R and Google Maps

Jim Thompson

September 21, 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 2,500[2] addresses that can be geocoded in a 24-hour period.

[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`

[2] `https://developers.google.com/maps/documentation/geocoding/#Limits`

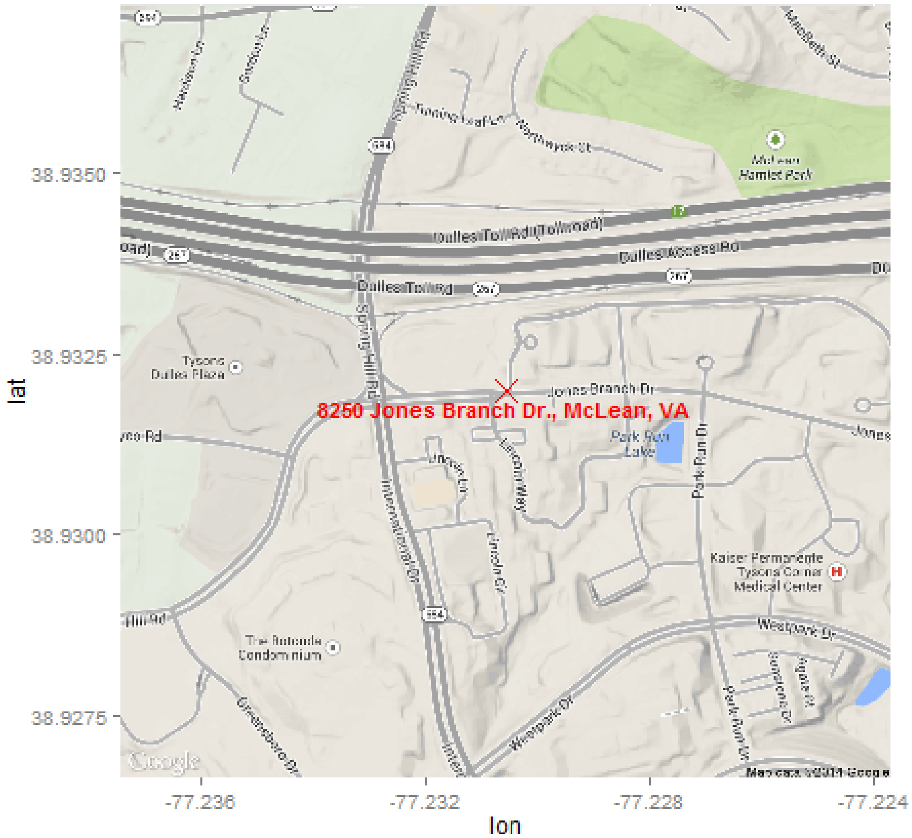

Figure 1: Sample Google Map

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

179 simulated properties were randomly placed in 10 Virginia counties. Property values for the simulated properties are uniform pseudo-random values between \$50,000 and \$200,000. Simulated loan UPB are uniform pseudo-random values between \$40,000 and \$150,000. The total value of the simulate properties is \$21,193,698 and total UPB is \$11,541,904. Figure 2 show locations of the simulated properties. Table 1 shows data for the first six simulated properties.

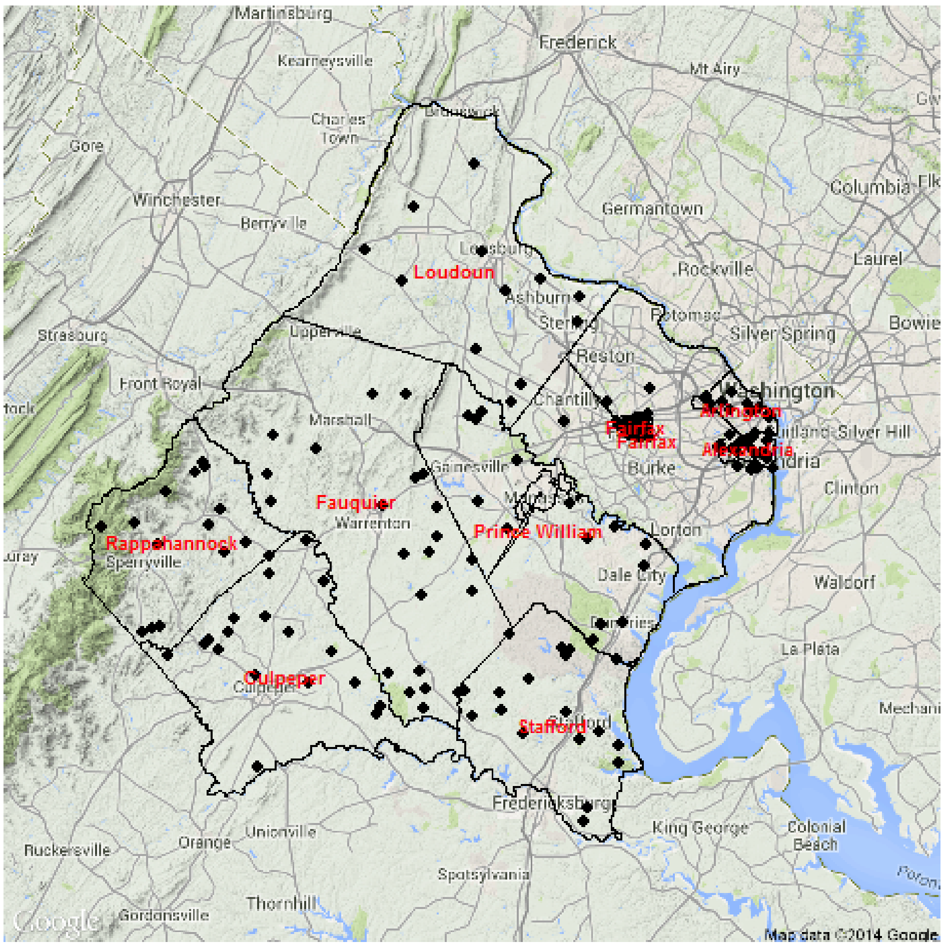

Figure 2: Simulated Property Locations

Longitude	Latitude	Property Value	UPB	Loan Identifer	County
-77.30	38.86	199676.27	36865.37	loan.8108	Fairfax city
-77.32	38.85	152093.74	65139.51	loan.9397	Fairfax County
-77.09	38.89	53199.35	62479.77	loan.511	Arlington County
-77.61	38.87	84989.42	40843.10	loan.9527	Prince William County
-77.60	39.23	199676.27	36865.37	loan.4195	Loudoun County
-77.08	38.87	133904.46	48596.03	loan.7207	Arlington County

Table 1: Sample Simulated Property Data

Storm Path Analysis

The National Hurricane Center provides storm forecast data in ERSI shapefile format. Shapefiles are a spatial 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 (NHC) loaded in **SpatialPolygonsDataFrame** structure. This code fragment reads the NHC storm path description into a R structure.

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

  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 polygon structures describing the storm path. This code fragment performs the test. For each point in the *property.locations* the value of *flag* will be the polygon identifier in *sp.storm*, which describe the storm path, containing the point or **NA** if none of the polygons contain the point.

```
# determine the properties in the storm path region
flag <- over(property.locations,sp.storm)
property.df$col <- factor(ifelse(!is.na(flag),
                                 "In Storm Path","Not In Storm Path"),
                          levels=c("In Storm Path","Not In Storm Path"))
```

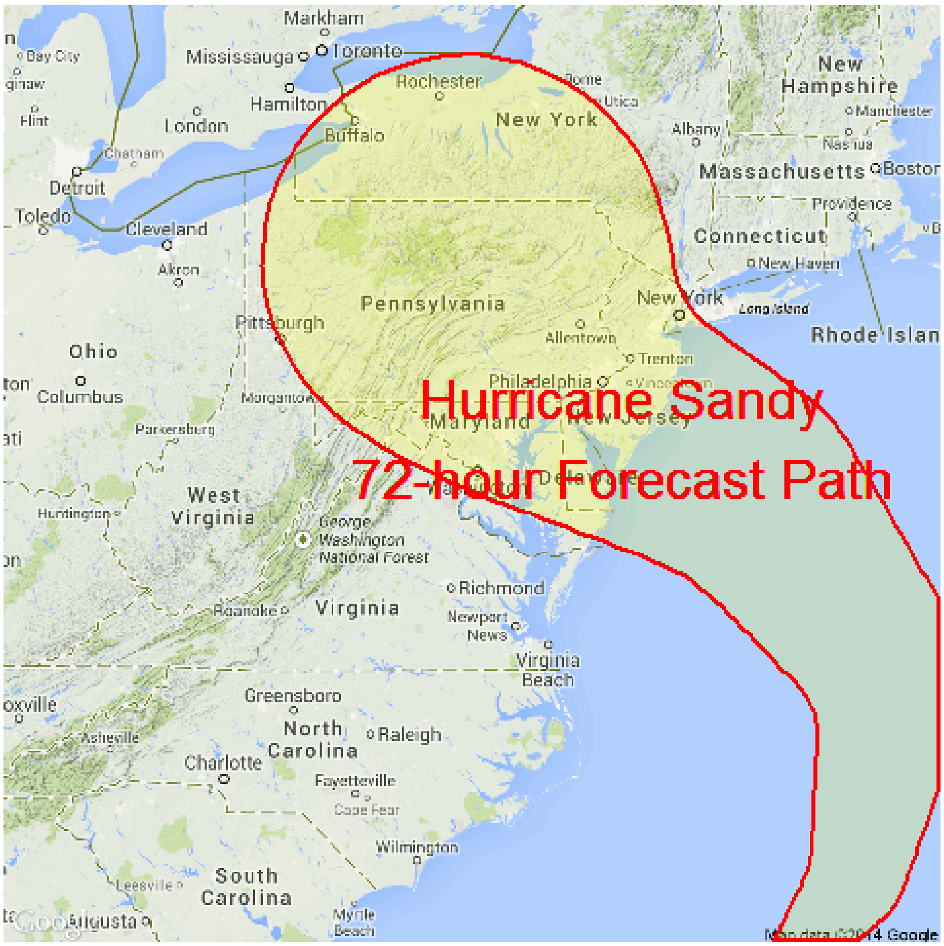

Figure 3: Hurricane Sandy Forecast Path (East Coast), October 28, 2012

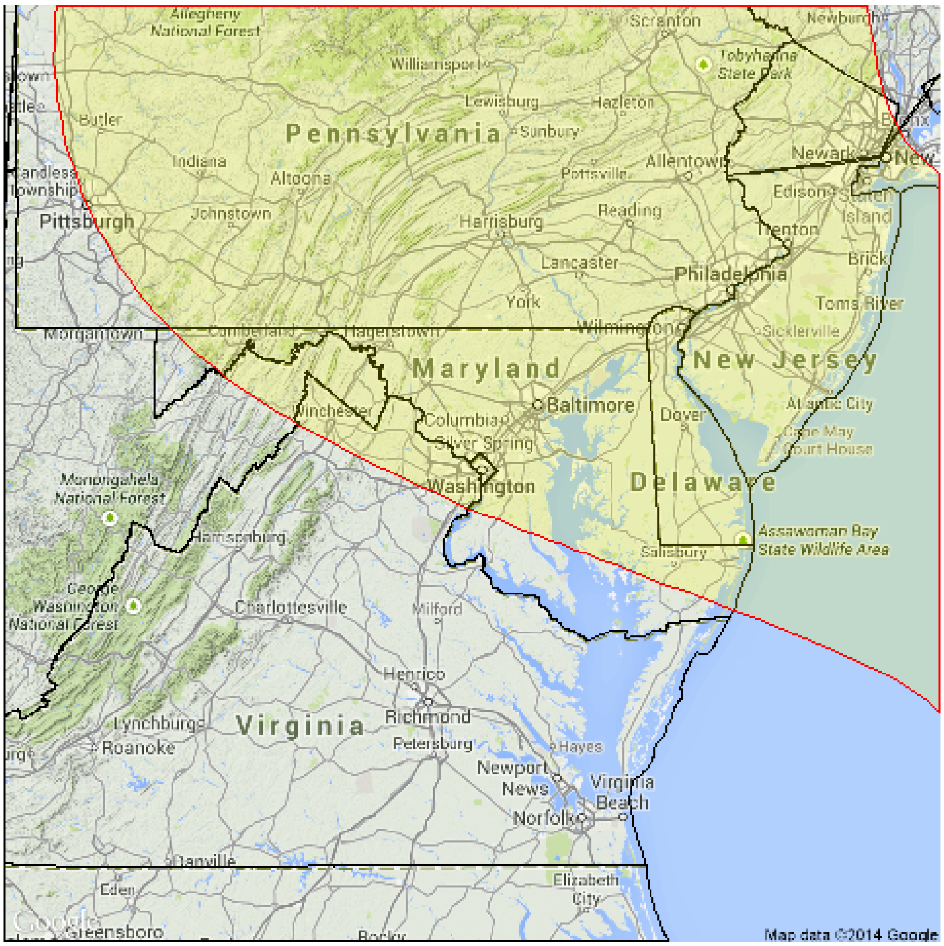

Figure 4: Hurricane Sandy Forecast Path (Mid-Atlantic), October 28, 2012

Figure 5 shows areas in the Washington, D.C. area potentially in the path of Hurricane Sandy. Properties in the hurricane path are shown in red. Out of the 179 properties in the study, we find 96 properties are in the projected storm path. These properties account for $9,940,648 out of the total $21,193,698 property value. In terms of UPB, the properties at risk account for $5,379,524 UPB out of a total of $11,541,904.

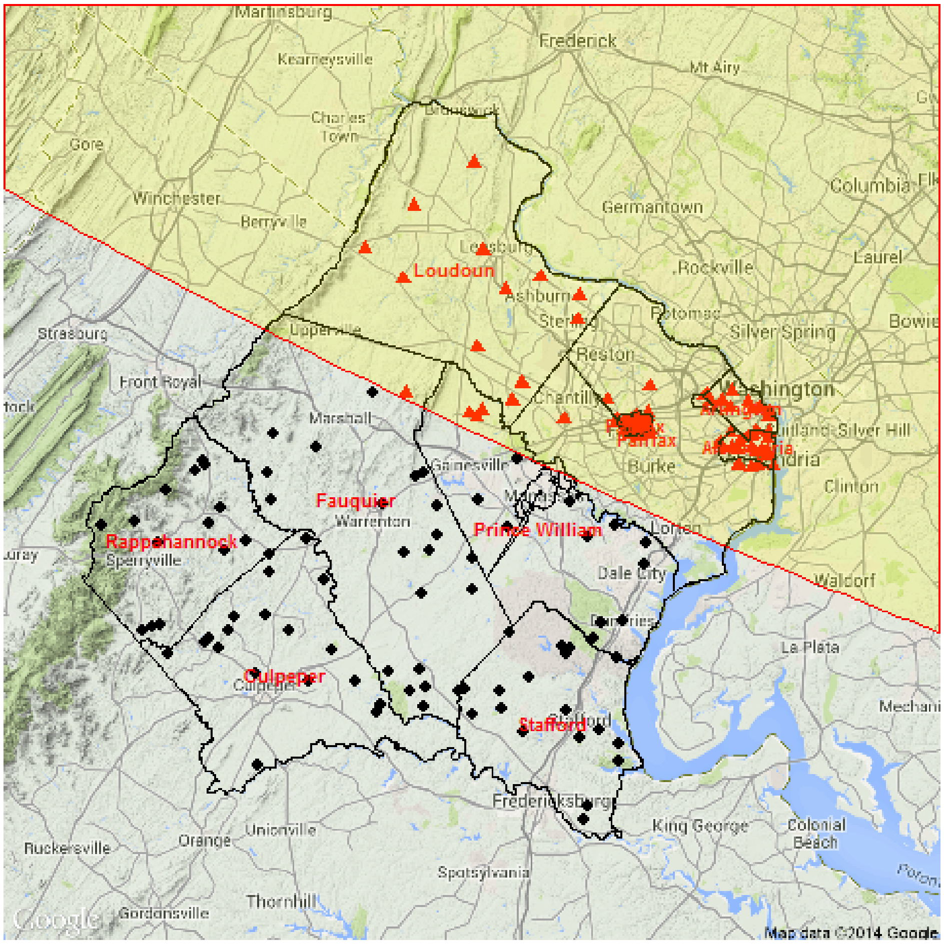

Figure 5: Properties Potentially Impacted by Hurricane Sandy

- Instead of making a county wide assumption of properties at risk, with the geocoded information of properties, we can determine the subset of properties in the storm path for given region. Fauquier and Prince William counties in Figure 6 illustrate this capability.

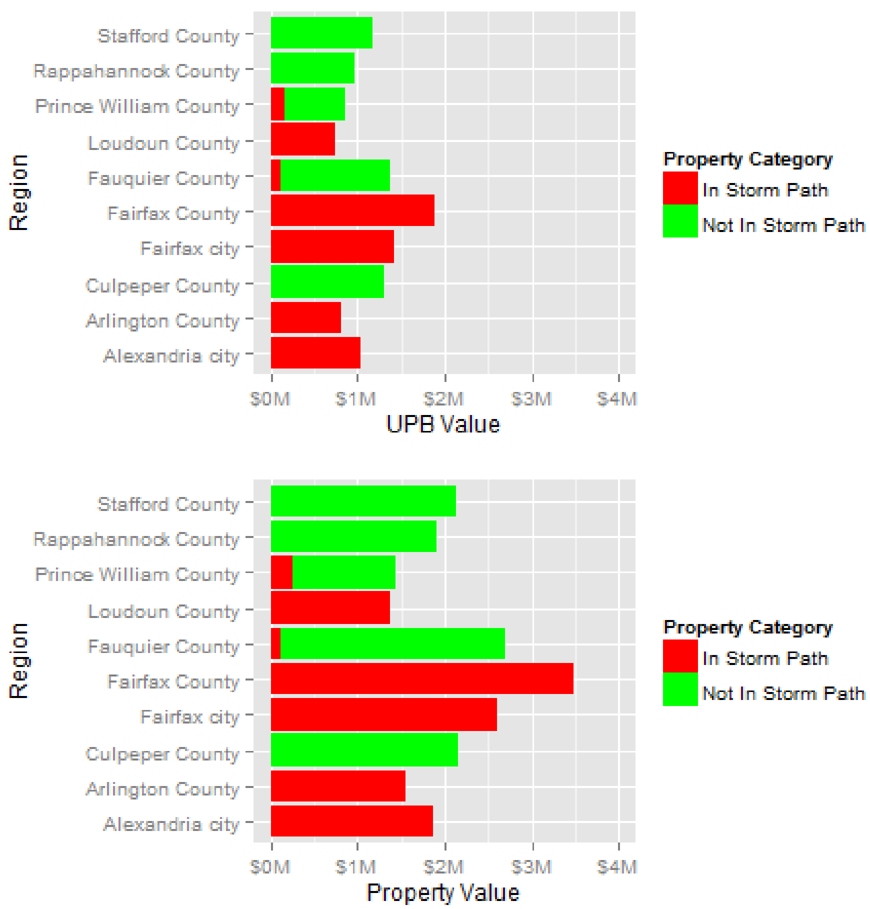

Figure 6: Property Values and Loan UPB at Risk

Wind Speed Forecast

In addition to storm path data, the NHC provides forecasts on expected wind strength. The forecasts are intended to show the expected size of the storm and the areas potentially affected by sustained winds of tropical storm force (34 Knot), 50 knot and hurricane force (64 knot) from a tropical cyclone.

Figure 7 shows the 36-hour forecast of wind speeds for Hurricane Sandy as of October 28, 2012. At this point in time, a large portion of the Northeast are expected to see tropical storm strength winds, with parts of Delaware, Maryland and New Jersey shores expected to experience hurricane strength winds. As in the storm path analysis, if there are geocode property data that contain longitude and latitude information, it is possible to determine the individual properties affected by wind strength.

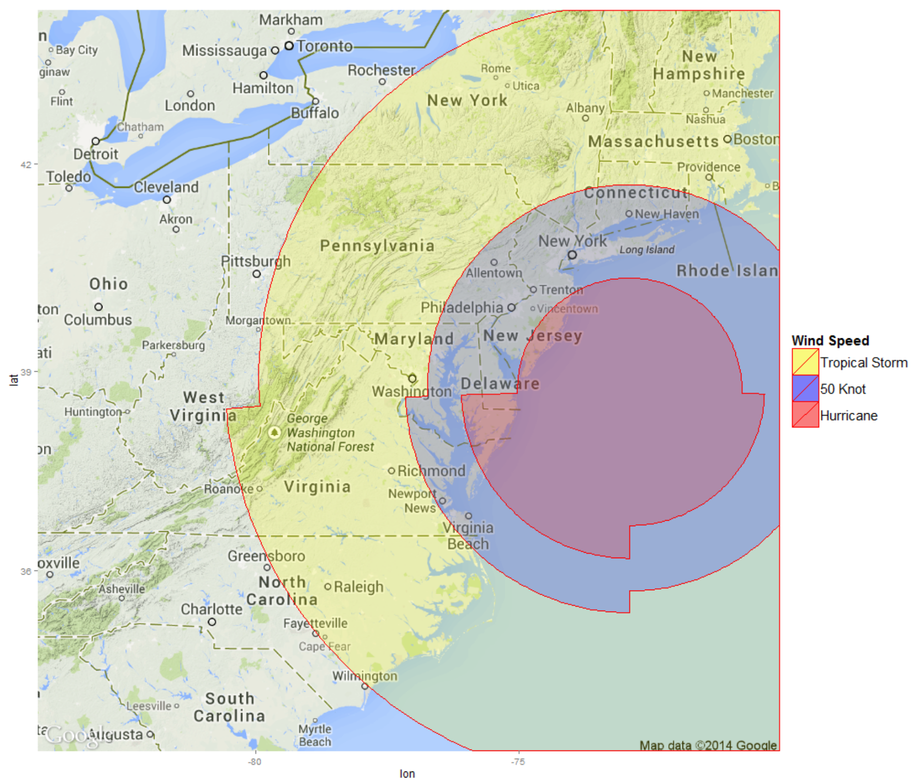

Figure 7: 36-hour Wind Forecast Regions, October 28, 2012

Appendix - Hurricane Path Analysis (drawHurricanePath.R)

```
###
# 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="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()


# 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()
```

```r
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$col <- factor(ifelse(!is.na(flag),"In Storm Path","Not In Storm Path"),
                          levels=c("In Storm Path","Not In Storm Path"))
property.df$pch <- ifelse(!is.na(flag),"17","16")
property.count <- length(flag)
property.value<- sum(floor(property.df$value))
upb.value <- sum(floor(property.df$upb))
property.count.at.risk <- sum(!is.na(flag))
property.value.at.risk <- sum(floor(property.df$value[is.na(flag)]))
upb.value.at.risk <- sum(floor(property.df$upb[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()
```

```
# do analytics on propeties
library(plyr)

df2 <- ddply(property.df,.(NAMELSAD, col),summarize, value=sum(value),
             upb=sum(upb))


p1 <- ggplot(df2, aes(x = NAMELSAD, y = upb, fill=col)) +
    scale_fill_manual(name="Property Category",values=c("red","green")) +
    scale_y_continuous(breaks=seq(0,4000000,1000000),
                       limits=c(0,4000000),
                       labels=paste0("$",seq(0,4,1),"M"))+
    geom_bar(stat='identity') +
    coord_flip()+
    xlab("Region") + ylab("UPB Value") +
    theme(axis.text.y=element_text(size=10),
          axis.text.x=element_text(size=10))

p2 <- ggplot(df2, aes(x = NAMELSAD, y = value, fill=col)) +
    scale_fill_manual(name="Property Category",values=c("red","green")) +
    scale_y_continuous(breaks=seq(0,4000000,1000000),
                       limits=c(0,4000000),
                       labels=paste0("$",seq(0,4,1),"M"))+
    geom_bar(stat='identity') +
    coord_flip()+
    xlab("Region") + ylab("Property Value") +
    theme(axis.text.y=element_text(size=10),
          axis.text.x=element_text(size=10))

png("../figures/property_analytics1.png")
multiplot(p1, p2, cols=1)
dev.off()
```

Appendix - Wind Forecast Analysis (windForecastAnalysis.R)

```
###
#  Wind Forecast Data
###
library(ggmap)
library(maptools)
library(rgeos)
library(raster)
library(grid)

source("CommonFunctions.R")

# retieve wind forecast shapefile
wind.fcst <- readShapeSpatial(paste0("../nhcdata/al182012_fcst_025",
                                      "/al182012_2012102812_forecastradii.shp"),
                               force_ring=TRUE,
                               proj4string = CRS("+proj=longlat +datum=WGS84"))
# select 72-hour forecast
wind.36 <- subset(wind.fcst,TAU==36)
wind.36@data$poly_id <- rownames(wind.36@data)


# generate map and crop wind speed polygons to map area
ec <- get_map("arlington, va",6)
wind.36.to.display <- cropToMap(ec,wind.36)
poly.ids <- data.frame(do.call(rbind,strsplit(wind.36.to.display$id," ")),
                       stringsAsFactors=FALSE)
names(poly.ids) <- c("poly_id","segment_id")
wind.36.to.display <- cbind(wind.36.to.display,poly.ids)


# combine cropped map polygon data with forecast data
wind.36.to.display <- merge(wind.36.to.display, wind.36@data)


# draw map and overlay with wind speed forecast
wind.map <- ggmap(ec) +
    geom_polygon(aes(x=long, y=lat, group=id, fill=as.character(RADII)),
                 data=subset(wind.36.to.display,RADII==34), # tropical storm winds
                 color="red", alpha=0.2, size=0.3) +
    geom_polygon(aes(x=long, y=lat, group=id, fill=as.character(RADII)),
                 data=subset(wind.36.to.display,RADII==50), # 50 knot winds
                 color="red", alpha=0.2, size=0.3) +
    geom_polygon(aes(x=long, y=lat, group=id, fill=as.character(RADII)),
                 data=subset(wind.36.to.display,RADII==64), # hurricane winds
                 color="red", alpha=0.2, size=0.3) +
    scale_fill_manual(values=c("yellow","blue","red"),
                      name="Wind Speed",
                      labels=c("Tropical Storm", "50 Knot","Hurricane")) +
    theme(legend.key.size=unit(2,"lines"),
```

```
          legend.title=element_text(size=15),
          legend.text=element_text(size=15))

png("../figures/wind_forecast.png",width=1024,height=1024)
print(wind.map)
dev.off()
```

Appendix - Generate Sample Property Data (generatePropertyTest.R)

```
###
# 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")
```

```r
    # 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)
    upb <- runif(num.pts,20000,125000)

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

set.seed(13)
ll <- lapply(attr(counties.of.interest,"polygons"), generatePropertyData,20)

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"))
# make sure 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$col <- factor(ifelse(!is.na(flag),"in","out"),levels=c("in","out"))
df$idx <- flag
df <- df[df$col=="in",]
df$loan.id <- paste0("loan.",sample(10000,nrow(df)))

# determine which counties the points are in
counties.of.interest$idx <- 1:nrow(counties.of.interest)
df<- merge(df,subset(counties.of.interest,select=c(idx,NAMELSAD)))

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

# Save property locations for analysis
property.df <- subset(df,select=-c(col,idx))
```

```
property.count <- nrow(property.df)
property.value<- sum(floor(property.df$value))
upb.value<- sum(floor(property.df$upb))
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)


}

# Multiple plot function
# based on code found at
# http://www.cookbook-r.com/Graphs/Multiple_graphs_on_one_page_%28ggplot2%29/
#
# ggplot objects can be passed in ..., or to plotlist (as a list of ggplot objects)
# - cols:   Number of columns in layout
# - layout: A matrix specifying the layout. If present, 'cols' is ignored.
#
# If the layout is something like matrix(c(1,2,3,3), nrow=2, byrow=TRUE),
# then plot 1 will go in the upper left, 2 will go in the upper right, and
# 3 will go all the way across the bottom.
#
multiplot <- function(..., plotlist=NULL, file, cols=1, layout=NULL) {
    require(grid)

    # Make a list from the ... arguments and plotlist
    plots <- c(list(...), plotlist)

    numPlots = length(plots)
```

```
    # If layout is NULL, then use 'cols' to determine layout
    if (is.null(layout)) {
        # Make the panel
        # ncol: Number of columns of plots
        # nrow: Number of rows needed, calculated from # of cols
        layout <- matrix(seq(1, cols * ceiling(numPlots/cols)),
                         ncol = cols, nrow = ceiling(numPlots/cols))
    }

    if (numPlots==1) {
        print(plots[[1]])

    } else {
        # Set up the page
        grid.newpage()
        pushViewport(viewport(layout = grid.layout(nrow(layout), ncol(layout))))

        # Make each plot, in the correct location
        for (i in 1:numPlots) {
            # Get the i,j matrix positions of the regions that contain this subplot
            matchidx <- as.data.frame(which(layout == i, arr.ind = TRUE))

            print(plots[[i]], vp = viewport(layout.pos.row = matchidx$row,
                                            layout.pos.col = matchidx$col))
        }
    }
}
```