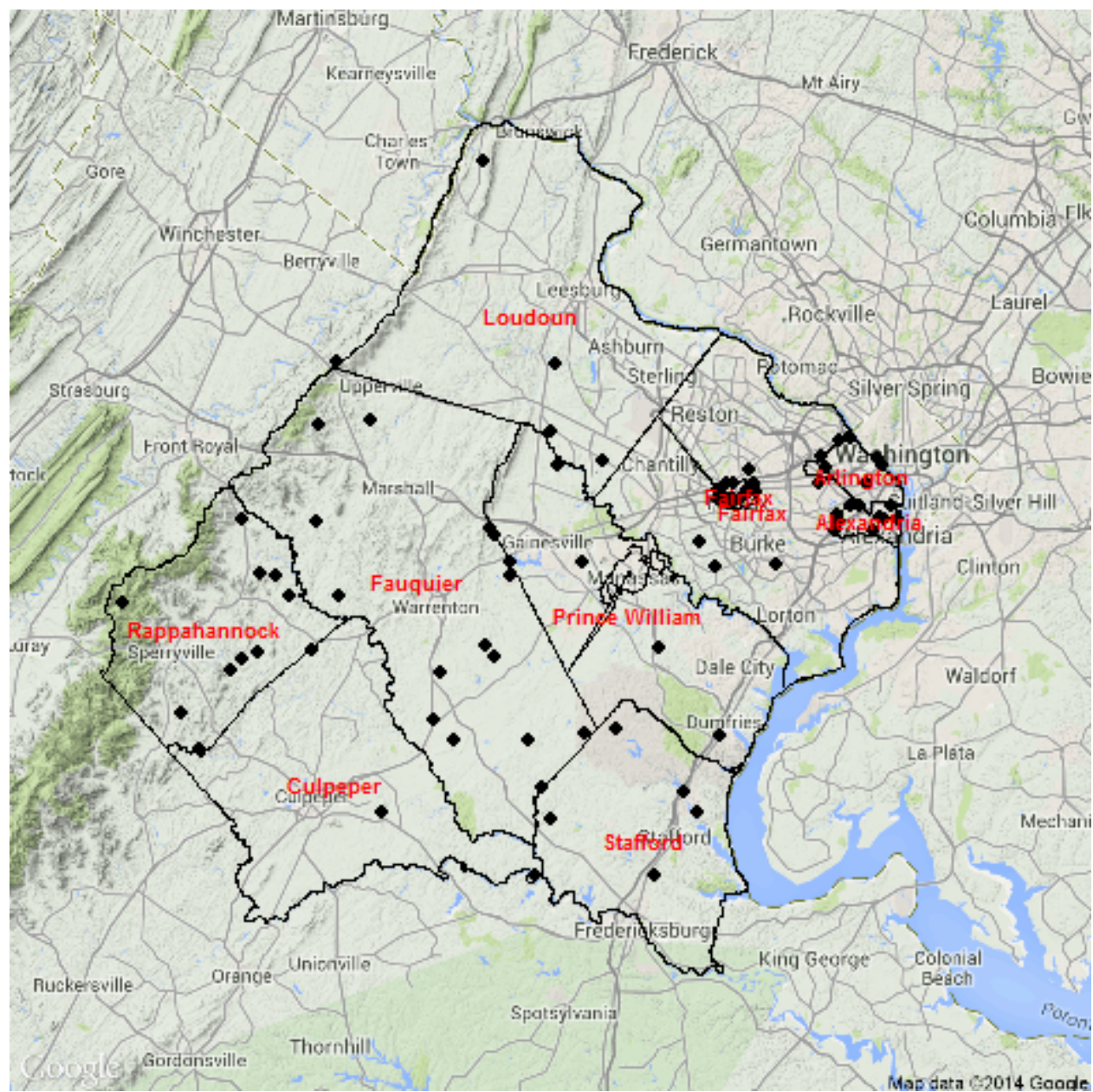


Figure 2: Simulated Property Locations

Longitude	Latitude	Property Value
-77.39	38.47	70559.98
-77.55	38.55	132042.33
-77.50	38.55	151686.96
-77.60	38.44	129212.22
-77.63	38.37	142998.58
-77.43	38.37	54758.73

Table 1: Sample Simulated Property Data

Storm Path Analysis

The National Hurricane Center provides storm forecast data in ERSI shapefile format. Shapefiles are a geospatial vector data format for geographic information systems software. Shapefiles spatially describe points, lines and polygons.

In addition to **ggmap**, these R packages are used:

- **maptools** - Set of tools for manipulating and reading geographic data, in particular ESRI shapefiles.
- **rgeos** - Interface to Geometry Engine - Open Source (GEOS) using the C API for topology operations on geometries.
- **raster** - Functions for reading, writing, manipulating, analyzing and modeling of gridded spatial data.
- **sp** - A package that provides classes and methods for spatial data.
- **ggplot2** - An implementation of the grammar of graphics in R.

Steps taken for the storm are analysis.

- Simulated property data loaded in **SpatialPointsDataFrame** structure.
- State and County boundaries from the Census Bureau loaded into **SpatialPolygonsDataFrame** structures.
- Storm path data, which are provided as shapefile, from the National Hurricane Center 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** provides a means to determine whether or not an individual property is contained within the storm path polygon structures. Figure 5 shows areas in the Washington, D.C. area potentially in the path of Hurricane 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.

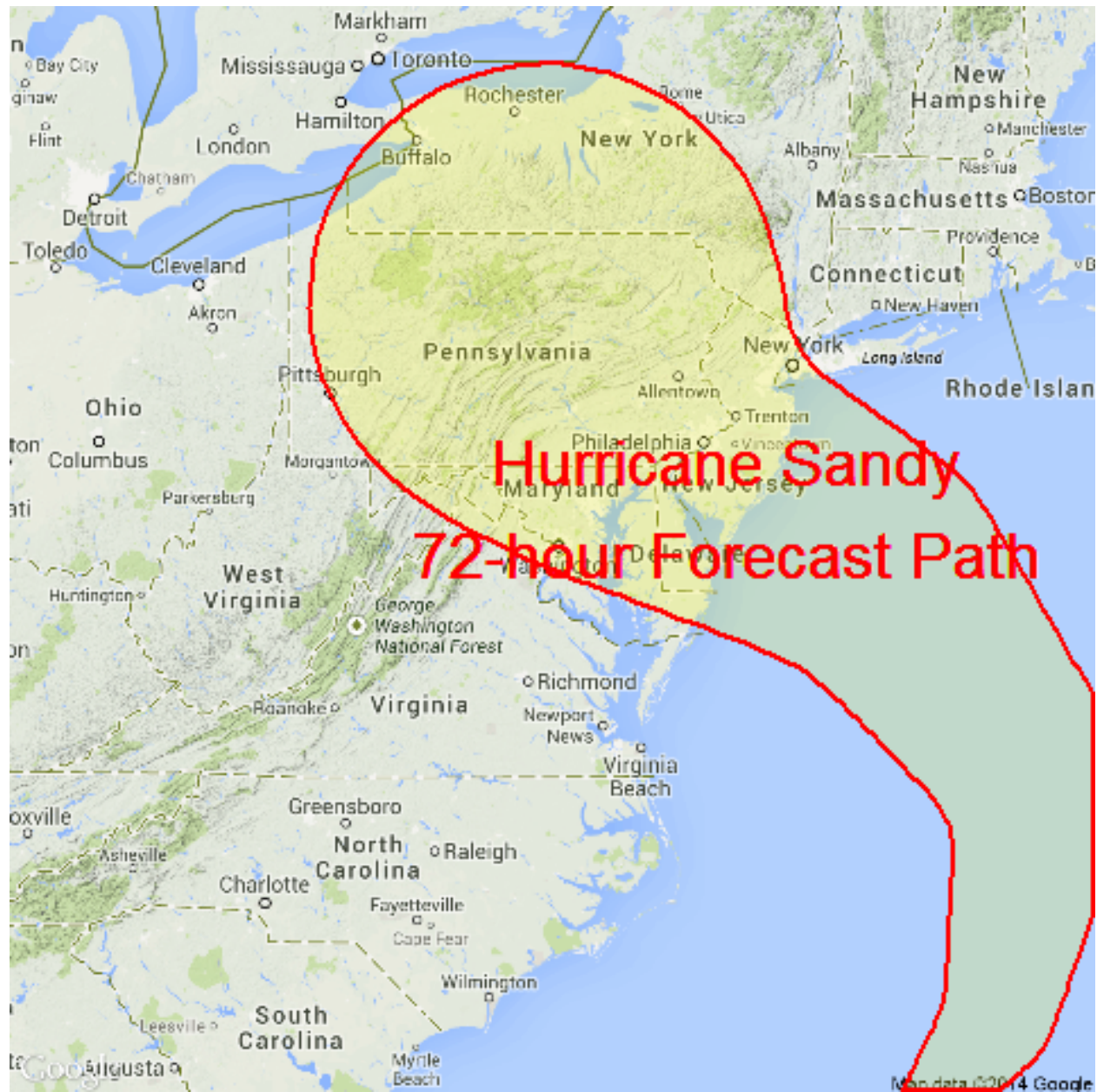


Figure 3: Hurricane Sandy Forecast Path (East Coast), October 28, 2012

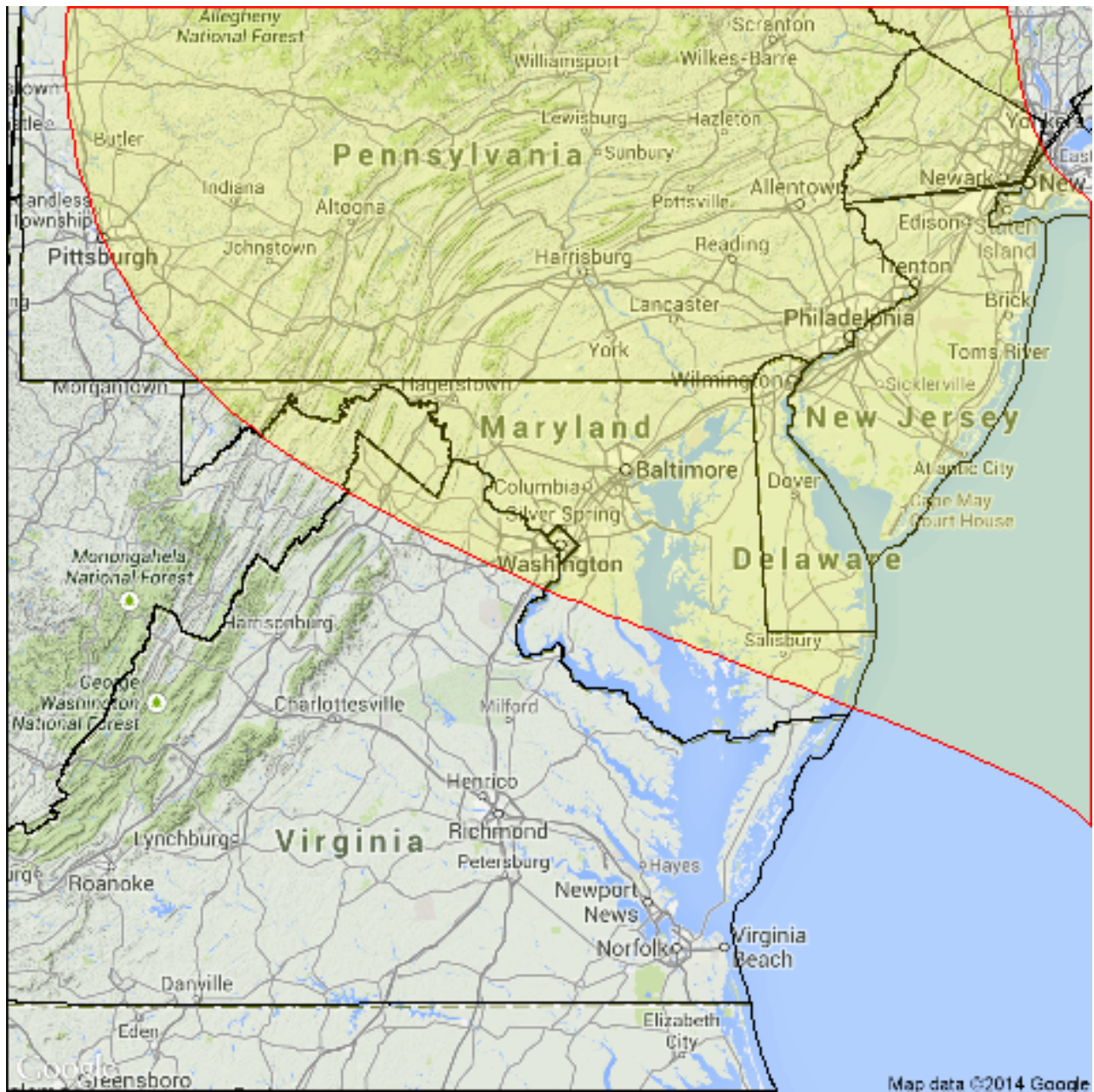


Figure 4: Hurricane Sandy Forecast Path (Mid-Atlantic), October 28, 2012

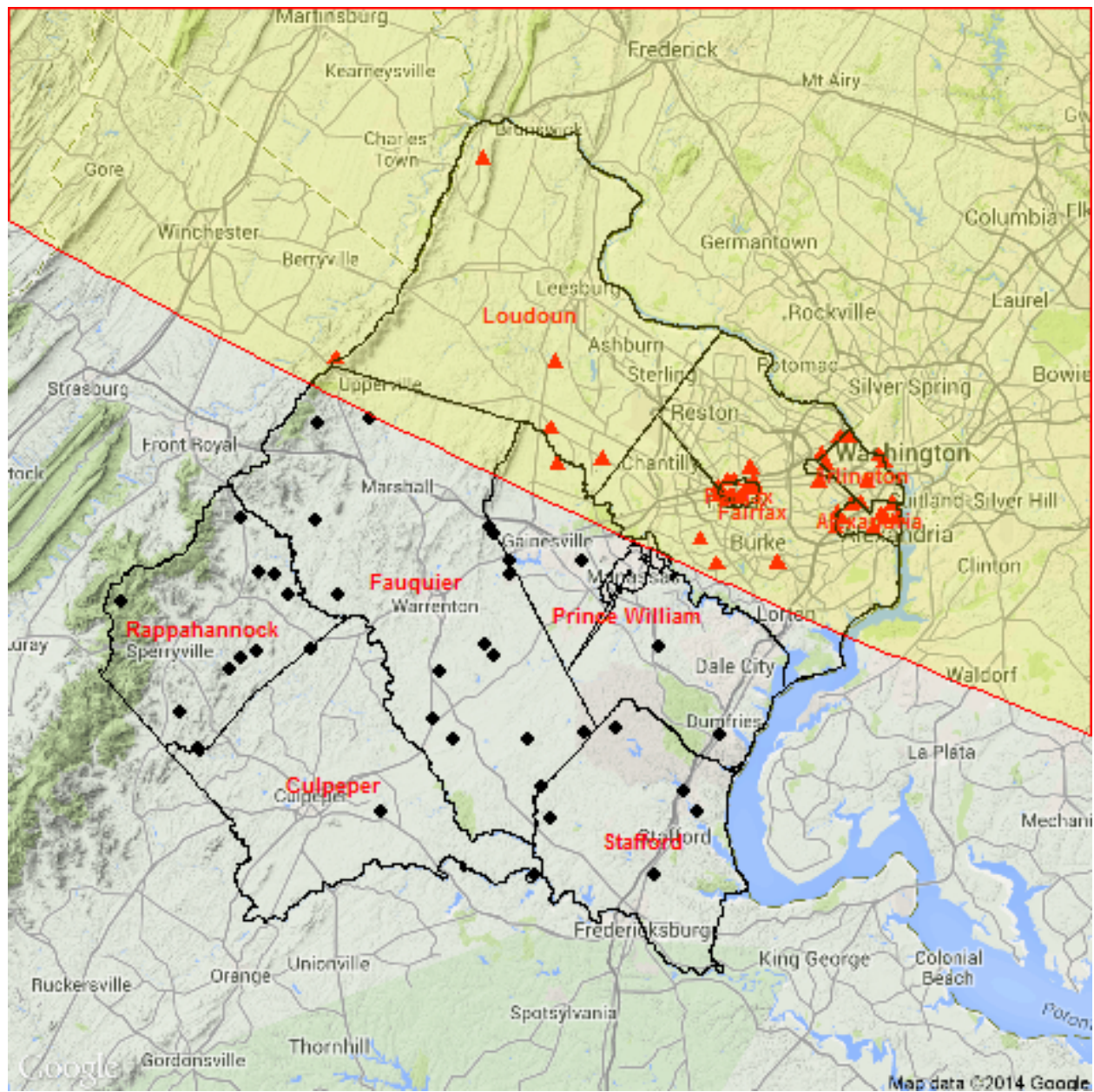


Figure 5: Properties Potentially Impacted by Hurricane Sandy

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 map 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="black", 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()

# retrieve 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="black",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 definition  
  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 region
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=c(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")

  # project string for Google Maps
  proj4string(CP) <- CRS("+proj=longlat +datum=WGS84")

  # 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)
}
```