Project 2

Isaac Horwitz (inh2102)

10/31/2020

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)

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()
```

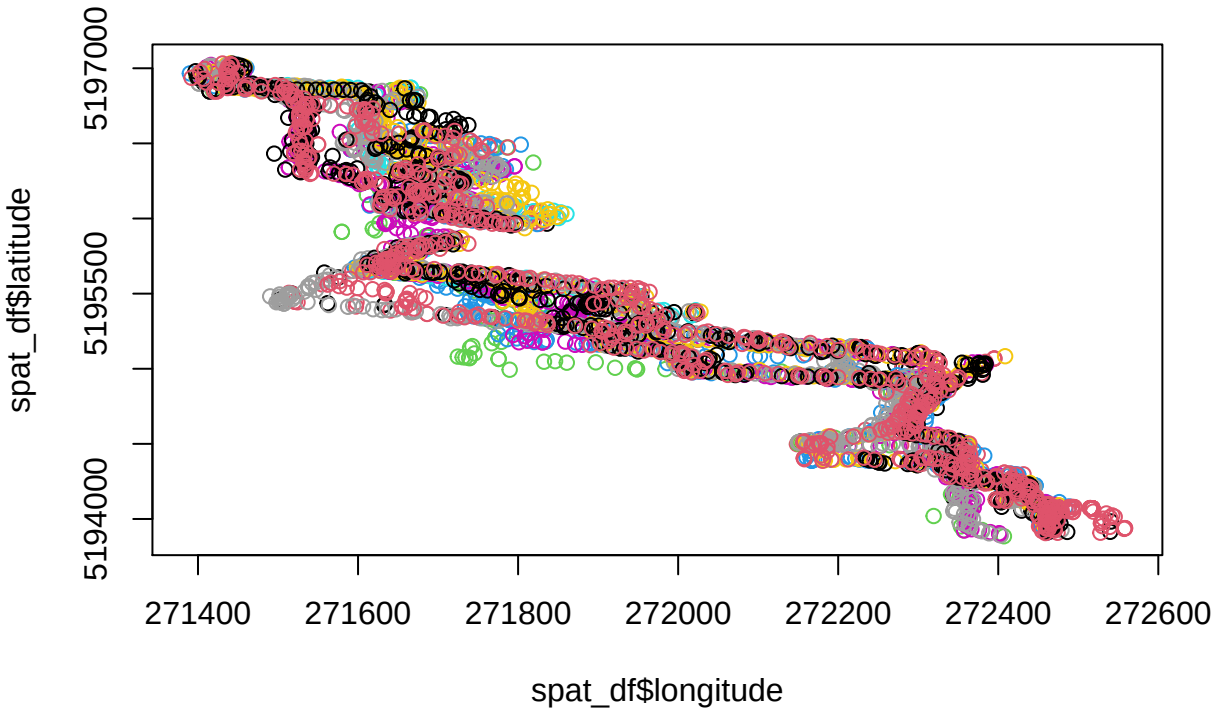
```
  longitude <- c()
  latitude <- c()
  for (i in dat[['features']]) {
    time <- c(time,i$properties[1])
    time_long <- c(time_long,i$properties[3])
    accuracy <- c(accuracy,i$properties[4])
    altitude <- c(altitude,i$properties[5])
    bearing <- c(bearing,i$properties[6])
    speed <- c(speed,i$properties[7])
    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")
```

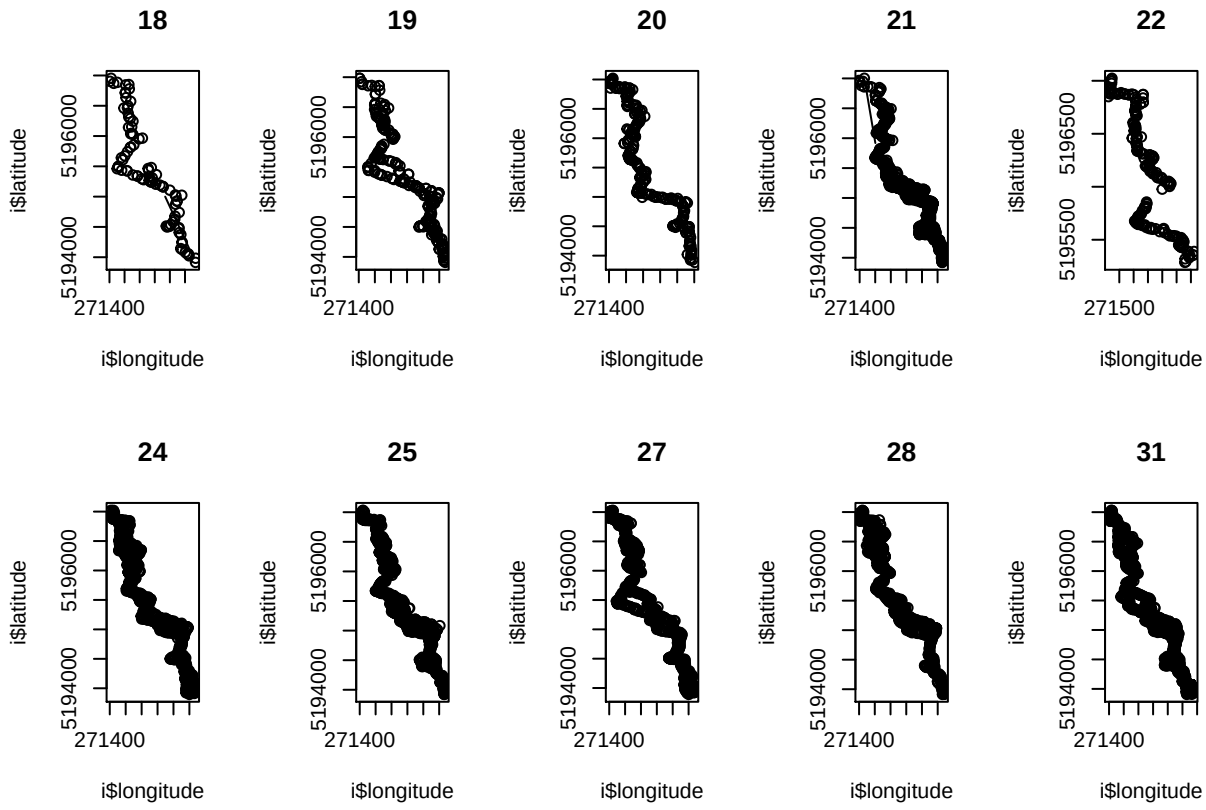
This is what the raw data looks like, with longitude on the X axis, latitude on the Y axis, and colors according to each day for which the raw data is labeled (assuming the default timezone GMT).

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



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(2,5))
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 is 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:

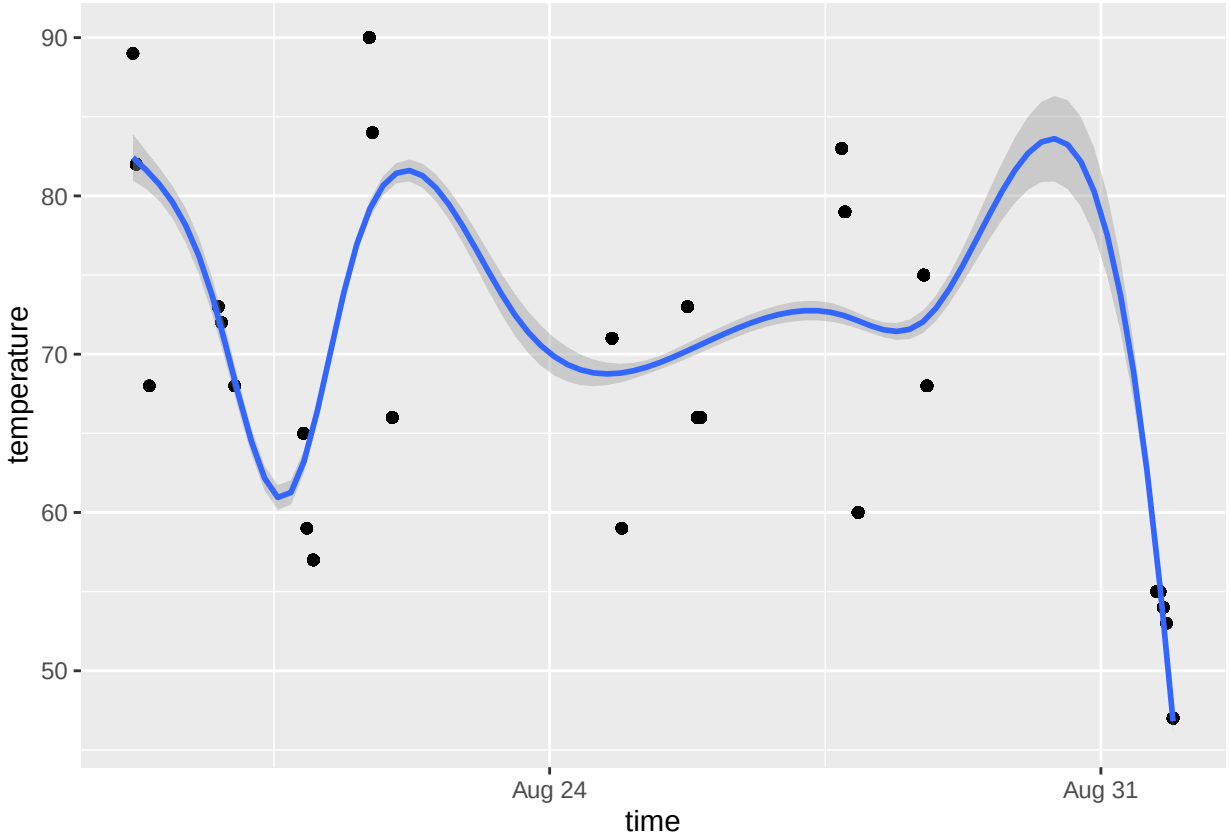
```
mt_weather$time <- format(mt_weather$time,"%D-%H")
df_time <- df %>% mutate(time = format(time,"%D-%H"))
time <- left_join(df_time,mt_weather,by='time') %>%
  mutate(time = lubridate::parse_date_time(time,orders='mdyh'))
ggplot(time,aes(x=time,y=temperature)) +
```

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

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

```
## Warning: Removed 955 rows containing non-finite values (stat_smooth).
```

```
## Warning: Removed 955 rows containing missing values (geom_point).
```

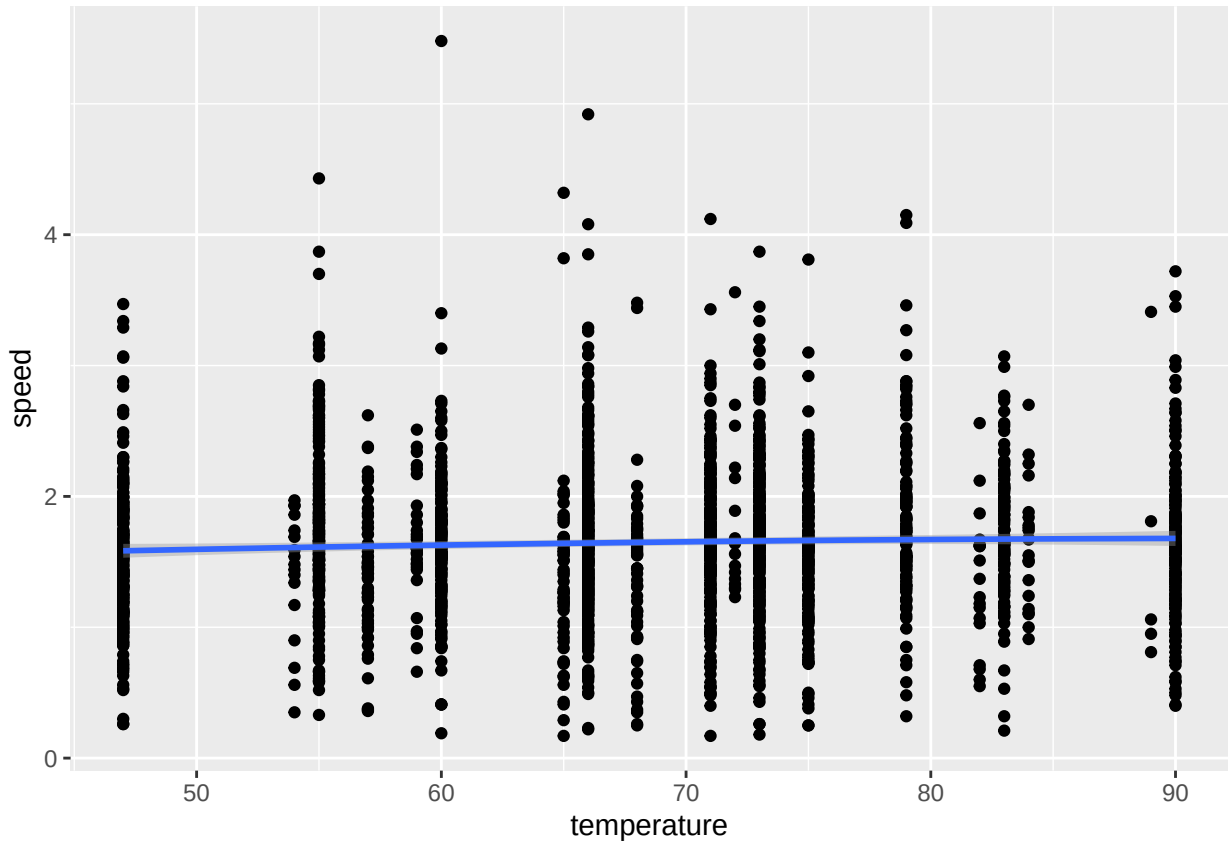


Using a locally fitted smoother (with grey bands representing SE, or uncertainty), we can see that there are fluctuations in temperature over the time period – it rises (during the morning) and drops at night, but over the entire time period, it goes down (as Montana transitions out of the summer).

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()
```

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




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

```
##
## Call:
## lm(formula = speed ~ temperature, data = time)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -1.4812 -0.3425 -0.0323  0.2888  3.8573
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 1.467152   0.071252  20.591   <2e-16 ***
## temperature 0.002593   0.001037   2.501   0.0125 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.5817 on 2285 degrees of freedom
##   (2705 observations deleted due to missingness)
## Multiple R-squared:  0.002729,   Adjusted R-squared:  0.002293
## F-statistic: 6.253 on 1 and 2285 DF,  p-value: 0.01247
```

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 increases by 0.003 units (possibly m/s, but not given in the data) for each increase in Farenheit temperature.

It's probably worthwhile calculating 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    228.6     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 (

$$time_i$$

-

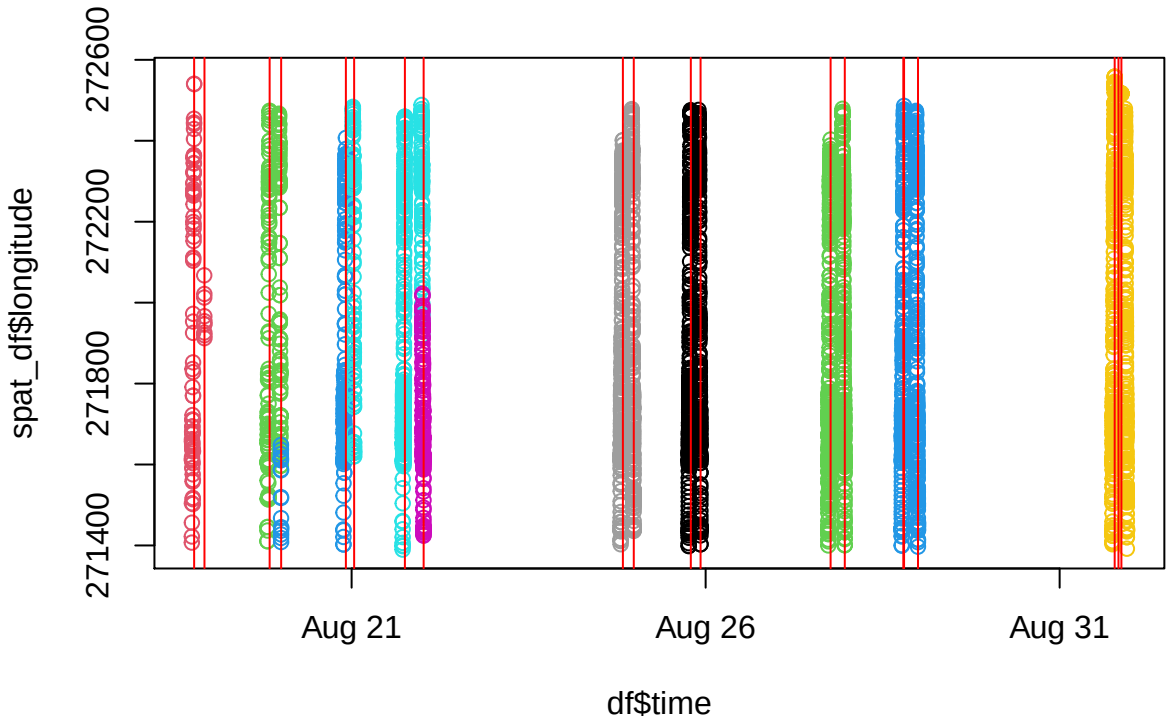
$$time_{i-1}$$

) < 120 seconds, meaning each session ends when data is not collected for 2 minutes or more.

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

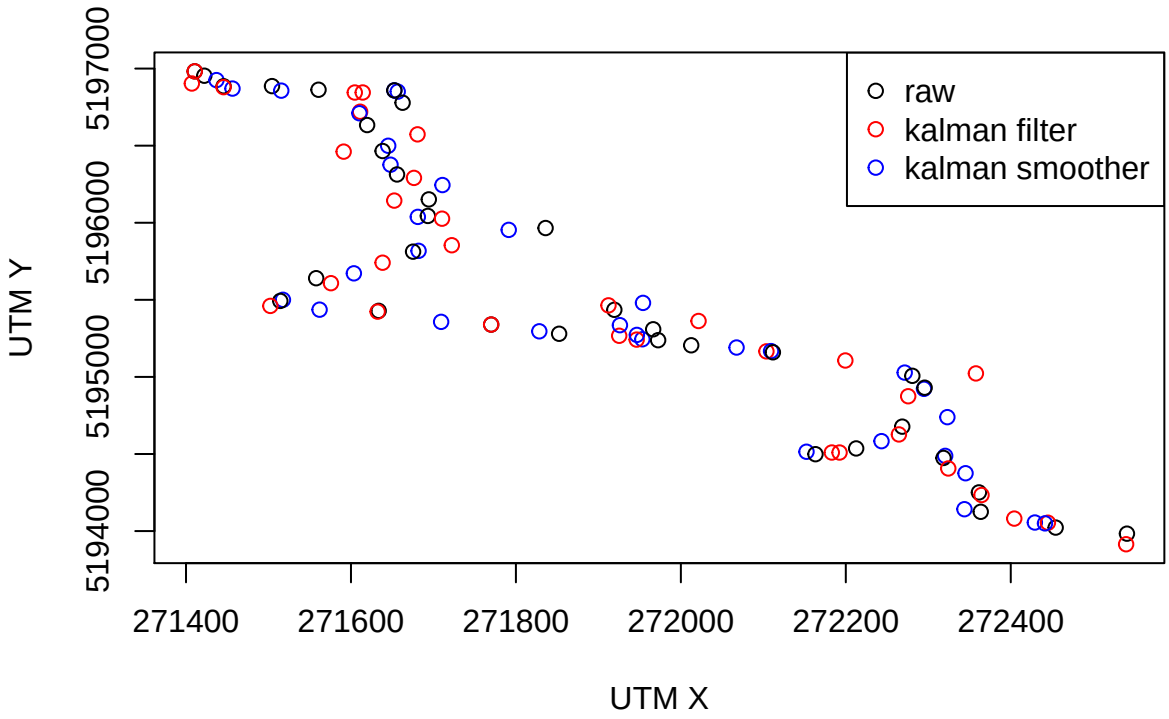
```
## diff
##         8        -2        16        17        -3         9
## 0.2591432 0.2410310 0.1428074 0.1236503 0.1170324 0.1163358
```

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




```
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)
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, dlm_spec)
dlm_smooth_mod <-dlmSmooth(dlm_filter_mod)
```

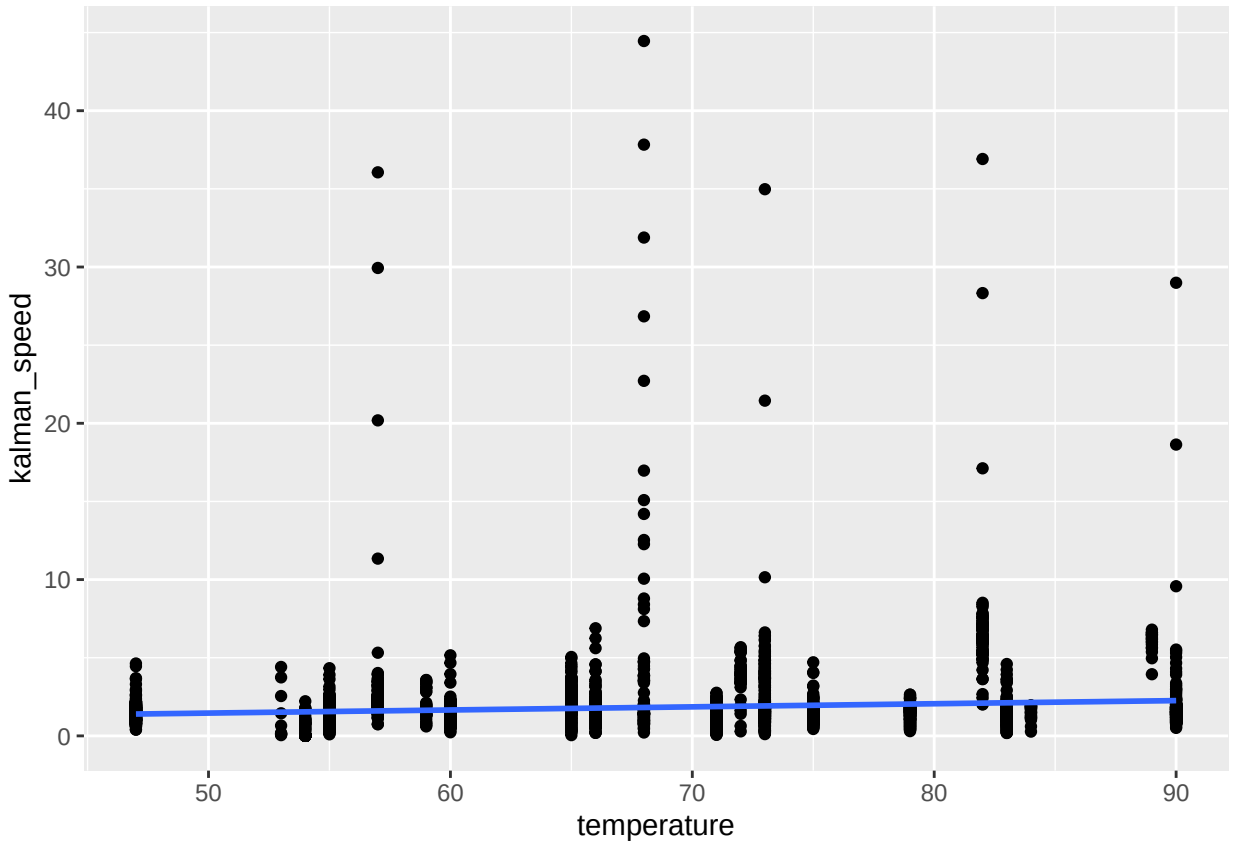
```
plot(cbind(coordinates[1:100, ], dlm_filter_mod$m[2:101, 1:2], dlm_smooth_mod$s[2:101,1:2]),
     type='p', col =c("black", "red", "blue"), xlab="UTM X", ylab="UTM Y")
legend("topright", col =c("black", "red", "blue"),pch = 1,
       legend =c("raw", "kalman filter","kalman smoother"))
```




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

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

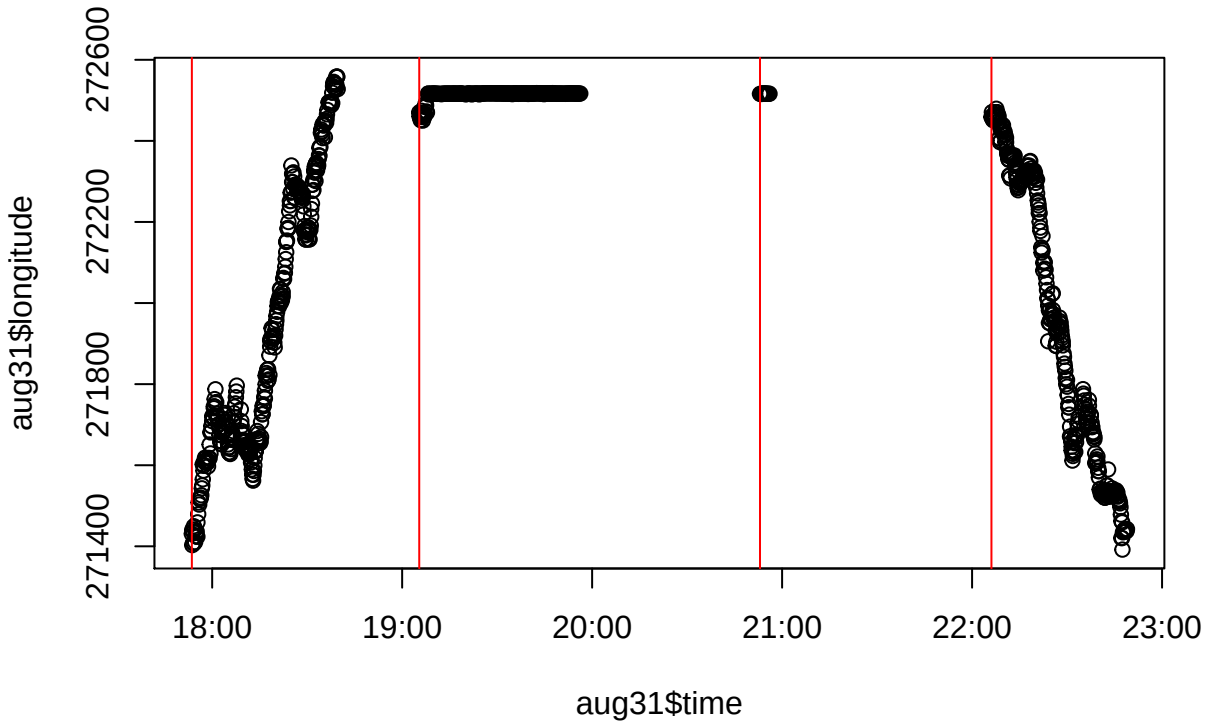
```
## `geom_smooth()` using formula 'y ~ x'
## Warning: Removed 955 rows containing non-finite values (stat_smooth).
## Warning: Removed 955 rows containing missing values (geom_point).
```



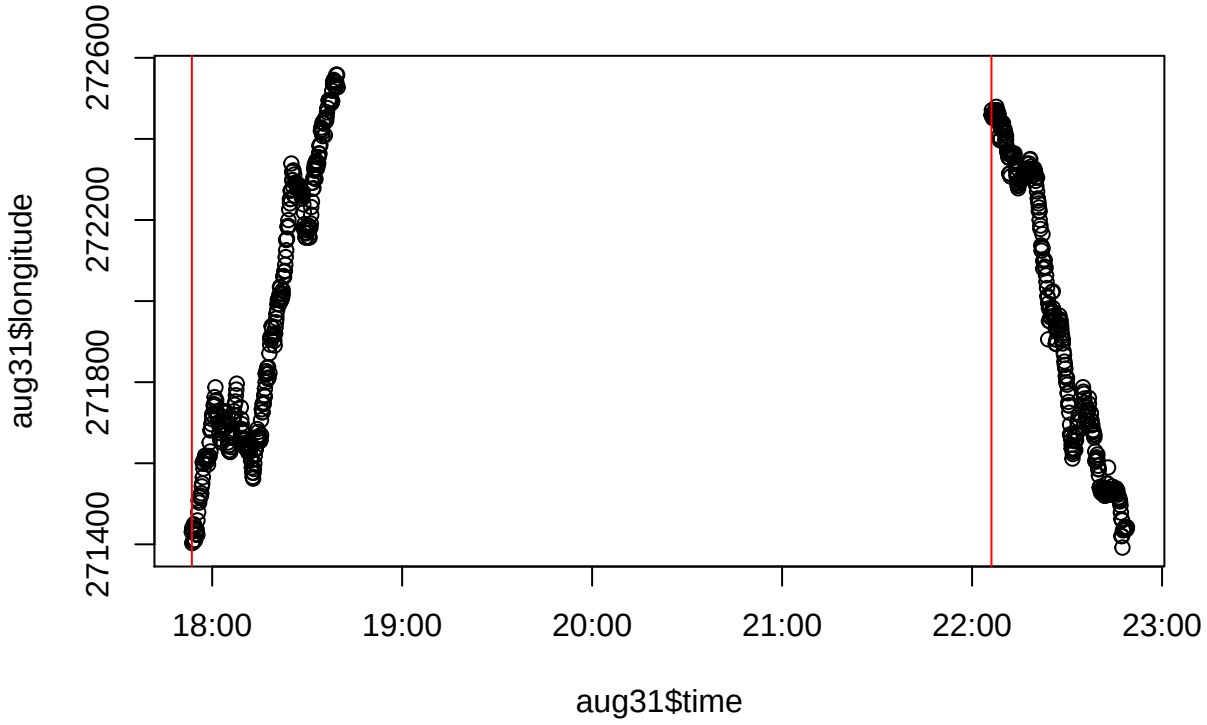

```
summary(lm(data=time,kalman_speed~temperature))
```

```
##
## Call:
## lm(formula = kalman_speed ~ temperature, data = time)
##
## Residuals:
##    Min     1Q Median     3Q    Max
## -1.932 -0.698 -0.316  0.163 42.642
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 0.460990   0.194480   2.370   0.0178 *
## temperature 0.019945   0.002834   7.039 2.27e-12 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 2.091 on 4035 degrees of freedom
##   (955 observations deleted due to missingness)
## Multiple R-squared:  0.01213,    Adjusted R-squared:  0.01188
## F-statistic: 49.54 on 1 and 4035 DF,  p-value: 2.272e-12
```

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

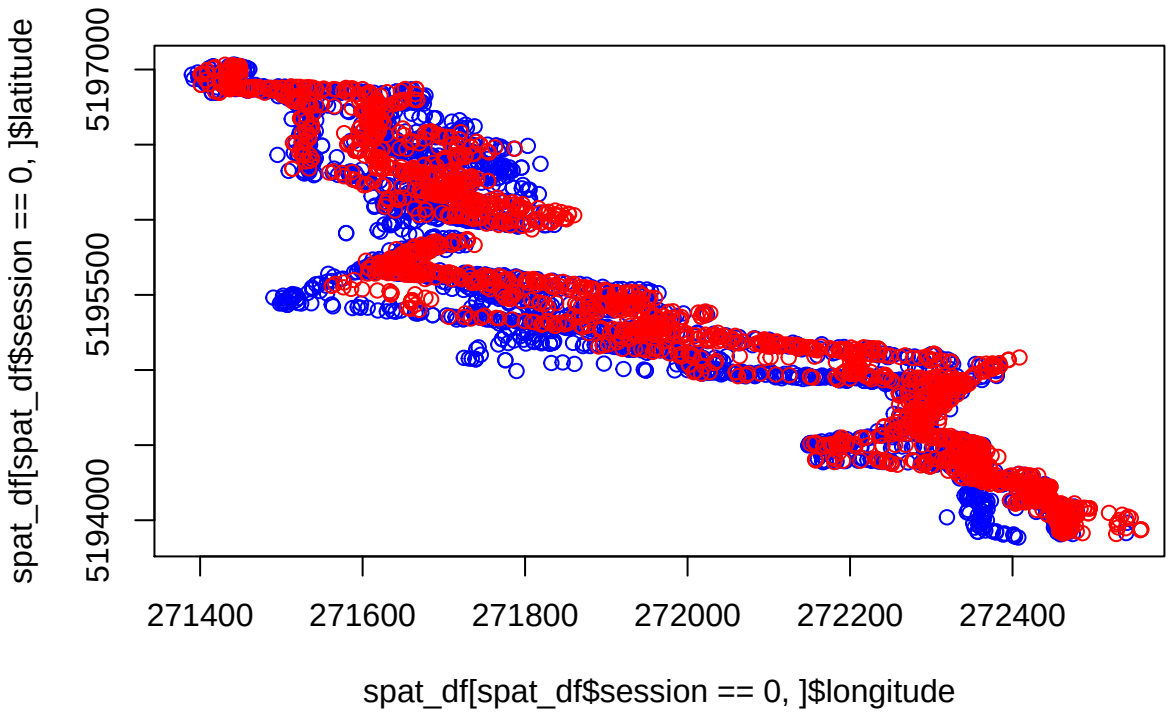



```
df <- df[-c(4487:4663),]
spat_df <- spat_df[-c(4487:4663),]
aug31 <- spat_df[spat_df$time>"2020-08-31 00:00:00 UTC",]
plot(aug31$time,aug31$longitude)
abline(v=aug31$time[aug31$diff>120],col="red")
```




```
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[2756:3118] <- 0
spat_df$session[3119:3467] <- 1
spat_df$session[3468:3734] <- 0
spat_df$session[3735:3795] <- 1
spat_df$session[3796:4125] <- 0
spat_df$session[4126:4486] <- 1
spat_df$session[4487:4815] <- 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')
```

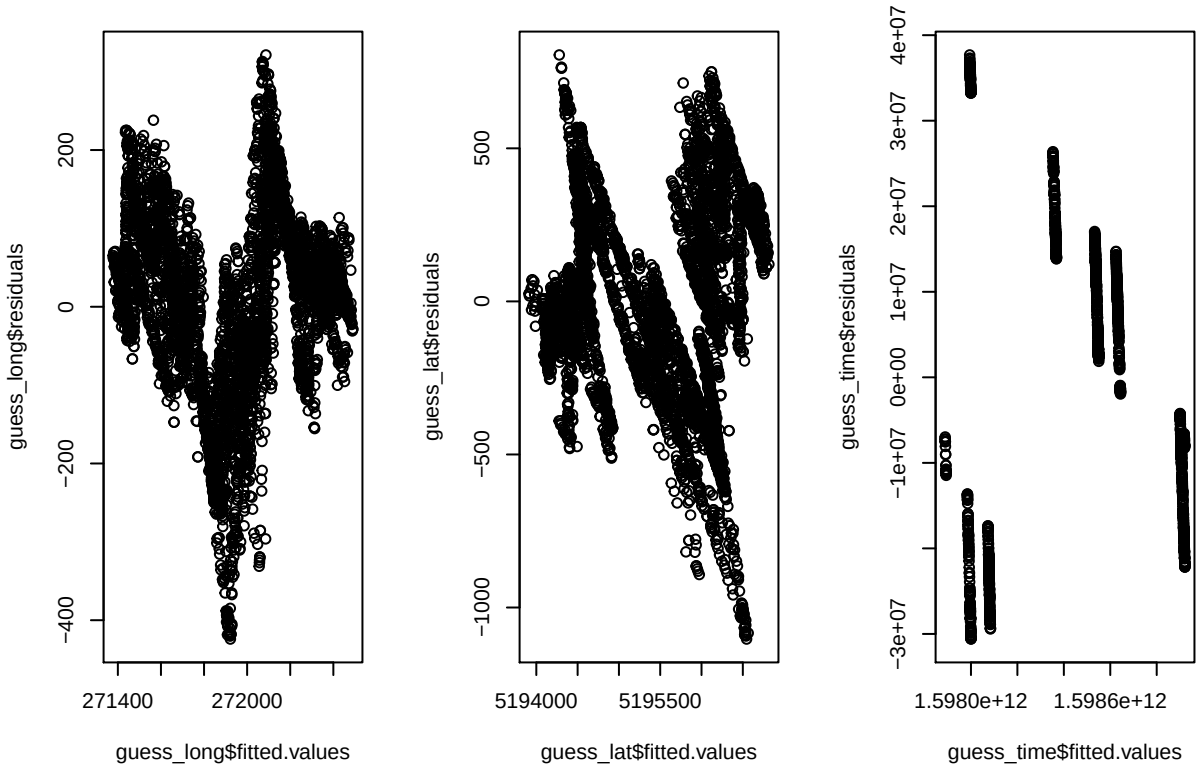



```
# guess_long1 <- lm(data=spat_df,longitude~(latitude+time_long+altitude+bearing+speed)^2)
# #step(guess_long,direction='backward')
# guess_long <- lm(data=spat_df,longitude~sin(latitude+altitude+speed+time_long+latitude*altitude))
# #summary(guess_long)
# #predict(guess_long,se.fit=TRUE)
#
# guess_lat1 <- lm(data=spat_df,latitude~(longitude+time+altitude+bearing+speed)^2)
# #step(guess_lat,direction='backward')
# guess_lat <- lm(data=spat_df,latitude~sin(longitude+altitude+speed+bearing+time_long+longitude*speed)
# #summary(guess_lat)
```

```
# #predict(guess_lat,se.fit=TRUE)
#
# guess_time1 <- lm(data=spat_df,time_long~(longitude+latitude+altitude+bearing+speed)^2)
# #step(guess_time,direction='backward')
# guess_time <- lm(data=spat_df,time_long~longitude+altitude+bearing+speed+day+longitude*altitude+altit
# #summary(guess_time)
# #predict(guess_time,se.fit=TRUE)
#
# guess_day <- lm(data=spat_df,day~longitude+altitude+bearing+speed+latitude+time_long)
```

```
df$longitude <- spat_df@coords[,1]
df$latitude <- spat_df@coords[,2]
df$session <- spat_df$session

guess_long <- lm(data=df,longitude~latitude+time+day+session)
#step(guess_long)

guess_lat <- lm(data=df,latitude~longitude+time+day+session)
#step(guess_lat)
guess_lat <- lm(data=df,latitude~longitude+time)

guess_time <- lm(data=df,time_long~longitude+day+latitude+session)
#step(guess_time)
guess_time <- lm(data=df[which(guess_time$residuals < (-4000000)),],time_long~longitude+day)

#guess_day <- lm(data=spat_df,day~longitude+latitude+time+session)
#step(guess_day)
#guess_day <- lm(data=spat_df,day~longitude+latitude+time)
#guess_day <- lm(data=spat_df[which(guess_day$residuals>0.5),],day~longitude+latitude+time)
```

```
par(mfrow=c(1,3))
plot(guess_long$fitted.values,guess_long$residuals)
plot(guess_lat$fitted.values,guess_lat$residuals)
plot(guess_time$fitted.values,guess_time$residuals)
```




```
#plot(guess_day$fitted.values,guess_day$residuals)
```

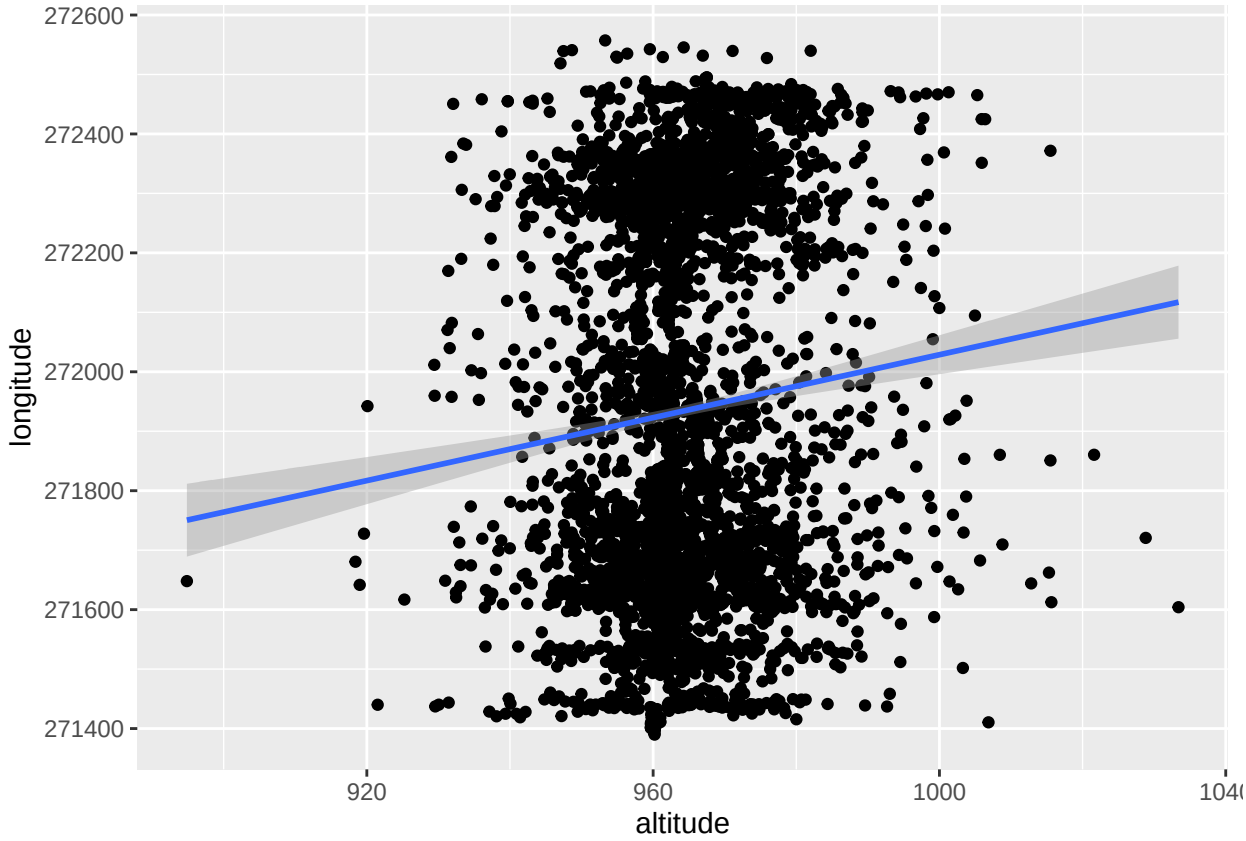
```
# errors <- bind_cols(long=as.vector(guess_long$residuals),lat=as.vector(guess_lat$residuals))
#
# errors$comb <- abs(errors$long)+abs(errors$lat)
#
# which(errors$comb==min(errors$comb))
#
# df_test[4819,]
# errors[4819,]
#
# # Bomb 1: Monday, at 22:25:30 at long=-113.9922; lat=46.87254.
#
# which(abs(errors$long)==min(abs(errors$long)))
# which(abs(errors$lat)==min(abs(errors$lat)))
#
# df_test[3599,]
# df_test[1773,]
# errors[3599,]
# errors[1773,]
#
# ggplot(df_test[2914,]) +
#   geom_point(aes(x=longitude,y=latitude)) +
#   geom_point(aes(x=pred_long,y=pred_lat),col='red')
#
# ggplot(head(df_test),aes(x=latitude,y=pred_lat)) +
#   geom_point() +
#   geom_smooth() +
#   geom_ribbon(aes(ymin=lat_lwr,ymax=lat_upr), alpha=0.3,col='red')
```

```
par(mfrow=c(1,2))
ggplot(df) + geom_point(aes(x=altitude,y=longitude)) + geom_smooth(aes(x=altitude,y=longitude),method="
```

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

```
## Warning: Removed 40 rows containing non-finite values (stat_smooth).
```

```
## Warning: Removed 40 rows containing missing values (geom_point).
```

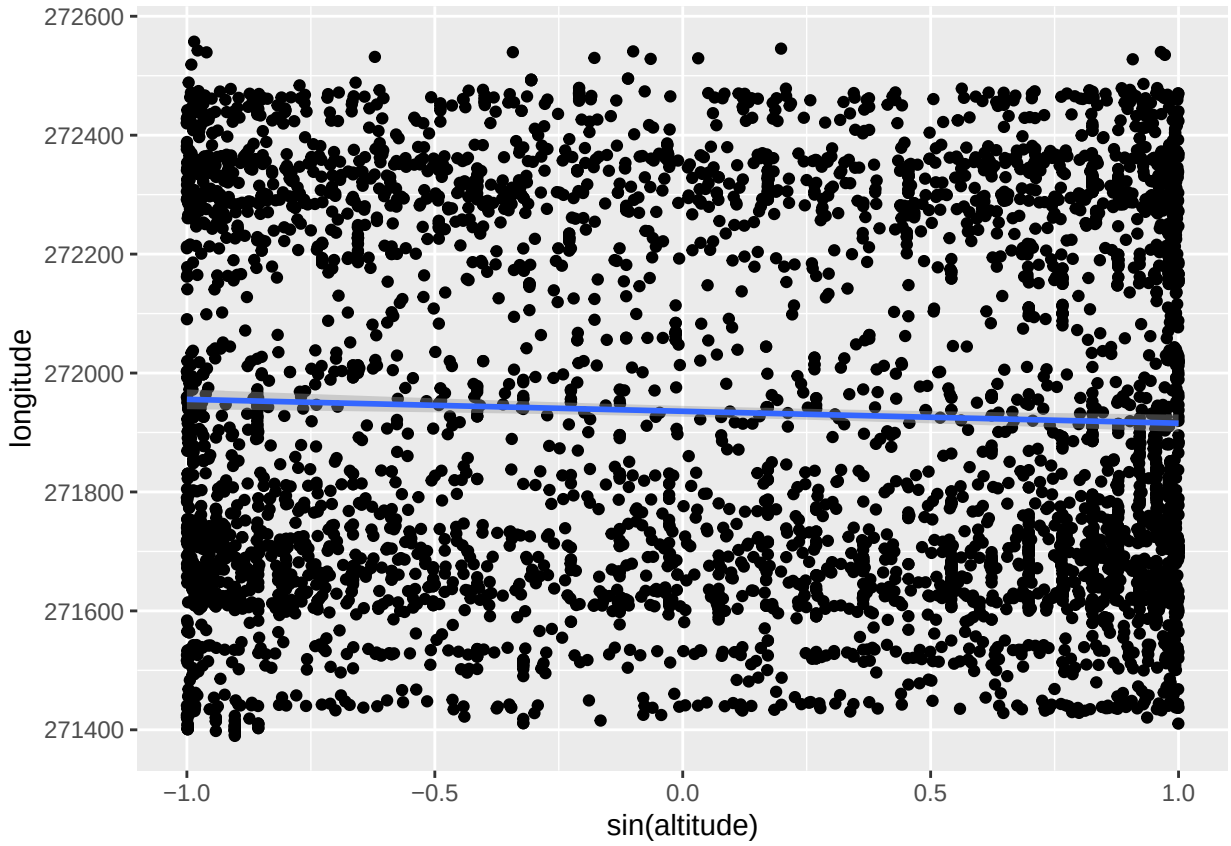



```
ggplot(df) + geom_point(aes(x=sin(altitude),y=longitude)) + geom_smooth(aes(x=sin(altitude),y=longitude
```

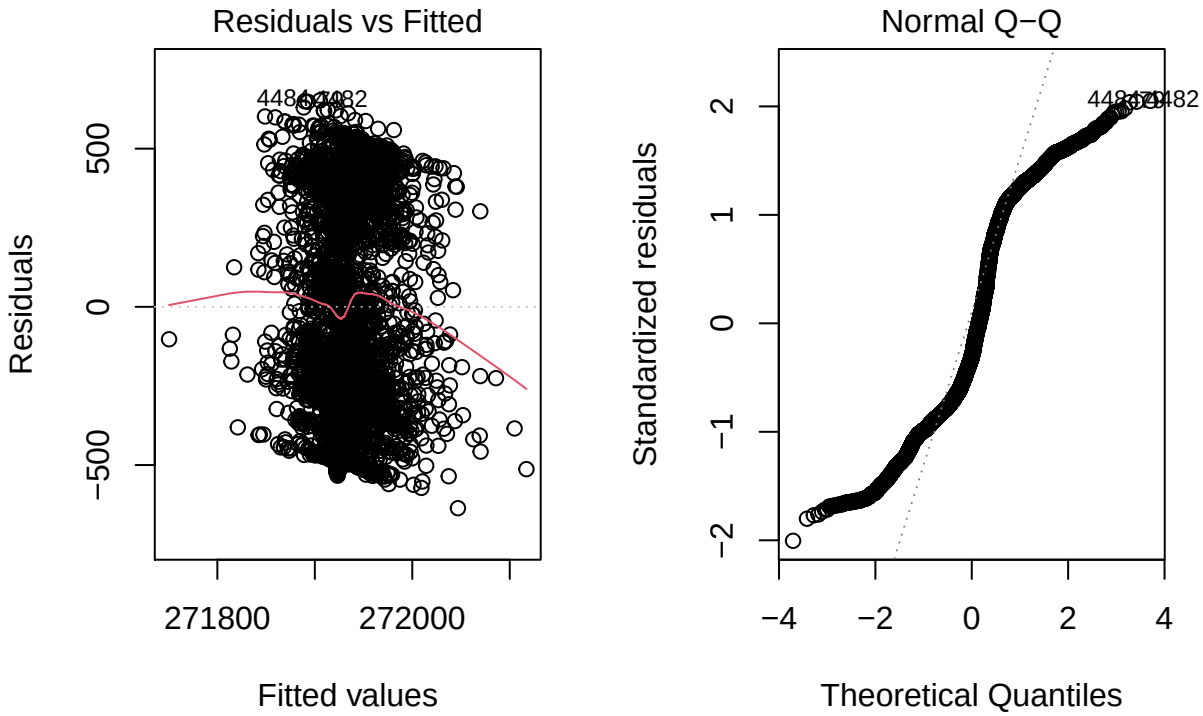
```
## `geom_smooth()` using formula 'y ~ x'
```

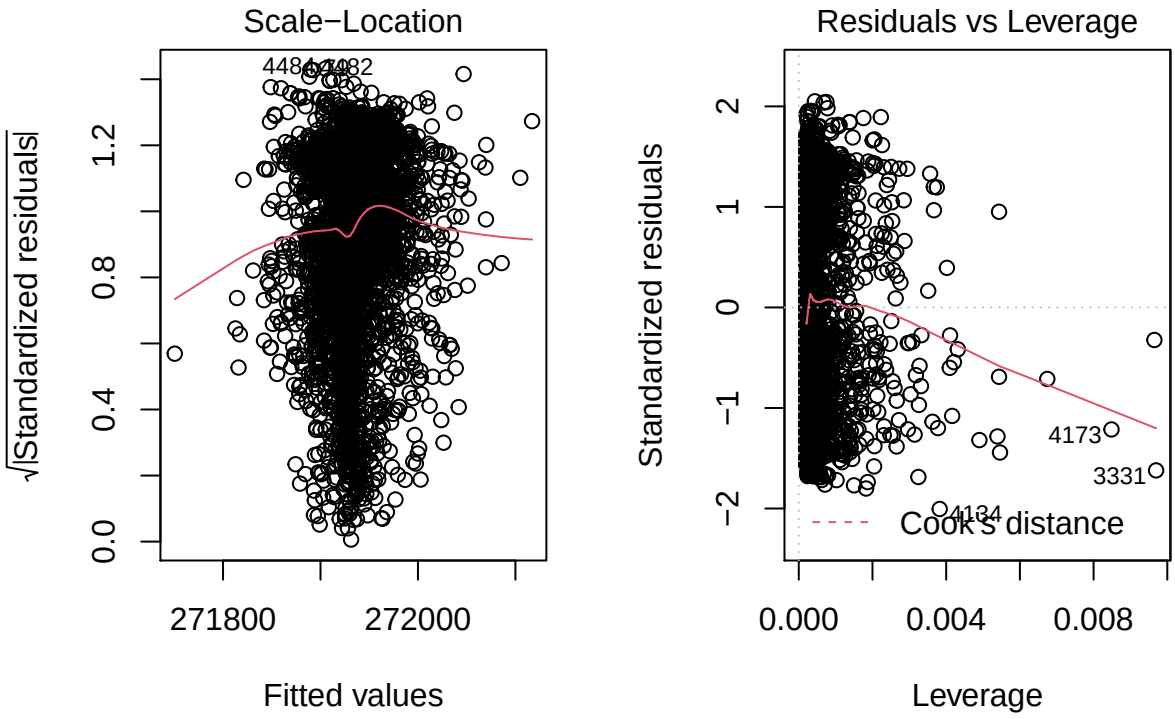
```
## Warning: Removed 40 rows containing non-finite values (stat_smooth).
```

```
## Warning: Removed 40 rows containing missing values (geom_point).
```




```
plot(lm(data=df,longitude~altitude))
```






```
# predict_long <- predict(guess_long,interval='confidence')
# predict_lat <- predict(guess_lat,interval='confidence')
# predict_time <- predict(guess_time,interval='confidence')
#
# full_test <- tibble(pred_long=predict_long[,1],long_lwr=predict_long[,2],long_upr=predict_long[,3],
#                     pred_lat=predict_lat[,1],lat_lwr=predict_lat[,2],lat_upr=predict_lat[,3],
#                     pred_time=predict_time[,1],time_lwr=predict_time[,2],time_upr=predict_time[,3])
#
# df_test <- cbind(spat_df@data,full_test) %>% tibble()
#
# ggplot(df_test,aes(x=longitude,y=pred_long)) +
#   geom_point() +
#   geom_smooth() +
#   geom_ribbon(aes(ymin=long_lwr,ymax=long_upr), alpha=0.3,col='red')
#
# ggplot(head(df_test),aes(x=longitude,y=pred_long)) +
#   geom_point() +
#   geom_smooth() +
#   geom_ribbon(aes(ymin=long_lwr,ymax=long_upr), alpha=0.3,col='red')
#
# ggplot(df_test,aes(x=latitude,y=pred_lat)) +
#   geom_point() +
#   geom_smooth() +
#   geom_ribbon(aes(ymin=lat_lwr,ymax=lat_upr), alpha=0.3,col='red')
#
# ggplot(head(df_test),aes(x=latitude,y=pred_lat)) +
#   geom_point() +
#   geom_smooth() +
#   geom_ribbon(aes(ymin=lat_lwr,ymax=lat_upr), alpha=0.3,col='red')
#
# ggplot(df_test,aes(x=time_long,y=pred_time)) +
```

```
#   geom_point() +
#   geom_ribbon(aes(ymin=time_lwr,ymax=time_upr), alpha=0.3,col='red')
#
# init <- tibble(longitude=spat_df$longitude[!duplicated(spat_df$day)],latitude=spat_df$latitude[!dupli
#                time_long=spat_df$time_long[!duplicated(spat_df$day)],
#                altitude=spat_df$altitude[!duplicated(spat_df$day)],
#                bearing=spat_df$bearing[!duplicated(spat_df$day)],speed=spat_df$speed[!duplicated(spat
#
# predict(newdata=init,object=guess_long)
```

```
training_indices <- sample(1:nrow(df),0.8*nrow(df))
training <- df[training_indices,]
testing <- df[-training_indices,]
```

```
make_pred <- function(df=spat_df) {

}
```