Project 2

Isaac Horwitz (inh2102)

11/13/2020

NOTE: The code to replicate this project is here on GitHub.

Introduction

The goal of this project is to use an individual's (in this case, Wayne's) raw GPS data from a mobile phone and use it to predict location at a time in a future week–ultimately in an attempt to "assassinate" them. This report will operate on the assumption that the best time and place at which to bomb the individual is the data point we can predict most "reliably"–that is, one that our final model can predict with the highest accuracy, or the lowest deviation between our model and where Wayne will be. Since we don't yet have the data where Wayne will be in the future, we will test our model against past data in order to model our best guess, and select times and locations that perform the best.

Important points that this report will examine and attempt to address are:

- uncertainty or inaccuracies in the raw data given (the GPS giving biased estimates for longitude or latitude, giving incorrect data about times or timezones, etc.)
- the assumptions of the model this report will create, and the degree of uncertainty associated with those predictions. The final model will be highly imperfect, but represents a best guess.

Importing and Cleaning the Raw Data

Even though the most important data is longitude, latitude, and timestamp, I keep the miscellaneous variables altitude, accuracy, bearing, and speed in case they will be useful later.

```
library(jsonlite)
library(tidyverse)
library(lubridate)
library(sp)
library(dlm)
library(mgcv)

df <- data.frame()
for (i in list.files("gps")) {
  setwd("~/inh2102_project2/gps")
  dat <- read_json(i,flatten=TRUE)
  time <- c()
  time_long <- c()
  accuracy <- c()
  altitude <- c()
  bearing <- c()
```

```
  speed <- c()
  longitude <- c()
  latitude <- c()
  for (i in dat[['features']]) {
    time <- c(time,i$properties[1])
    time_long <- c(time_long,i$properties[3])
    accuracy <- c(accuracy,i$properties[4])
    altitude <- c(altitude,i$properties[5])
    bearing <- c(bearing,i$properties[6])
    speed <- c(speed,i$properties[7])
    longitude <- c(longitude,i$geometry[[2]][1])
    latitude <- c(latitude,i$geometry[[2]][2])
  }
  altitude[sapply(altitude, is.null)] <- NA
  bearing[sapply(bearing, is.null)] <- NA
  speed[sapply(speed, is.null)] <- NA

  df <- bind_rows(df,tibble(time=unlist(time),time_long=unlist(time_long),
                  accuracy=unlist(accuracy),altitude=unlist(altitude),
                  bearing=unlist(bearing),
                speed=unlist(speed),longitude=unlist(longitude),
               latitude=unlist(latitude))) %>% tibble()
}
```

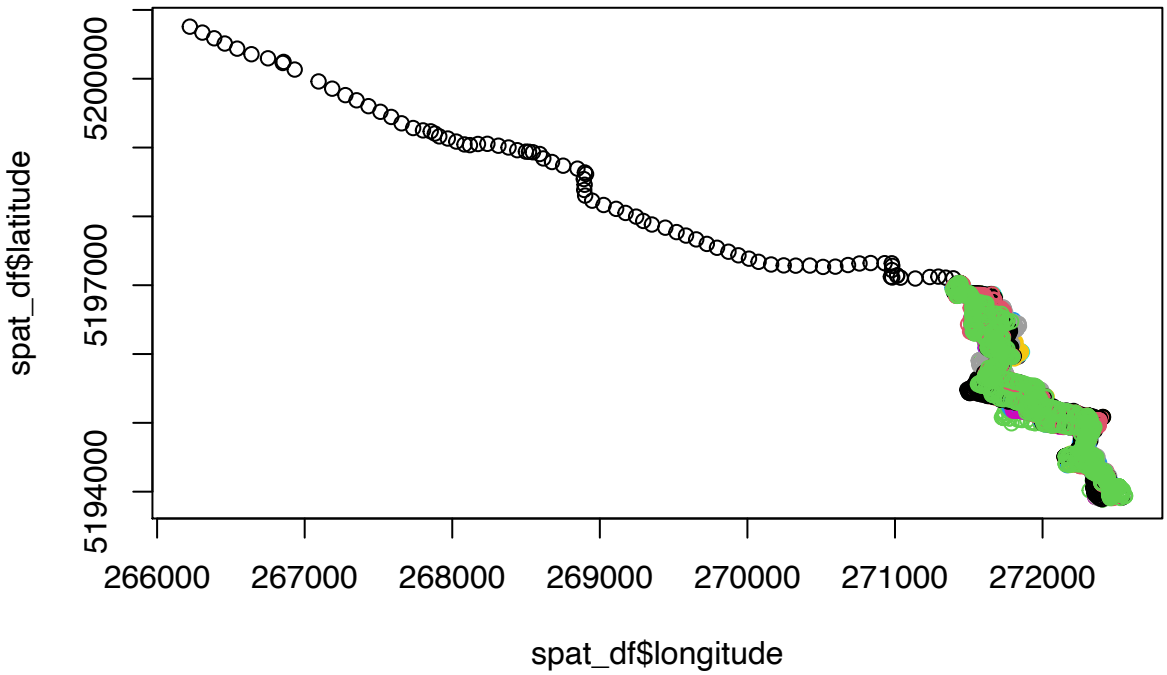
The following code chunk does two things: cleans the timestamp using the Lubridate R package to make it more easily understandable in a regression and visualization context; and creates a "SpatialPointsDataFrame" with "projected" coordinates so that our geographic data can be manipulated to calculate based on meters.

```
df$time <- paste0(substr(df$time,1,10)," ",substr(df$time,12,23))
df$time <- as.POSIXct(df$time, "%Y-%m-%d %H:%M:%S",tz="GMT")
df$day <- lubridate::day(df$time)
df$hour <- lubridate::hour(df$time)

spat_df <- SpatialPointsDataFrame(coords = df[, c("longitude", "latitude")],
                              data = df["time"],
                              proj4string=CRS("+proj=longlat +datum=WGS84"))
spat_df@data <- spat_df@data %>%
  mutate(time_long=df$time_long,accuracy=df$accuracy,altitude=df$altitude,
         bearing=df$bearing,speed=df$speed,day=df$day,hour=df$hour) %>%
  mutate(longitude=spat_df@coords[,1],latitude=spat_df@coords[,2])
spat_df <- spTransform(spat_df,CRSobj = "+proj=utm +zone=12 +datum=WGS84")
```

This is what the raw data looks like, with longitude on the X axis, latitude on the Y axis, and colors according to each day for which the raw data is labeled (assuming the default timezone GMT):

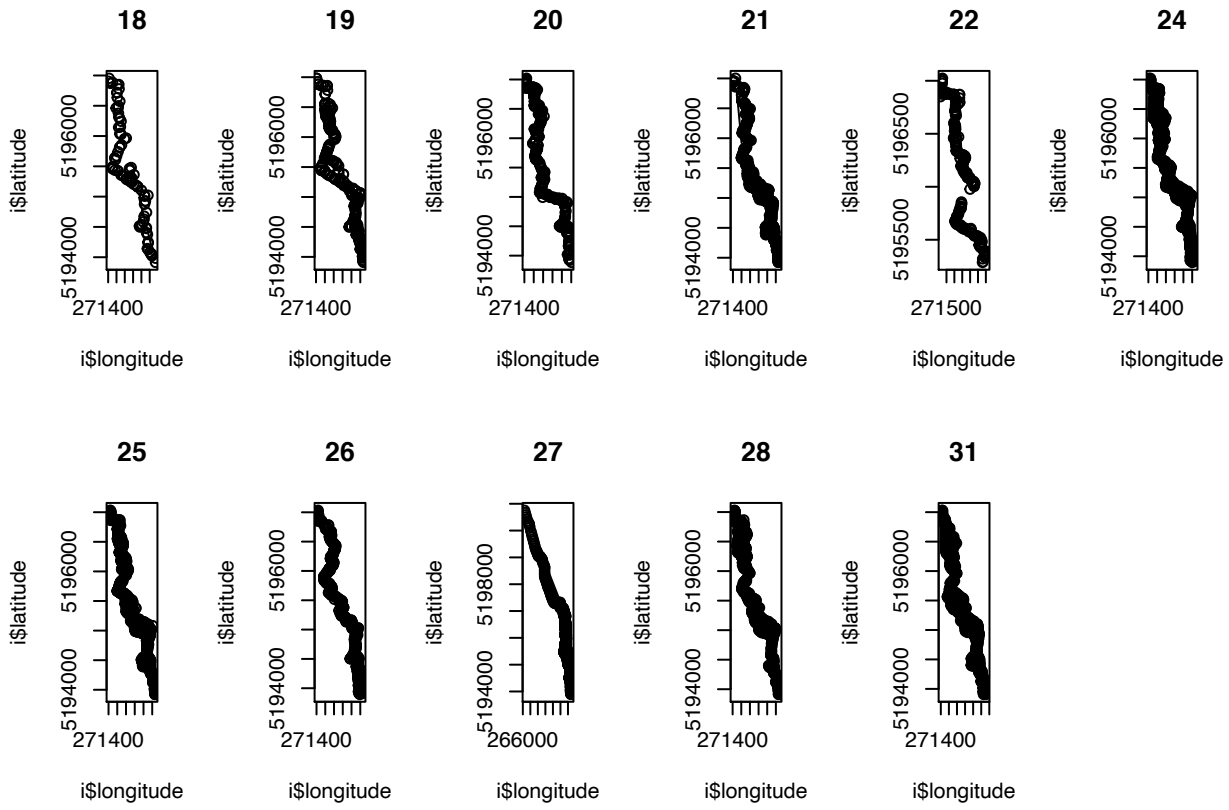
```
plot(spat_df$longitude,spat_df$latitude,col=factor(spat_df$day))
```



Without even viewing each day individually, we can see that one day's projected coordinates are much different from each other day's – the one indicated in black-outlined points.

In order to visualize the data day-by-day and note any preliminary observations, I split the data by day and visualize.

```
par(mfrow=c(2,6))
for (i in split(spat_df,spat_df$day)) {
  plot(i$longitude,i$latitude,main=i$day[!duplicated(i$day)],type='b')
}
```



Here are some preliminary thoughts:

- Some of the days have more data points than others. Aug. 18 only has 90 data points, and thus appears much more sparse than the other days – for example, the maximum is 712 data points on day Aug 27.
- The path for each day is roughly the same – meaning we don't have to deal with expressing probabilities for where Wayne will go or where he will leave from. The origin and destination are constant.
- We will certainly have to deal with the problem of data inaccuracies, including suspected missing data (for example, that is evident on Aug. 22), as well as extra junk data we'll look at more closely on Aug. 31.

Exploring Trends

Before building a predictive model, one interesting thing to explore is whether trends that are not present in the raw data influence the patterns in the data itself. First, we import pre-cleaned data from Weather Underground and extract Missoula, MT hourly Farenheit temperature values for the timestamps given.

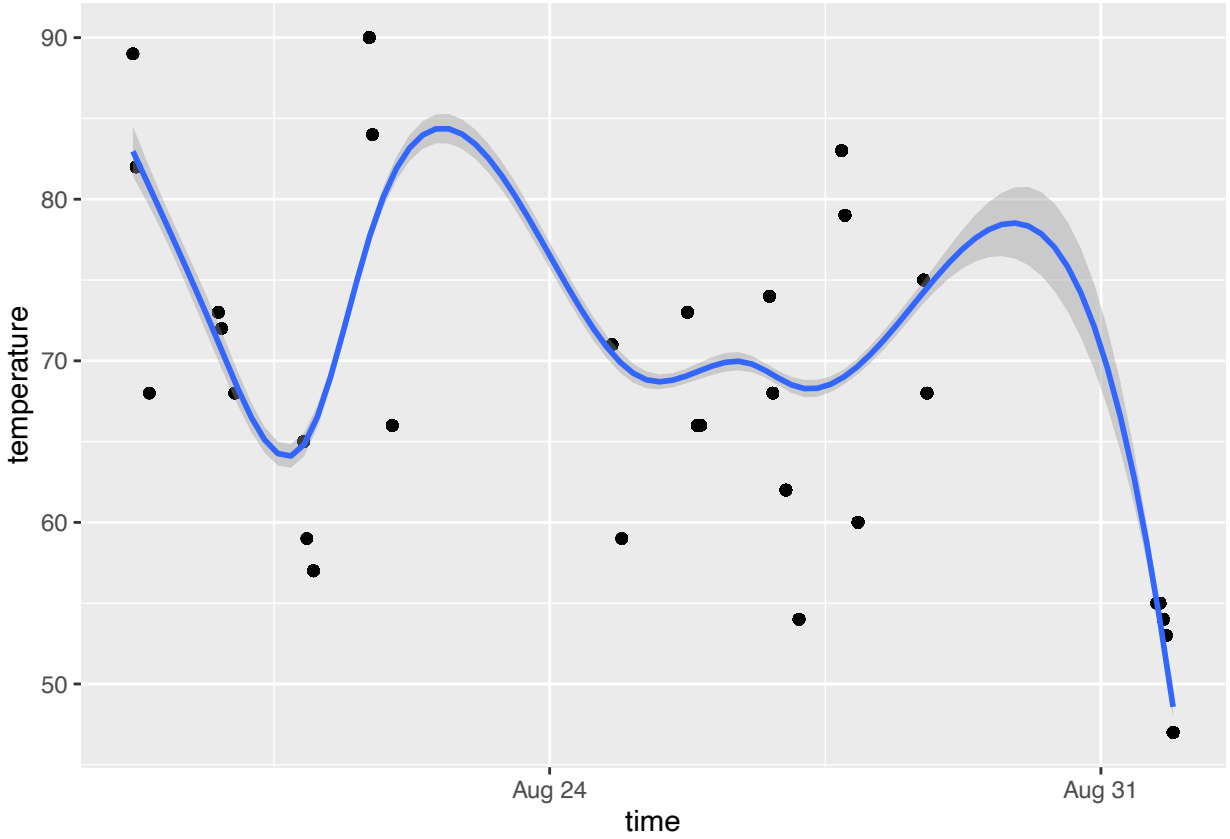
```
mt_weather <- readxl::read_excel("mt_weather.xlsx")
mt_weather <- mt_weather %>% mutate(temperature = as.numeric(substr(temperature,1,3)))
```

Since the temperature data is only hourly (while the GPS data is much more granular), we standardize both datasets so that they each contain a column with the day-hour indication, and join the two dataframes in a new one called "time" so that we can visualize temperature over the time period for the data:

```
mt_weather$time <- format(mt_weather$time,"%D-%H")
df_time <- df %>% mutate(time = format(time,"%D-%H"))
time <- left_join(df_time,mt_weather,by='time') %>%
  mutate(time = lubridate::parse_date_time(time,orders='mdyh'))
ggplot(time,aes(x=time,y=temperature)) +
```

```
  geom_point() +
  geom_smooth()
```

```
## `geom_smooth()` using method = 'gam' and formula 'y ~ s(x, bs = "cs")'
## Warning: Removed 1033 rows containing non-finite values (stat_smooth).
## Warning: Removed 1033 rows containing missing values (geom_point).
```

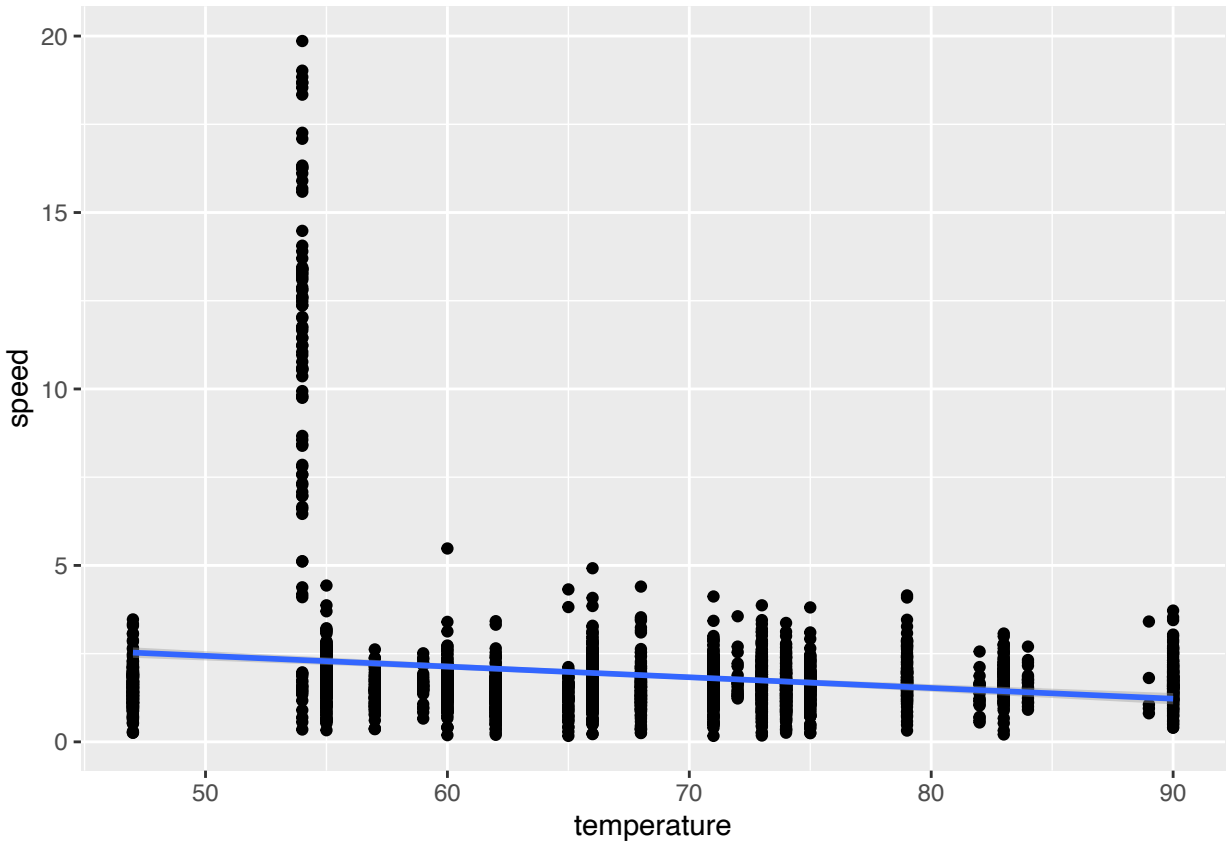


Using a locally fitted smoother (with grey bands representing SE, or uncertainty), we can see that there are fluctuations in temperature over the time period – it rises (during the morning) and drops (at night), but over the entire time period, it generally decreases (as Montana transitions out of the summer).

We can also quickly visualize whether there is an association between the given "speed" data (much of which is NA) and the temperature data:

```
ggplot(time,aes(x=temperature,y=speed)) +
  geom_point() +
  geom_smooth(method='lm')
```

```
## `geom_smooth()` using formula 'y ~ x'
```




```
speed_lm <- lm(data=time,speed~temperature)
summary(speed_lm)
```

```
##
## Call:
## lm(formula = speed ~ temperature, data = time)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -2.2697 -0.7152 -0.3024  0.1728 17.5430
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)  3.957618   0.221579  17.861   <2e-16 ***
## temperature -0.030382   0.003244  -9.366   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.9 on 2802 degrees of freedom
##   (3053 observations deleted due to missingness)
## Multiple R-squared:  0.03036,    Adjusted R-squared:  0.03001
## F-statistic: 87.73 on 1 and 2802 DF,  p-value: < 2.2e-16
```

Note: as the speed data given is likely flawed, we will not here attempt to create a perfect regression model and control for the model performing better at certain values. Regardless, we can see that there is on average a modest but likely significant relationship between speed and temperature – and on average, speed decreases by 0.03 units (possibly m/s, but not given in the data) for each increase in Farenheit temperature.

It is worthwhile to calculate our own speed given the coordinates and timestamps, but first we need to understand some properties of the data, including how frequent each data point is, and how the data can be divided into periods of collection and non-collection (stationary, or lack of movement by Wayne).

The following code chunk examines the amount of time between each (chronologically sorted) data point to get a sense of collection frequency:

```
diff_df <- df %>% arrange(-desc(time))
diff <- c()
for (i in 1:length(df$time)) {
  diff[i] = difftime(df$time[i], df$time[i-1], units="secs")
}
diff <- diff[!is.na(diff)]
summary(diff)
```

```
##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
##    -21.0     -1.0      8.0    194.9     16.0 240381.0
```

```
session_indices <- which(diff>=120)
spat_df$diff[2:nrow(spat_df)] <- diff
```

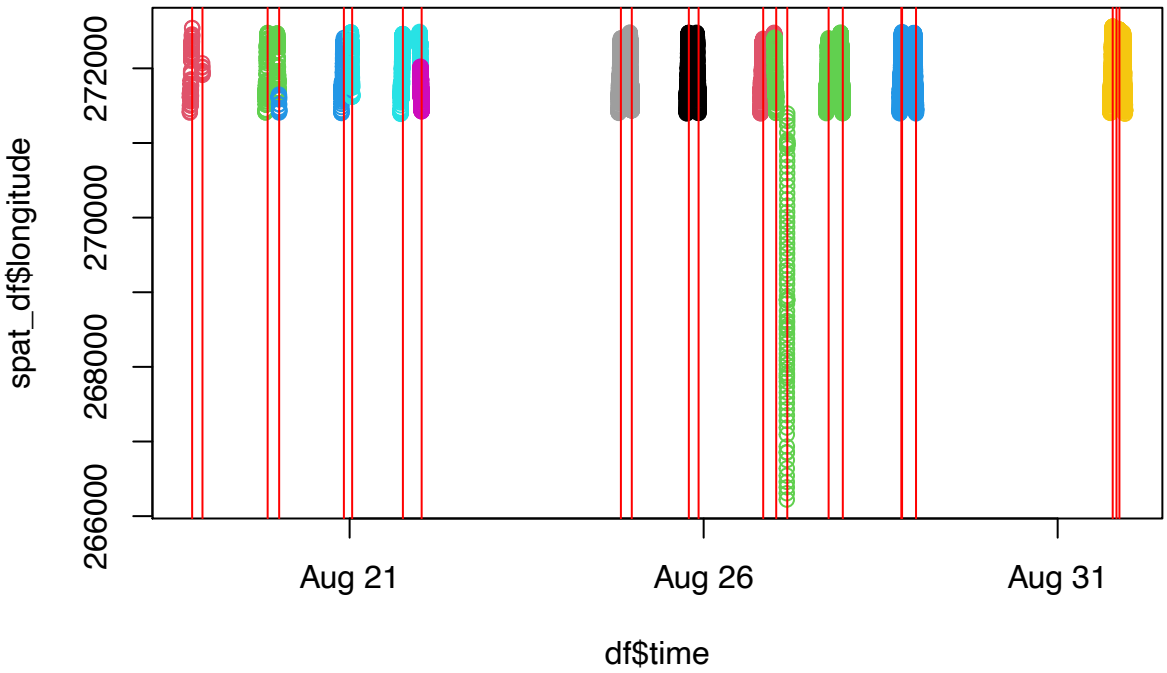
The median number of seconds between each observation is 8, although there are some extreme values – for example, a maximum of 240,381 seconds. We can define each "session" of activity somewhat arbitrarily as points for which there are fewer than 120 seconds between each data point, meaning a session ends when data is not collected for 2 minutes or more.

```
table(diff) %>% sort(decreasing=T) %>% head() %>% prop.table()
```

```
## diff
##         8        -2        16        17        -3         9
## 0.2708393 0.2341394 0.1465149 0.1209104 0.1189189 0.1086771
```

Again, this tells us that the data is very noisy and imperfect – and thus we can't take timestamps for granted whatsoever – but that it seems the intended interval of data collection is every 8 seconds.

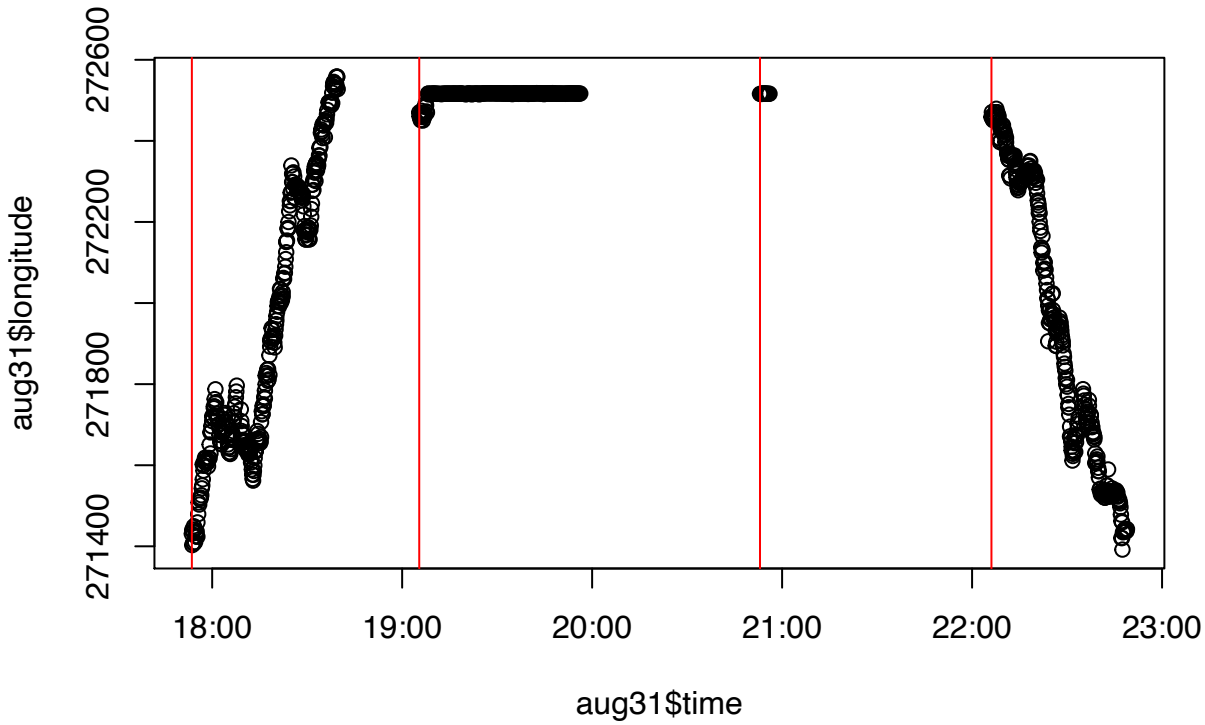
```
plot(df$time,spat_df$longitude,col=df$day)
abline(v=df$time[session_indices],col="red")
```



The plot above presents each day in a different color, and each red line indicates the beginning of a "session." We can begin to notice that a session begins when Wayne leaves his home and ends when he is at work, and then another session begins when he leaves work for home. This is true for all data except two days: **Aug. 26** and **Aug. 31**, both of which have multiple sessions due to extraneous or missing data.

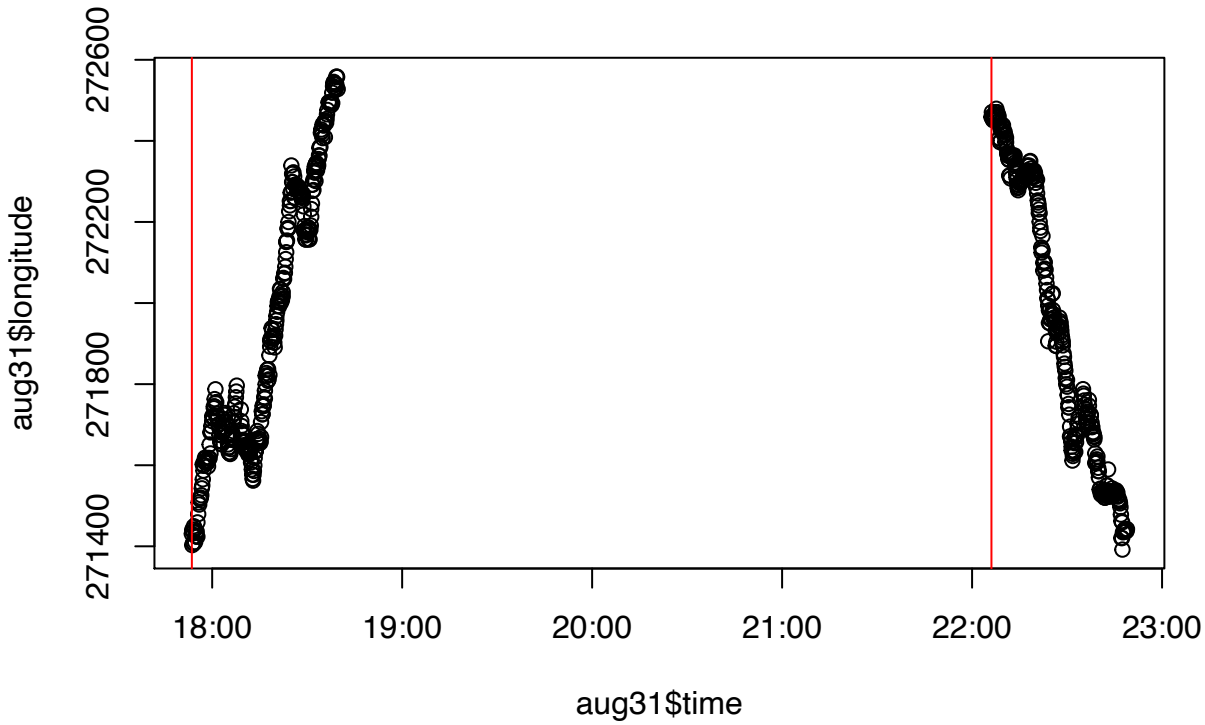
Data Cleaning: Aug. 31

```
aug31 <- spat_df[df$time>"2020-08-31 00:00:00 GMT",]
plot(aug31$time,aug31$longitude)
abline(v=aug31$time[aug31$diff>120],col="red")
```



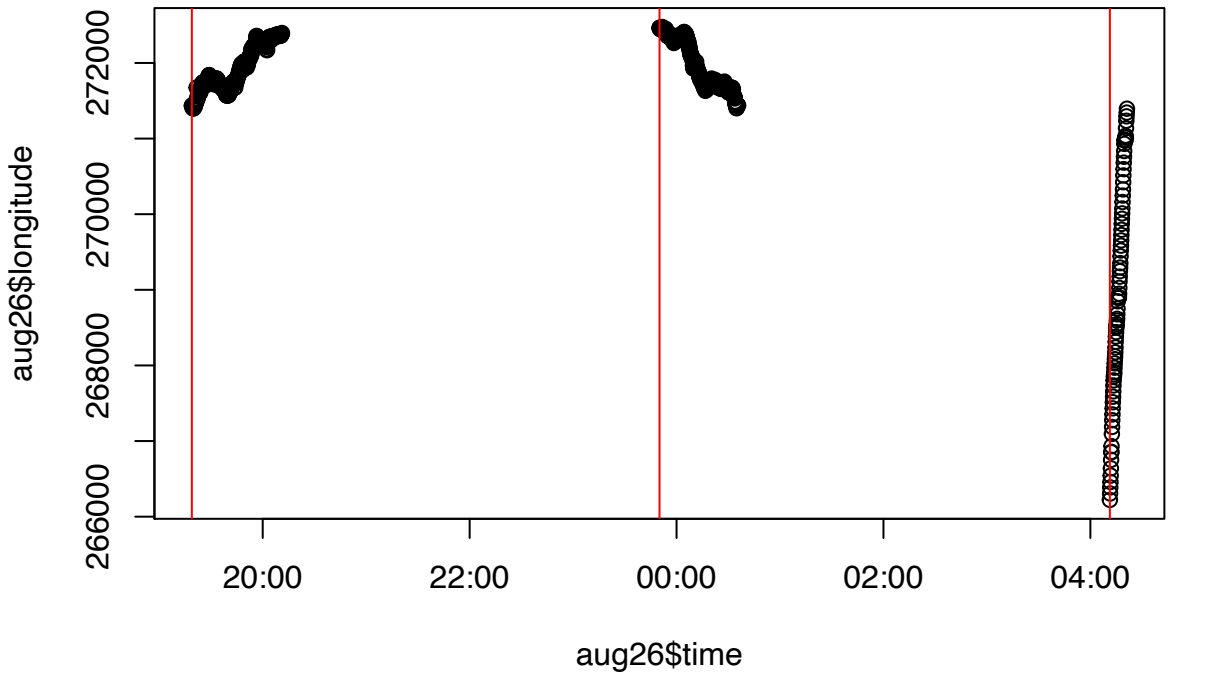
Aug. 31 has extra data when Wayne is stationary (at work), so we need to move this since we're prohibited from bombing him when he's sheltered.

```
df <- df[-c(5352:5528),]
spat_df <- spat_df[-c(5352:5528),]
time <- time[-c(5352:5528),]
aug31 <- spat_df[spat_df$time>"2020-08-31 00:00:00 GMT",]
plot(aug31$time,aug31$longitude)
abline(v=aug31$time[aug31$diff>120],col="red")
```



Data Cleaning: Aug. 26

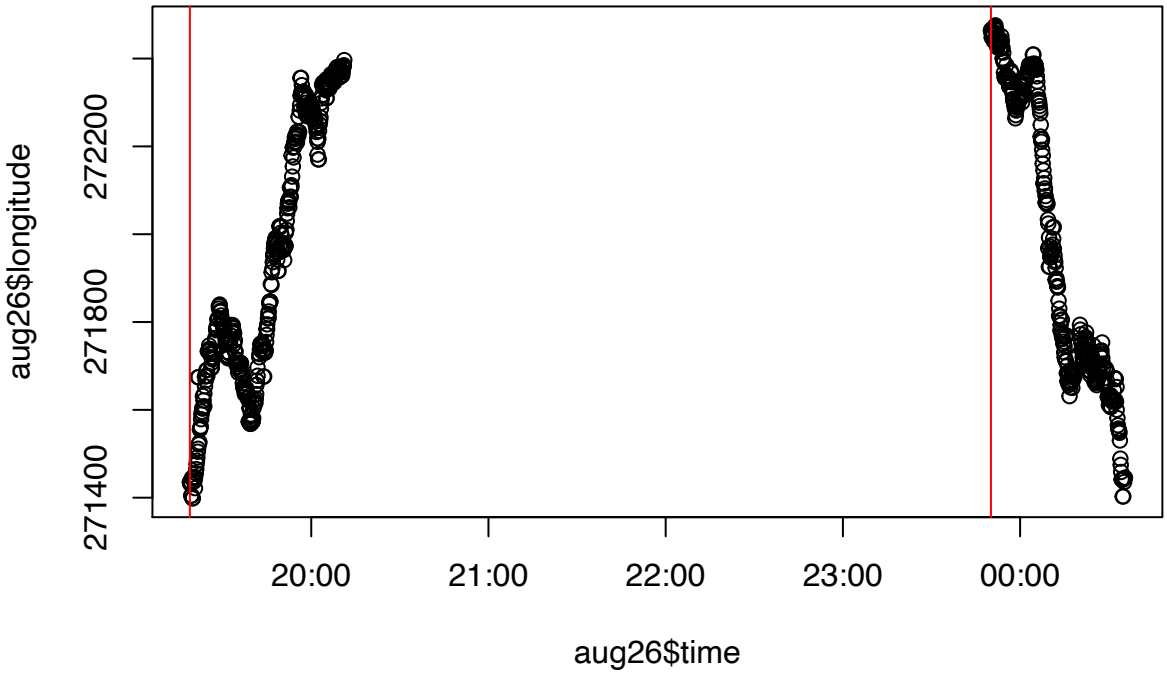
```
aug26 <- spat_df[spat_df$time>"2020-08-26 00:00:00 GMT"&spat_df$time<"2020-08-27 00:00:00 GMT",]
plot(aug26$time,aug26$longitude)
abline(v=aug26$time[aug26$diff>120],col="red")
```



Aug. 26 contains data where Wayne goes somewhere in the complete opposition direction of his work or home

– and thus including this data will bias all of our estimates when we have no reason to suspect he will re-visit this location. We remove these data points/this session altogether:

```
df <- df[-c(3534:3620),]
spat_df <- spat_df[-c(3534:3620),]
time <- time[-c(3534:3620),]
aug26 <- spat_df[spat_df$time>"2020-08-26 00:00:00 GMT"&spat_df$time<"2020-08-27 00:00:00 GMT",]
plot(aug26$time,aug26$longitude)
abline(v=aug26$time[aug26$diff>120],col="red")
```

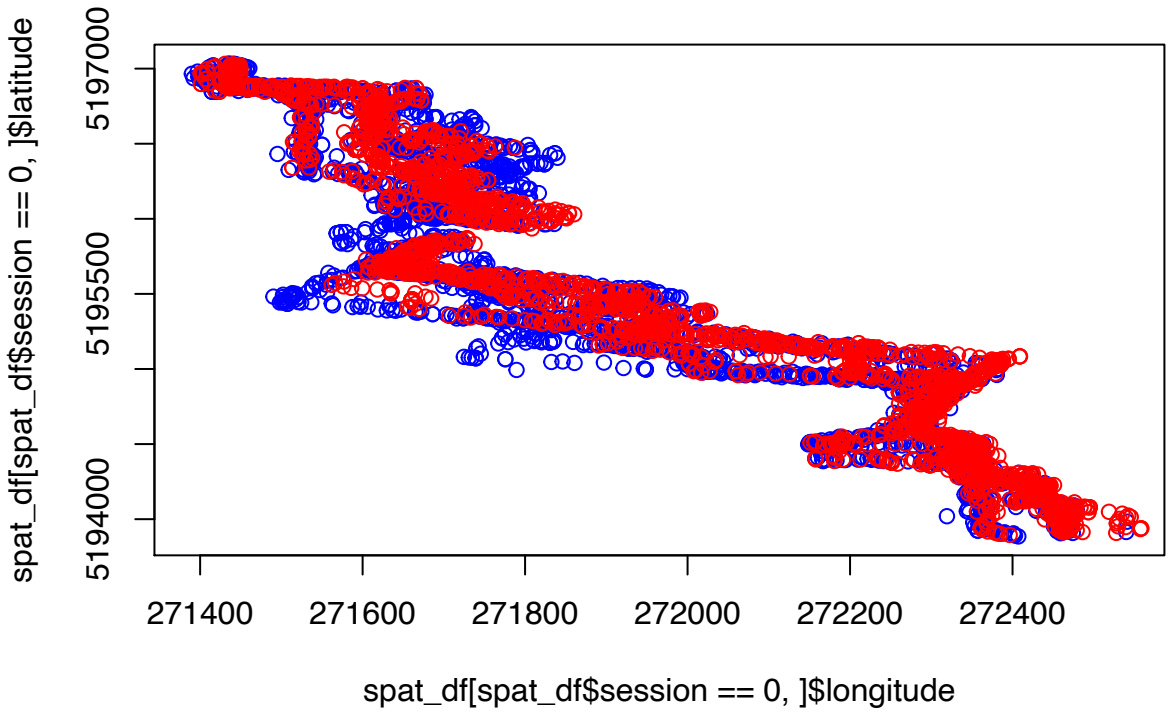


For visualization purposes, I hand-code each session as either 0 (home location to work location) or 1 (work location to home location).

```
spat_df$session <- rep(NA,nrow(spat_df@data))
spat_df$session[1:80] <- 0
spat_df$session[81:90] <- 1
spat_df$session[91:213] <- 0
spat_df$session[214:339] <- 1
spat_df$session[340:564] <- 0
spat_df$session[565:693] <- 1
spat_df$session[694:1021] <- 0
spat_df$session[1022:1360] <- 1
spat_df$session[1361:1722] <- 0
spat_df$session[1723:2044] <- 1
spat_df$session[2045:2403] <- 0
spat_df$session[2404:2755] <- 1
spat_df$session[2756:3117] <- 0
spat_df$session[3118:3533] <- 1
spat_df$session[3534:3896] <- 0
spat_df$session[3897:4245] <- 1
spat_df$session[4246:4512] <- 0
spat_df$session[4513:4573] <- 1
spat_df$session[4574:4903] <- 0
spat_df$session[4904:5264] <- 1
```

```
spat_df$session[5265:5593] <- 0

plot(spat_df[spat_df$session==0,]$longitude,spat_df[spat_df$session==0,]$latitude,col='blue')
points(spat_df[spat_df$session==1,]$longitude,spat_df[spat_df$session==1,]$latitude,col='red')
```



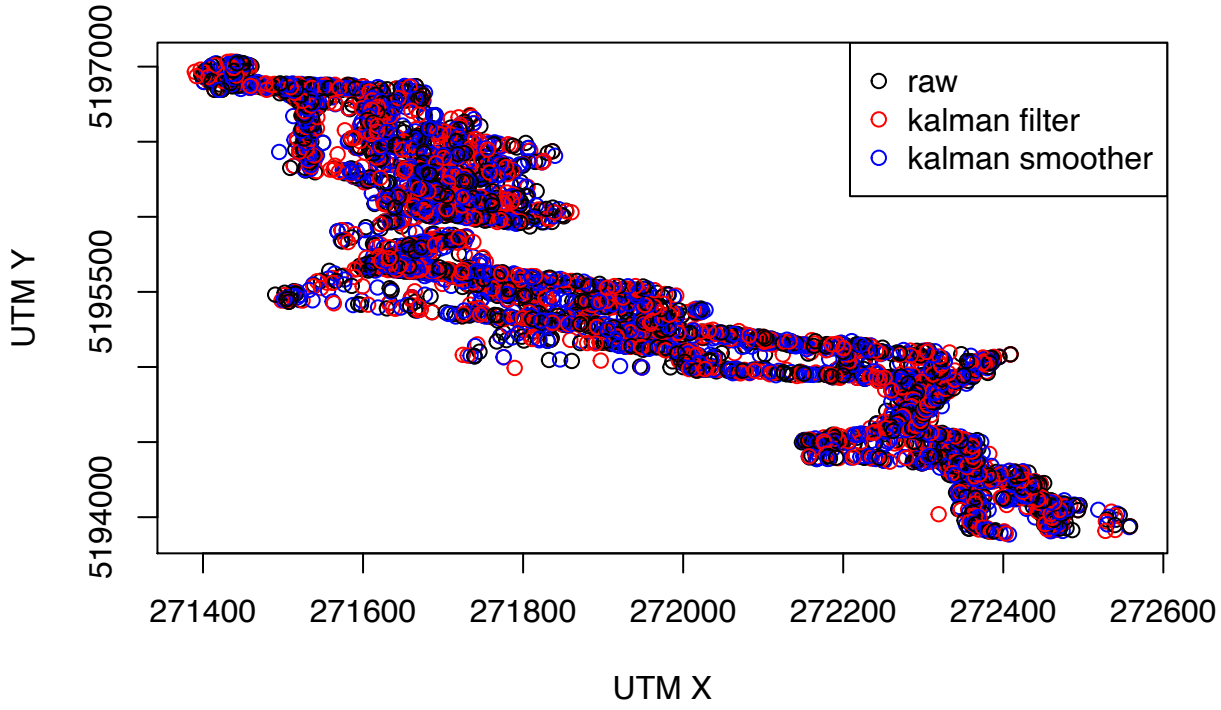
In blue are the data points for the commute to work, and mirroring them are the red data points representing travel from work to home.

Having examined the period between individual data points, defined "sessions" of activity, and removed sessions containing unwanted data, we can now get a better picture of the speed between the data points we're interested in. For this, we'll construct a DLM (dynamic linear model) – also known as a "Kalman filter," which uses uses what we know about our longitudinal data and its uncertainties to create refined time and position estimates.

```
td <- as.numeric(diff(df$time), units="secs")
gps_variance <- 20^2
v_mat <-diag(c(gps_variance, gps_variance))
f_mat <-matrix(c(1,0,0,0, 0,1,0,0), nrow=2, byrow = TRUE)
dt <- median(td)
g_mat <-matrix(c(1, 0, dt, 0,0, 1, 0, dt,0, 0, 1, 0,0, 0, 0, 1), byrow=TRUE, ncol=4)
avg_walk_speed_m_per_sec <- median(df$speed,na.rm=TRUE) # Average time
dlm_spec <-dlm(FF= f_mat,GG= g_mat,V= v_mat,W =diag(c(5, 5, 1, 1)^2),
               m0= matrix(c(coordinates[1, ],rep(avg_walk_speed_m_per_sec/dt, 2)),ncol=1),
               C0= diag(rep(10^2, 4)))
dlm_filter_mod <-dlmFilter(coordinates(spat_df), dlm_spec)
dlm_smooth_mod <-dlmSmooth(dlm_filter_mod)
```

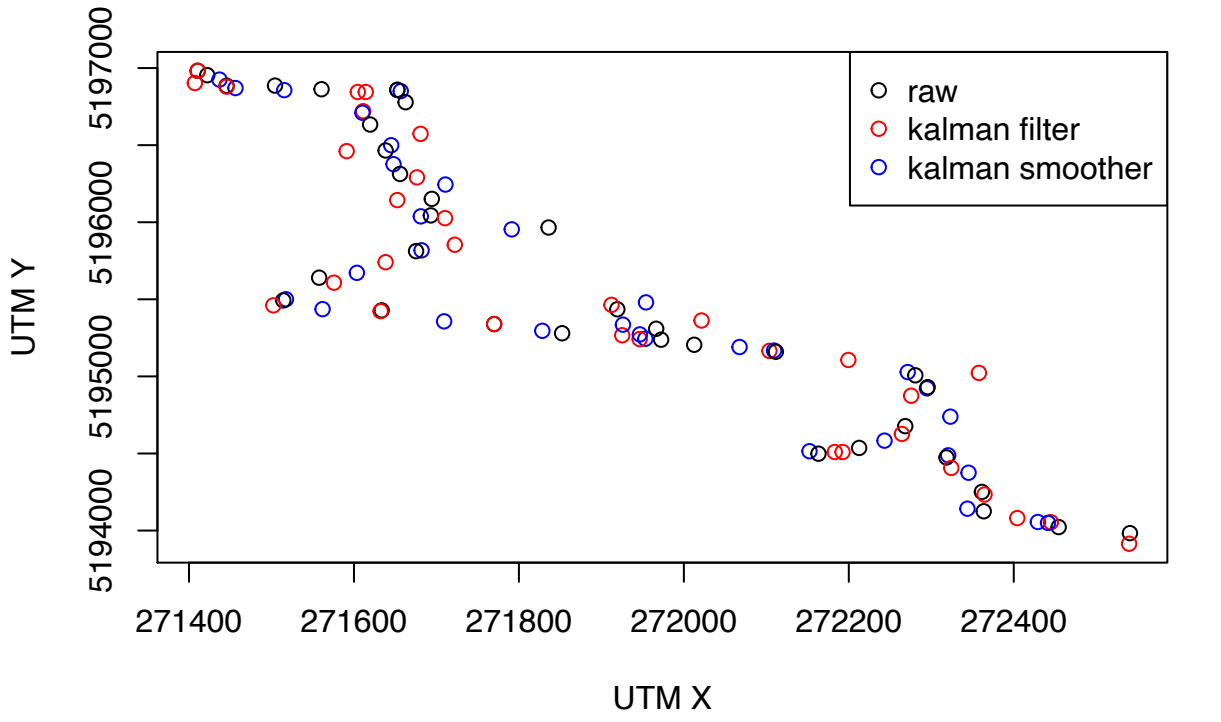
Below are the full raw data points, Kalman filter estimates, and Kalman smoother estimates.

```
plot(cbind(coordinates(spat_df)[1:nrow(coordinates(spat_df)), ], dlm_filter_mod$m[2:nrow(dlm_filter_mod$
     type='p', col =c("black", "red", "blue"), xlab="UTM X", ylab="UTM Y")
legend("topright", col =c("black", "red", "blue"),pch = 1,
       legend =c("raw", "kalman filter","kalman smoother"))
```



To get a clearer view, we plot the first 100 observations of each:

```
plot(cbind(coordinates(spat_df)[1:100,], dlm_filter_mod$m[2:101, 1:2], dlm_smooth_mod$s[2:101,1:2]),
     type='p', col =c("black", "red", "blue"), xlab="UTM X", ylab="UTM Y")
legend("topright", col =c("black", "red", "blue"),pch = 1,
       legend =c("raw", "kalman filter","kalman smoother"))
```



To calculate speed, we use the following equation:

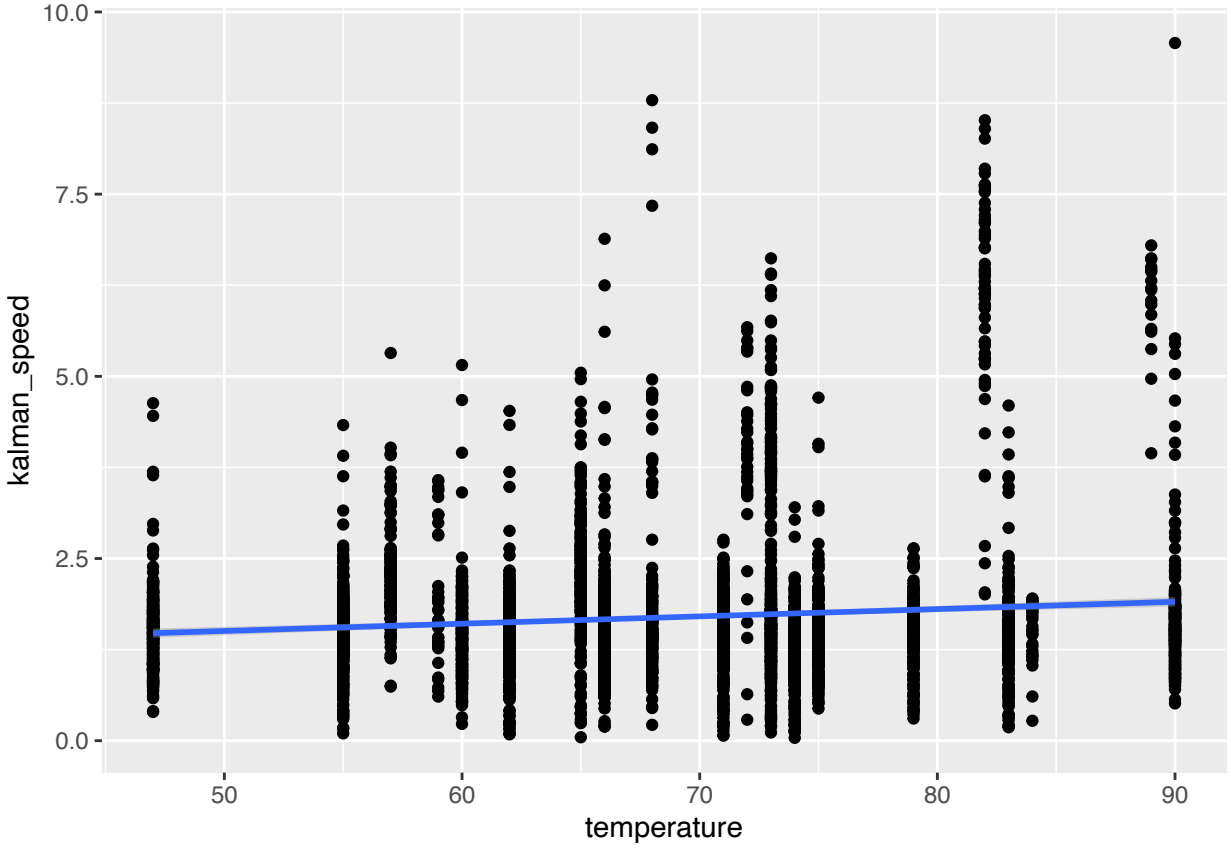
```r
speed <- sqrt(dlm_smooth_mod$s[, 3]^2+dlm_smooth_mod$s[, 4]^2)
```

```r
time$kalman_speed <- speed[2:length(speed)]
time <- time[time$kalman_speed<10,] # Removing unrealistic speeds
ggplot(time) +
  geom_point(aes(x=temperature,y=kalman_speed)) +
  geom_smooth(aes(x=temperature,y=kalman_speed),method='lm')
```

```
## `geom_smooth()` using formula 'y ~ x'
```

```
## Warning: Removed 1033 rows containing non-finite values (stat_smooth).
```

```
## Warning: Removed 1033 rows containing missing values (geom_point).
```




```r
summary(lm(data=time,kalman_speed~temperature))
```

```
##
## Call:
## lm(formula = kalman_speed ~ temperature, data = time)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -1.7068 -0.5136 -0.1929  0.1976  7.6680
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 1.005200   0.094546  10.632  < 2e-16 ***
## temperature 0.010011   0.001367   7.321  2.9e-13 ***
```

```
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.9953 on 4535 degrees of freedom
##   (1033 observations deleted due to missingness)
## Multiple R-squared:  0.01168,    Adjusted R-squared:  0.01146
## F-statistic: 53.59 on 1 and 4535 DF,  p-value: 2.903e-13
```

After removing unwanted data points and modeling the data using a "Kalman filter" to calculate our own speed values, we observe the opposite trend: that each increase in Farenheit is associated with a 0.01 unit *increase* in speed – possibly indicating Wayne hurried along when it was getting hotter.

Model Selection

Before beginning to select a bomb prediction model, we must narrow our data to only those data points after the first 5 minutes of Wayne's travel:

```
df$longitude <- spat_df@coords[,1]
df$latitude <- spat_df@coords[,2]
new_df <- data.frame()
for (i in split(df,df$day)) {
  for (j in 1:nrow(i)) {
    if (i$time[j] > i[1,]$time + 300) { # First 5 minutes
      new_df <- rbind(new_df,i[j,])
    }
  }
}
df <- new_df
```

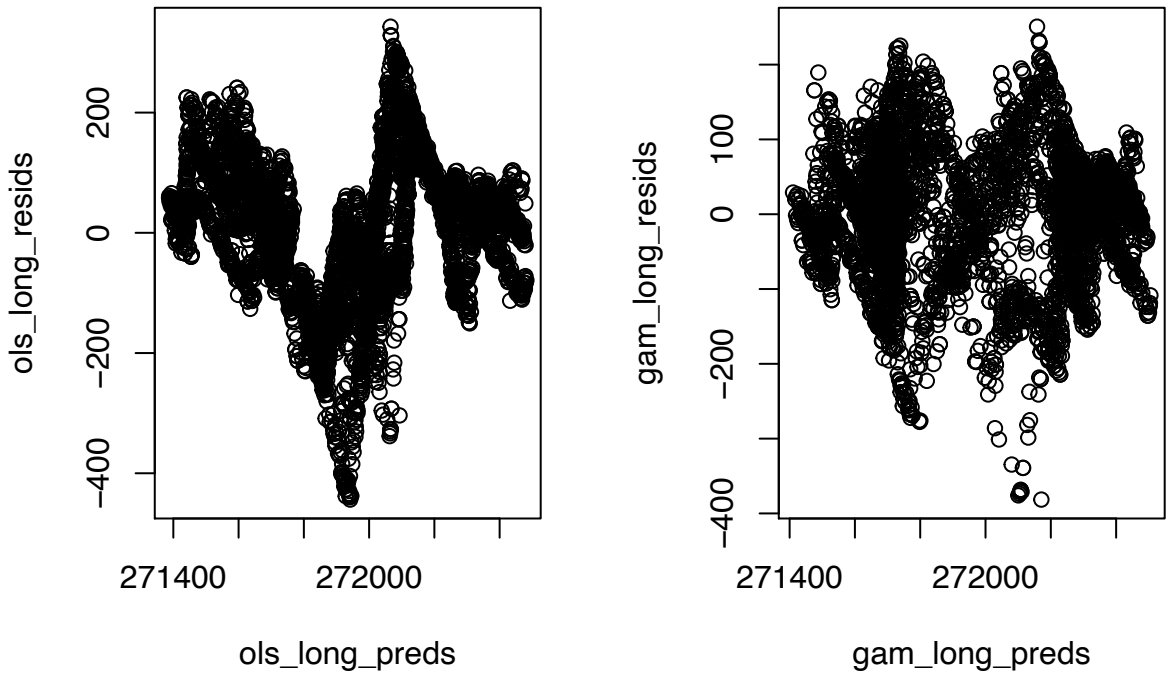
This portion of the report will focus on selecting a model to predict values for Wayne's trips. It will predict three values necessary to plant a bomb: time, longitude, and latitude.

As predicting based on Kalman filter alone (dlm::dlmForecast) produced extremely biased predictions, I narrowed my model choices down to two basic forms: OLS regression (which assumes a linear relationship between variables, and attempts to minimize squared errors), and a generalized additive model (GAM), which is a generalized linear model that "smooths," or transforms, variables by applying a function to them. I begin by creating both models to explain longitude data (given latitude and time):

```
ols_long <- lm(data=df,longitude~latitude+time) # OLS
ols_long_preds <- predict(ols_long)
ols_long_resids <- df$longitude - ols_long_preds

gam_long <- mgcv::gam(longitude~s(latitude)+s(time_long),data=df) # GAM
gam_long_preds <- predict(gam_long)
gam_long_resids <- df$longitude - gam_long_preds

par(mfrow=c(1,2))
plot(ols_long_preds,ols_long_resids)
plot(gam_long_preds,gam_long_resids)
```



The OLS model (left) produces small errors at certain points, but wildly inaccurate guesses at other points. The GAM model (right) applies "smoothing" to latitude and time data, and thus creates a lot of data with modest amounts of error. While both models have some degree of accuracy, I end up choosing the GAM model because its errors appear to be consistently smaller:

```
summary(ols_long_resids); summary(gam_long_resids)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## -443.79  -75.25   16.95    0.00   76.50  342.94
```

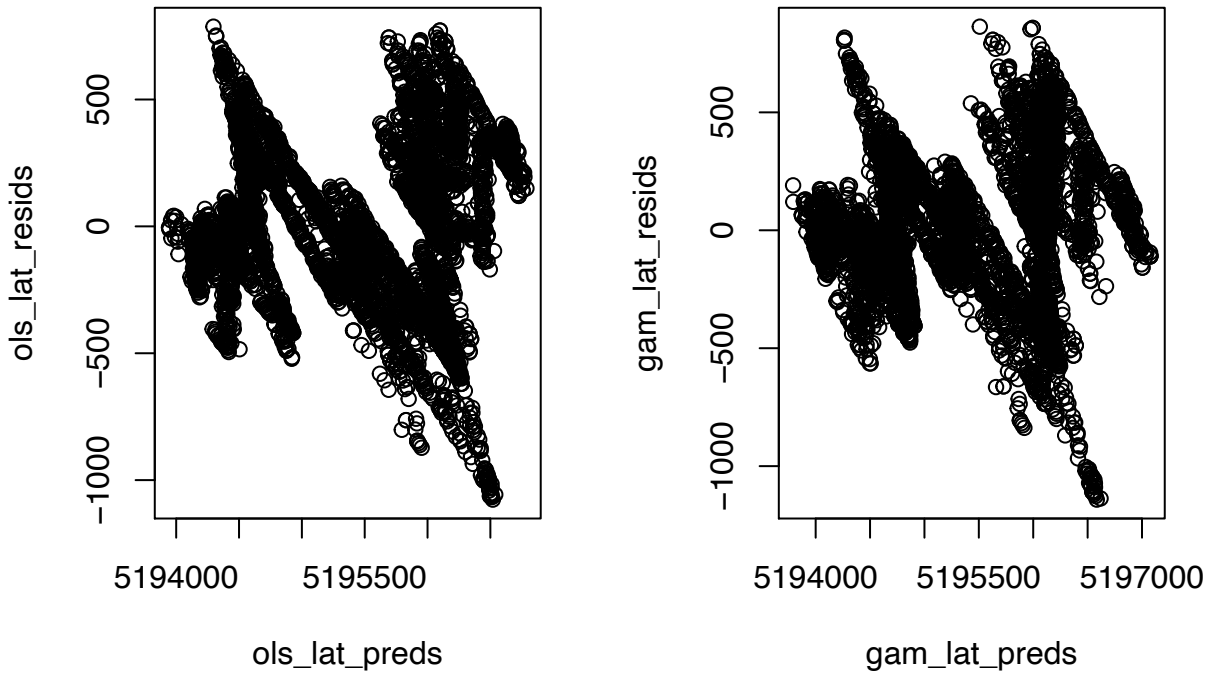
```
##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
## -381.475  -44.043    6.679    0.000   47.815  250.732
```

The following code applies OLS and GAM models to the other variables – latitude and time elapsed.

```
ols_lat <- lm(data=df,latitude~longitude+time) # OLS
ols_lat_preds <- predict(ols_lat)
ols_lat_resids <- df$latitude - ols_lat_preds

gam_lat<- mgcv::gam(latitude~s(longitude)+s(time_long),data=df) # GAM
gam_lat_preds <- predict(gam_lat)
gam_lat_resids <- df$latitude - gam_lat_preds

par(mfrow=c(1,2))
plot(ols_lat_preds,ols_lat_resids)
plot(gam_lat_preds,gam_lat_resids)
```




```
summary(ols_lat_resids); summary(gam_lat_resids)
```

```
##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
## -1076.77  -216.66   -14.33     0.00   244.26   786.93
```

```
##      Min.   1st Qu.    Median      Mean   3rd Qu.      Max.
## -1141.886  -190.575     4.553     0.000   192.792   863.248
```

After modeling longitude and latitude, I create visualizations to understand what the model is doing.

The following graph shows GAM model estimates and real longitude/latitude data for Week1/Week2.

```
ggplot() +
  geom_point(aes(x=df$longitude,y=df$latitude,col='blue')) +
  geom_point(aes(x=gam_long_preds,y=gam_lat_preds,col='red')) +
  scale_color_discrete(name="",
                       labels=c("real","gam predictions"),
                       breaks=c("blue","red"))
```

The following graph shows the GAM model's estimates for longitude/latitude, and in red lines plots our uncertainties about the model's predictions. To be precise, the red lines are ~95% credible intervals = estimate +/- 2*SE (as recommended by the mgcv package).

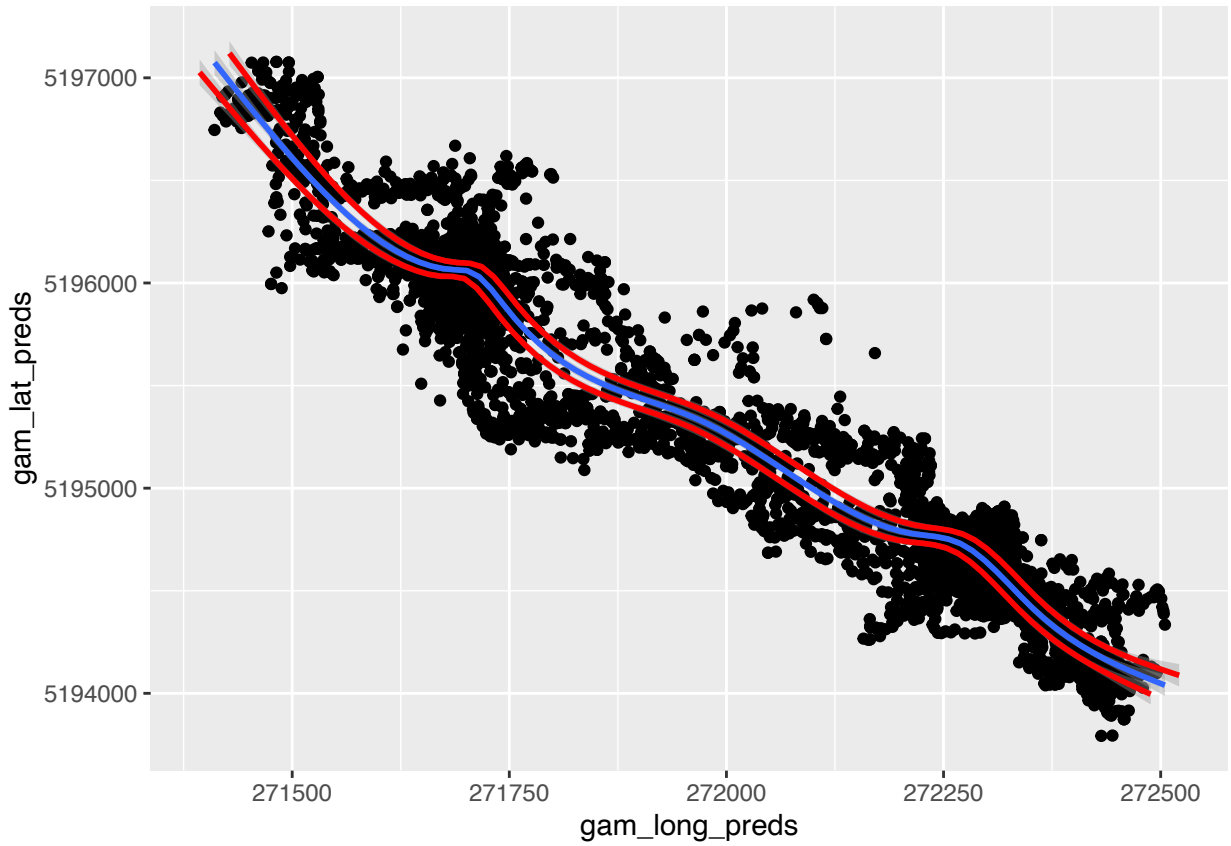
```
long_se <- predict(gam_long,se.fit=T)$se
long_lwr <- predict(gam_long) - (2*long_se)
long_upr <- predict(gam_long) + (2*long_se)

lat_se <- predict(gam_lat,se.fit=T)$se
lat_lwr <- predict(gam_lat) - (2*lat_se)
lat_upr <- predict(gam_lat) + (2*lat_se)

ggplot() +
  geom_point(aes(x=gam_long_preds,y=gam_lat_preds)) +
  geom_smooth(method='gam',mapping=aes(x=gam_long_preds,y=gam_lat_preds)) +
  geom_smooth(col='red',aes(x=long_lwr,y=lat_lwr)) +
  geom_smooth(col='red',aes(x=long_upr,y=lat_upr))
```

```
## `geom_smooth()` using formula 'y ~ s(x, bs = "cs")'
```

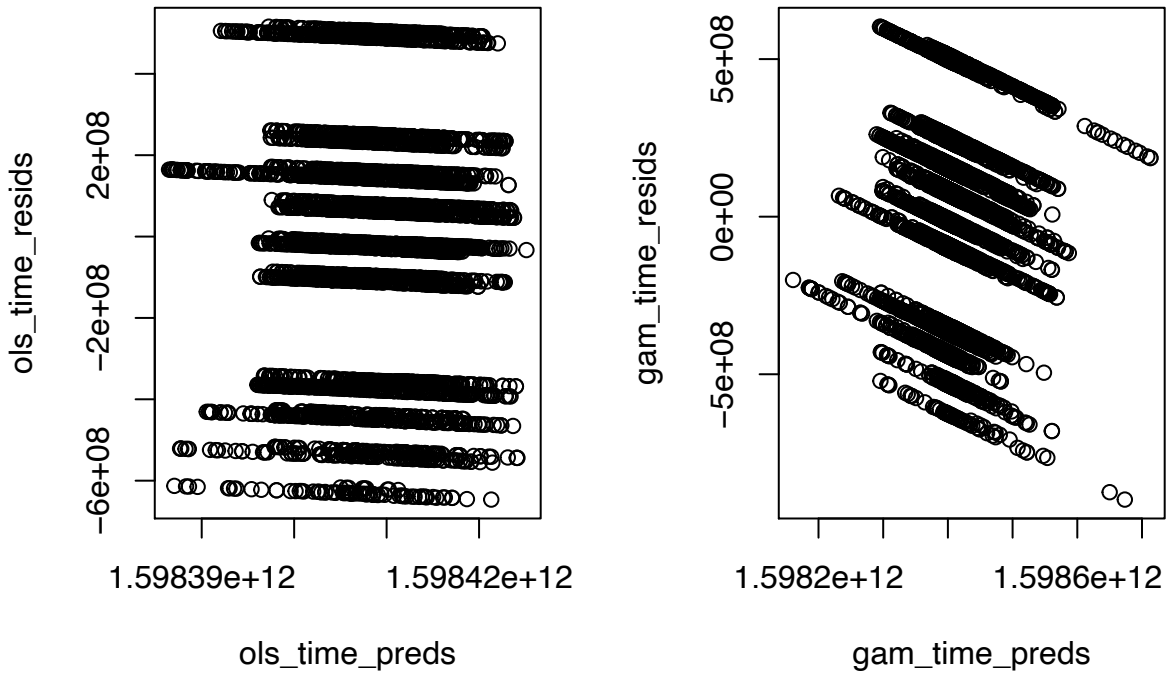
```
## `geom_smooth()` using method = 'gam' and formula 'y ~ s(x, bs = "cs")'
## `geom_smooth()` using method = 'gam' and formula 'y ~ s(x, bs = "cs")'
```




```
ols_time <- lm(data=df,time_long~longitude+latitude) # OLS
ols_time_preds <- predict(ols_time)
ols_time_resids <- df$time_long - ols_time_preds

gam_time <- mgcv::gam(time_long~s(longitude)+s(latitude),data=df) # GAM
gam_time_preds <- predict(gam_time)
gam_time_resids <- df$time_long - gam_time_preds

par(mfrow=c(1,2))
plot(ols_time_preds,ols_time_resids)
plot(gam_time_preds,gam_time_resids)
```




```
summary(ols_time_resids); summary(gam_time_resids)
```

```
##        Min.     1st Qu.      Median        Mean     3rd Qu.        Max.
## -646460929 -117203893   50389678           0  164258395  515848575
```

```
##        Min.     1st Qu.      Median        Mean     3rd Qu.        Max.
## -897636051 -172019688   32500324           0  181034297  602843432
```

Having created prediction algorithms, to select bomb locations, I choose the place where the prediction's error for longitude and latitude (their summed absolute value, to be precise) is smallest.

```
errors <- bind_cols(long=as.vector(gam_long_resids),lat=as.vector(gam_lat_resids))

errors$comb <- abs(errors$long)+abs(errors$lat)

#which(errors$comb==min(errors$comb))
#df[2966,]
#errors[2966,]

#which(errors$comb==sort(errors$comb)[2])
#df[1721,]
#errors[1721,]

#which(errors$comb==sort(errors$comb)[3])
#df[4482,]
#errors[1721,]
```

Bomb 1:

Bomb one will be placed at the following location and time:

```
paste("UTM Longitude:",gam_long_preds[which(errors$comb==min(errors$comb))])
```

```
## [1] "UTM Longitude: 272433.825819921"
```

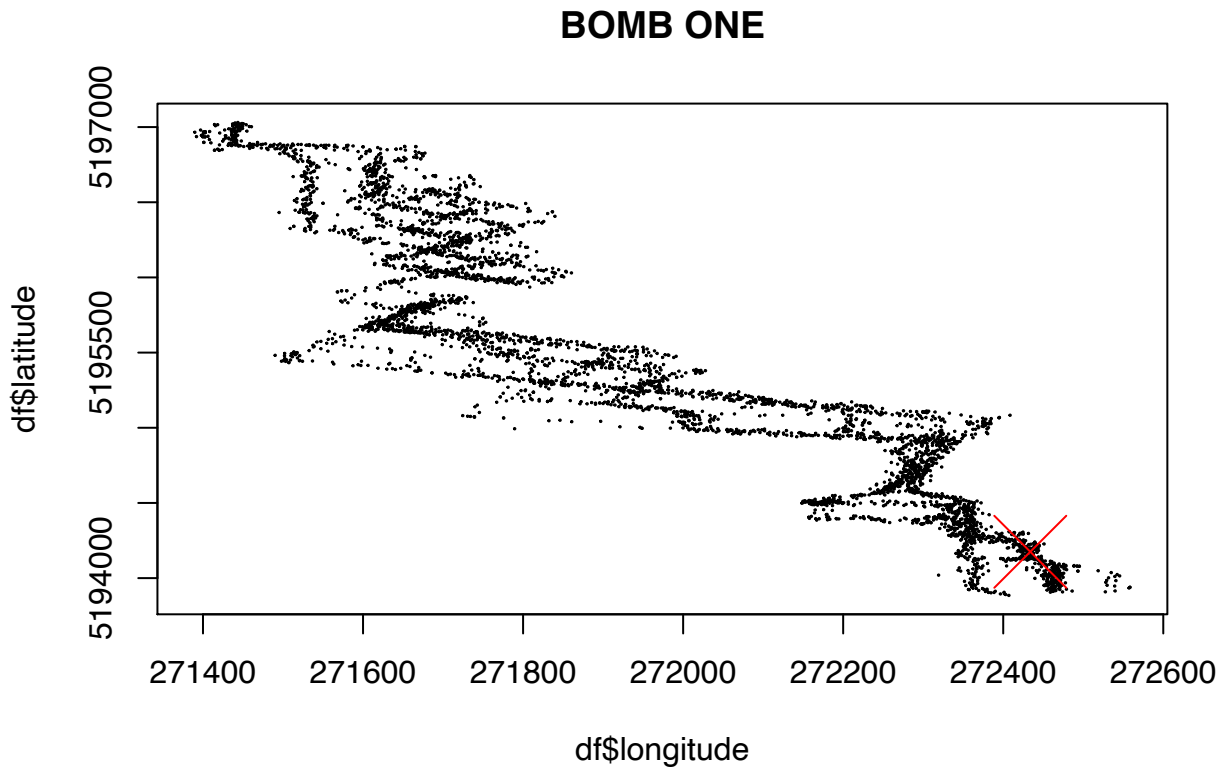
```
paste("UTM Latitude:",gam_lat_preds[which(errors$comb==min(errors$comb))])
```

```
## [1] "UTM Latitude: 5194175.18348023"
```

```
paste("UNIX Time:",(86400*7)+
        gam_time_preds[which(errors$comb==min(errors$comb))]) # Adding 7 days
```

```
## [1] "UNIX Time: 1598443910440.11"
```

```
plot(cex=0.1,df$longitude,df$latitude,main="BOMB ONE")
points(gam_long_preds[which(errors$comb==min(errors$comb))],
       gam_lat_preds[which(errors$comb==min(errors$comb))],cex=5,pch=4,col='red')
```



Bomb 2:

Bomb two will be placed at the following location and time:

```
paste("UTM Longitude:",gam_long_preds[which(errors$comb==sort(errors$comb)[3])])
```

```
## [1] "UTM Longitude: 271706.407611829"
```

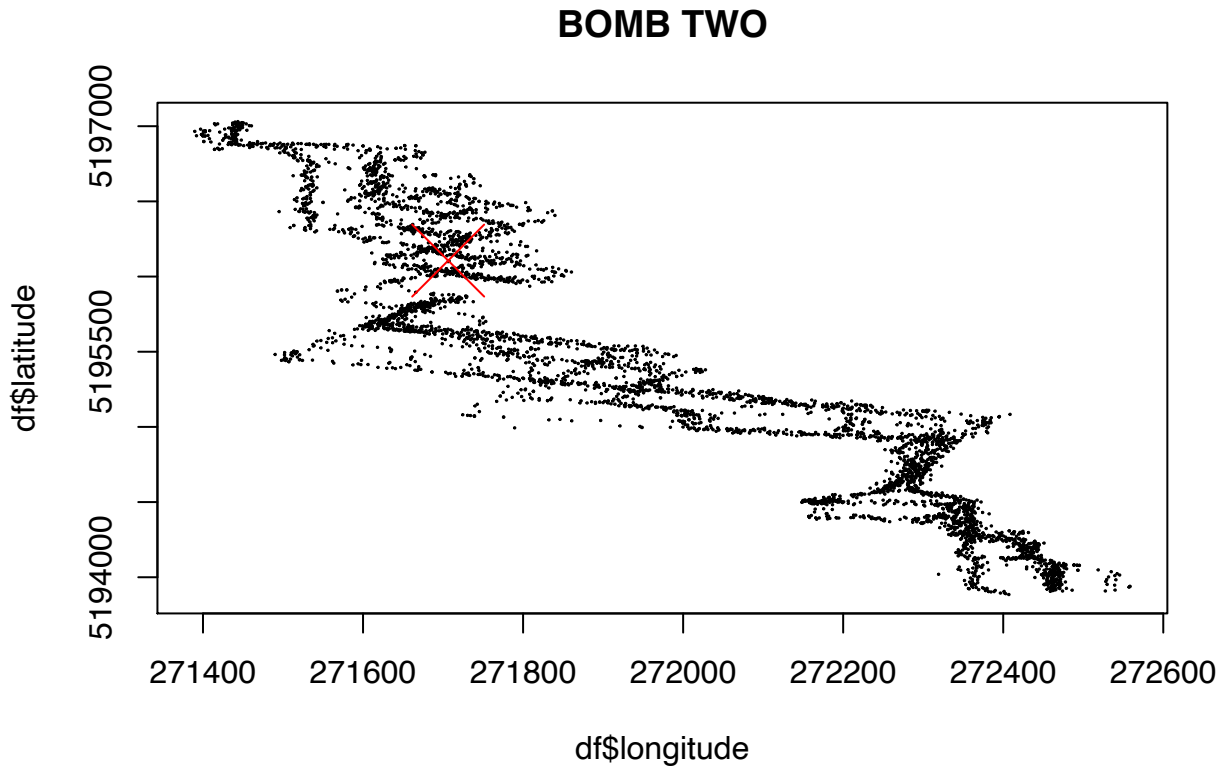
```
paste("UTM Latitude:",gam_lat_preds[which(errors$comb==sort(errors$comb)[3])])
```

```
## [1] "UTM Latitude: 5196106.15567278"
```

```
paste("UNIX Time:",(86400*7)+
        gam_time_preds[which(errors$comb==sort(errors$comb)[3])]) # Adding 7 days
```

```
## [1] "UNIX Time: 1598402271976.34"
```

```
plot(cex=0.1,df$longitude,df$latitude,main="BOMB TWO")
points(gam_long_preds[which(errors$comb==sort(errors$comb)[3])],
       gam_lat_preds[which(errors$comb==sort(errors$comb)[3])],cex=5,pch=4,col='red')
```



Applying Model Using Week 3 Initial Points

The data below imports files from the "test_gps" folder containing initial data points for each day in Week 3, and generates new predictions for bomb locations given additional information added to the GAM model.

```
df <- data.frame()
for (i in list.files("test_gps")) {
  setwd("~/inh2102_project2/test_gps")
  dat <- read_json(i,flatten=TRUE)
  time <- c()
  time_long <- c()
  accuracy <- c()
  altitude <- c()
  bearing <- c()
  speed <- c()
  longitude <- c()
  latitude <- c()
  for (i in dat[['features']]) {
    time <- c(time,i$properties[1])
    time_long <- c(time_long,i$properties[3])
    accuracy <- c(accuracy,i$properties[4])
    altitude <- c(altitude,i$properties[5])
    bearing <- c(bearing,i$properties[6])
    speed <- c(speed,i$properties[7])
```

```
    longitude <- c(longitude,i$geometry[[2]][1])
    latitude <- c(latitude,i$geometry[[2]][2])
  }
  altitude[sapply(altitude, is.null)] <- NA
  bearing[sapply(bearing, is.null)] <- NA
  speed[sapply(speed, is.null)] <- NA

  df <- bind_rows(df,tibble(time=unlist(time),time_long=unlist(time_long),
                  accuracy=unlist(accuracy),altitude=unlist(altitude),
                  bearing=unlist(bearing),
                 speed=unlist(speed),longitude=unlist(longitude),
                latitude=unlist(latitude))) %>% tibble()
}
```

```
df$time <- paste0(substr(df$time,1,10)," ",substr(df$time,12,23))
df$time <- as.POSIXct(df$time, "%Y-%m-%d %H:%M:%S",tz="GMT")
df$day <- lubridate::day(df$time)
df$hour <- lubridate::hour(df$time)

spat_df <- SpatialPointsDataFrame(coords = df[, c("longitude", "latitude")],
                              data = df["time"],
                              proj4string=CRS("+proj=longlat +datum=WGS84"))
spat_df@data <- spat_df@data %>%
  mutate(time_long=df$time_long,accuracy=df$accuracy,altitude=df$altitude,
         bearing=df$bearing,speed=df$speed,day=df$day,hour=df$hour) %>%
  mutate(longitude=spat_df@coords[,1],latitude=spat_df@coords[,2])
spat_df <- spTransform(spat_df,CRSobj = "+proj=utm +zone=12 +datum=WGS84")
df$longitude <- coordinates(spat_df)[,1]
df$latitude <- coordinates(spat_df)[,2]
new_df <- data.frame()
for (i in split(df,df$day)) {
  for (j in 1:nrow(i)) {
    if (i$time[j] > i[1,]$time + 300) { # First 5 minutes
      new_df <- rbind(new_df,i[j,])
    }
  }
}
df <- new_df
```

```
init_long <- mgcv::gam(longitude~s(latitude,k=1)+s(time_long,k=1),data=df) # GAM
init_long_preds <- predict(init_long)
init_long_resids <- df$longitude - init_long_preds

init_lat <- mgcv::gam(latitude~s(longitude,k=1)+s(time_long,k=1),data=df) # GAM
init_lat_preds <- predict(init_lat)
init_lat_resids <- df$latitude - init_lat_preds

init_time <- mgcv::gam(time_long~s(longitude,k=1)+s(latitude,k=1),data=df) # GAM
init_time_preds <- predict(init_time)
init_time_resids <- df$time_long - init_time_preds
```

```
errors <- bind_cols(long=as.vector(init_long_resids),lat=as.vector(init_lat_resids))

errors$comb <- abs(errors$long)+abs(errors$lat)
```

Bomb 1:

```
paste("UTM Longitude:",init_long_preds[which(errors$comb==min(errors$comb))])
```

```
## [1] "UTM Longitude: 272442.601838387"
```

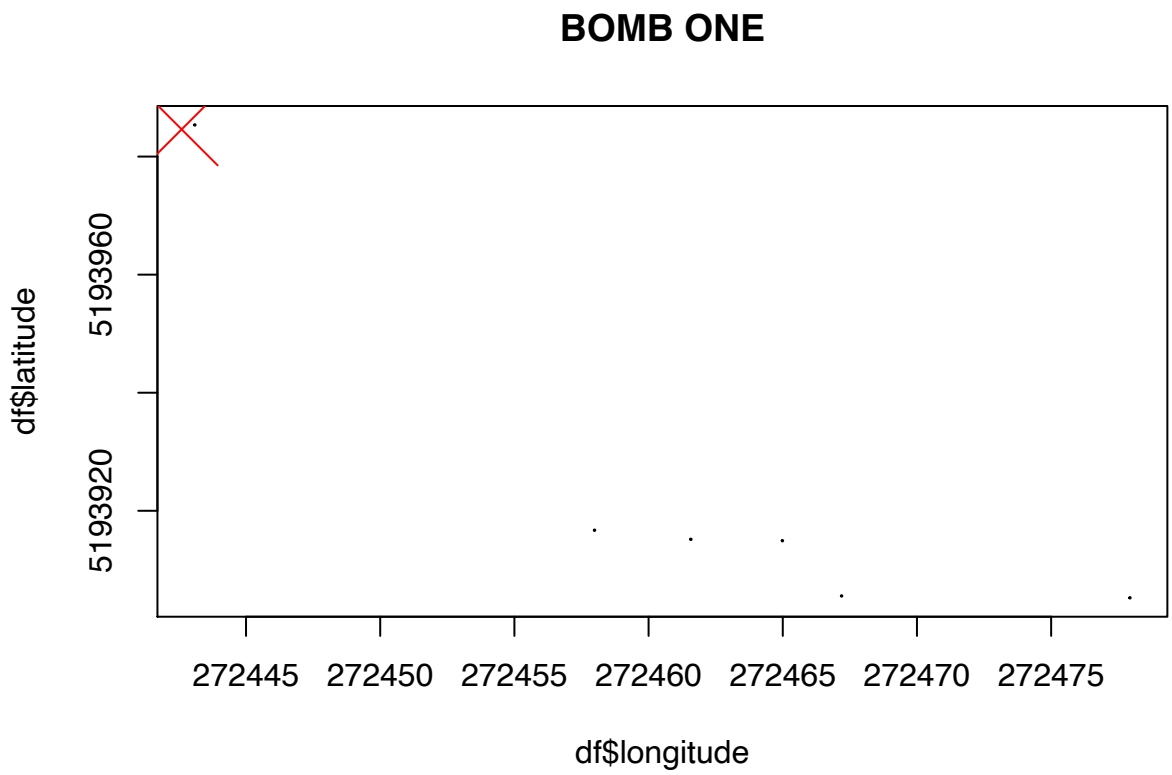
```
paste("UTM Latitude:",init_lat_preds[which(errors$comb==min(errors$comb))])
```

```
## [1] "UTM Latitude: 5193984.61005781"
```

```
paste("UNIX Time:",init_time_preds[which(errors$comb==min(errors$comb))])
```

```
## [1] "UNIX Time: 1600136945672.06"
```

```
plot(cex=0.1,df$longitude,df$latitude,main="BOMB ONE")
points(init_long_preds[which(errors$comb==min(errors$comb))],
       init_lat_preds[which(errors$comb==min(errors$comb))],cex=5,pch=4,col='red')
```



Bomb 2:

```
paste("UTM Longitude:",init_long_preds[which(errors$comb==sort(errors$comb)[2])])
```

```
## [1] "UTM Longitude: 272462.504919879"
```

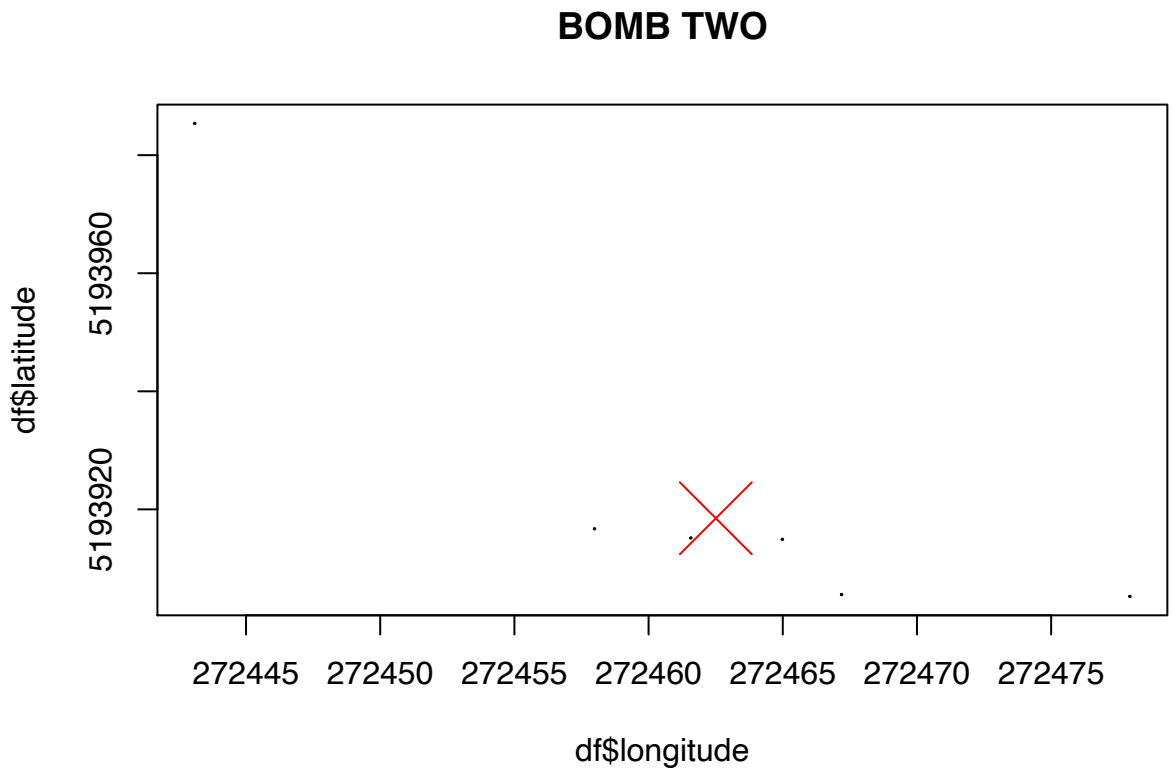
```
paste("UTM Latitude:",init_lat_preds[which(errors$comb==sort(errors$comb)[2])])
```

```
## [1] "UTM Latitude: 5193918.4891058"
```

```
paste("UNIX Time:",init_time_preds[which(errors$comb==sort(errors$comb)[2])])
```

```
## [1] "UNIX Time: 1599479065663.86"
```

```
plot(cex=0.1,df$longitude,df$latitude,main="BOMB TWO")
points(init_long_preds[which(errors$comb==sort(errors$comb)[2])],
       init_lat_preds[which(errors$comb==sort(errors$comb)[2])],cex=5,pch=4,col='red')
```



The following function allows the user to input the name of a folder containing geojson folders and obtain my bomb predictions.

```
make_pred <- function(folder_name) {
  setwd(paste0("~/inh2102_project2/",folder_name))
  df <- data.frame()
for (i in list.files(getwd())) {
  setwd(paste0("~/inh2102_project2/",folder_name))
  dat <- read_json(i,flatten=TRUE)
  time <- c()
  time_long <- c()
  accuracy <- c()
  altitude <- c()
  bearing <- c()
  speed <- c()
  longitude <- c()
  latitude <- c()
  for (i in dat[['features']]) {
    time <- c(time,i$properties[1])
    time_long <- c(time_long,i$properties[3])
    accuracy <- c(accuracy,i$properties[4])
    altitude <- c(altitude,i$properties[5])
    bearing <- c(bearing,i$properties[6])
    speed <- c(speed,i$properties[7])
    longitude <- c(longitude,i$geometry[[2]][1])
```

```
    latitude <- c(latitude,i$geometry[[2]][2])
  }
  altitude[sapply(altitude, is.null)] <- NA
  bearing[sapply(bearing, is.null)] <- NA
  speed[sapply(speed, is.null)] <- NA

  df <- bind_rows(df,tibble(time=unlist(time),time_long=unlist(time_long),
                  accuracy=unlist(accuracy),altitude=unlist(altitude),
                  bearing=unlist(bearing),
                speed=unlist(speed),longitude=unlist(longitude),
               latitude=unlist(latitude))) %>% tibble()
}

df$time <- paste0(substr(df$time,1,10)," ",substr(df$time,12,23))
df$time <- as.POSIXct(df$time, "%Y-%m-%d %H:%M:%S",tz="GMT")
df$day <- lubridate::day(df$time)
df$hour <- lubridate::hour(df$time)

spat_df <- SpatialPointsDataFrame(coords = df[, c("longitude", "latitude")],
                            data = df["time"],
                            proj4string=CRS("+proj=longlat +datum=WGS84"))
spat_df@data <- spat_df@data %>%
  mutate(time_long=df$time_long,accuracy=df$accuracy,altitude=df$altitude,
         bearing=df$bearing,speed=df$speed,day=df$day,hour=df$hour) %>%
  mutate(longitude=spat_df@coords[,1],latitude=spat_df@coords[,2])
spat_df <- spTransform(spat_df,CRSobj = "+proj=utm +zone=12 +datum=WGS84")
df$longitude <- coordinates(spat_df)[,1]
df$latitude <- coordinates(spat_df)[,2]
new_df <- data.frame()
for (i in split(df,df$day)) {
  for (j in 1:nrow(i)) {
    if (i$time[j] > i[1,]$time + 300) { # First 5 minutes
      new_df <- rbind(new_df,i[j,])
    }
  }
}
df <- new_df

init_long <- mgcv::gam(longitude~s(latitude,k=1)+s(time_long,k=1),data=df) # GAM
init_long_preds <- predict(init_long)
init_long_resids <- df$longitude - init_long_preds

init_lat <- mgcv::gam(latitude~s(longitude,k=1)+s(time_long,k=1),data=df) # GAM
init_lat_preds <- predict(init_lat)
init_lat_resids <- df$latitude - init_lat_preds

init_time <- mgcv::gam(time_long~s(longitude,k=1)+s(latitude,k=1),data=df) # GAM
init_time_preds <- predict(init_time)
init_time_resids <- df$time_long - init_time_preds

errors <- bind_cols(long=as.vector(init_long_resids),lat=as.vector(init_lat_resids))

errors$comb <- abs(errors$long)+abs(errors$lat)
```

```
print(paste("BOMB1 UTM Longitude:",init_long_preds[which(errors$comb==min(errors$comb))]))
print(paste("BOMB1 UTM Latitude:",init_lat_preds[which(errors$comb==min(errors$comb))]))
print(paste("BOMB1 UNIX Time:",init_time_preds[which(errors$comb==min(errors$comb))]))

print(paste("BOMB2 UTM Longitude:",init_long_preds[which(errors$comb==sort(errors$comb)[2])]))
print(paste("BOMB2 UTM Latitude:",init_lat_preds[which(errors$comb==sort(errors$comb)[2])]))
print(paste("BOMB2 UNIX Time:",init_time_preds[which(errors$comb==sort(errors$comb)[2])]))
}
```

Example Input

This input replicates bomb locations based on Week1/Week2 data:

```
make_pred('gps')
```

```
## [1] "BOMB1 UTM Longitude: 272270.975951571"
## [1] "BOMB1 UTM Latitude: 5194645.36022651"
## [1] "BOMB1 UNIX Time: 1598445931835.93"
## [1] "BOMB2 UTM Longitude: 272279.326456823"
## [1] "BOMB2 UTM Latitude: 5194601.37228389"
## [1] "BOMB2 UNIX Time: 1598449228767.68"
```

Conclusion

This report describes my attempt to bomb Wayne and uses two weeks of GPS data to predict time and location for the third week. It's important to note that before beginning to build a set model, this report examined and attempted to correct elements of the two weeks of data. I identified unwanted data points – whether that was stationary activity that needed to be removed, or data with coordinates nowhere near the normal work to home route – and organized the data into "sessions" of activity. Before actually creating a prediction model, I explored the possible impact of weather on Wayne's walking speed, and used a special "dynamic linear model" to do so – concluding that after removing data, there was a significant relationship between temperature and speed. I then compared the performance of two model structures – OLS and GAM – and selected the GAM model to predict longitude, latitude, and time. Importantly, I described the uncertainties of the model before designating bomb times/locations, repeating the process for Week 3 initial points, and creating a function to run the model on Week 3 or any other data.