Lab 1

Carmen Canedo

May 24, 2020

About

For this lab, I used the GPS tracking app on my iPhone, myTracker, to trace myself walking a straight path outside of my house in Nashville, TN. I tried to maintain a steady pace as I walked up and down the street. The GPX and .csv file are included alongside the submission of my lab.

Below you will find an analysis of my path that was created using R statistical software. My code is available below, but my step-by-step process can be found on *my GitHub page*.

Necessary packages

In order to run the code below, the following packages must be installed and loaded on your computer.

```
library(mdsr)
library(XML)
library(OpenStreetMap)
library(lubridate)
library(ggmap)
library(raster)
library(sp)
```

Getting the data ready

Before conducting an analysis on my data, I found that it was necessary to narrow down the number of columns and rename some columns for clarity.

Loading in data from .csv

Saving .csv file into a dataframe that can be easily manipulated.

```
walking <- read.csv("lab-1.csv", header = TRUE)
```

Cleaning data

Getting rid of unnecessary columns

```
walking <- walking %>%
  dplyr::select(-type, -desc, -name)
```

Making column names simpler

```
walking <- walking %>%
  rename(altitude = altitude..ft.) %>%
  rename(speed = speed..mph.) %>%
  rename(distance_mi = distance..mi.) %>%
  rename(distance_int_ft = distance_interval..ft.)
```

Summary stats calculations

Summary statistics provide us with key information on the center and spread of our data; below, I calculate the minimum, first quartile, median, third quartile, max, mean, and standard deviation of all the variables in the data set.

```
sum_latitude <- favstats( ~ latitude, data = walking)

sum_longitude <- favstats( ~ longitude, data = walking)

sum_altitude <- favstats( ~ altitude, data = walking)

sum_speed <- favstats( ~ speed, data = walking)

sum_distance_mi <- favstats( ~ distance_mi, data = walking)

sum_dist_int_ft <- favstats( ~ distance_int_ft, data = walking)
```

Results

```
sum_latitude
```

```
##       min       Q1  median       Q3      max    mean           sd   n missing
##  36.18001 36.18029 36.1806 36.18091 36.18117 36.1806 0.0003514294 221       0
```

```
sum_longitude
```

```
##        min        Q1   median        Q3       max     mean           sd   n
##  -86.74297 -86.74294 -86.7429 -86.74286 -86.74278 -86.7429 4.645446e-05 221
##  missing
##        0
```

```
sum_altitude
```

```
##    min    Q1 median    Q3   max     mean       sd   n missing
##  500.8 505.8  511.2 512.4 516.1 509.3181 4.041696 221       0
```

```
sum_speed
```

```
##  min    Q1 median    Q3 max     mean       sd   n missing
##    0 2.275    2.7 3.325  10 2.839545 1.226507 220       1
```

```
sum_distance_mi
```

```
##  min    Q1 median    Q3   max       mean         sd   n missing
##    0 0.047   0.09 0.137 0.181 0.09093213 0.05260591 221       0
```

```
sum_dist_int_ft
```

```
##  min   Q1 median   Q3   max    mean       sd   n missing
##    0 3.14   4.01 5.29 14.58 4.32362 1.937173 221       0
```

Analysis

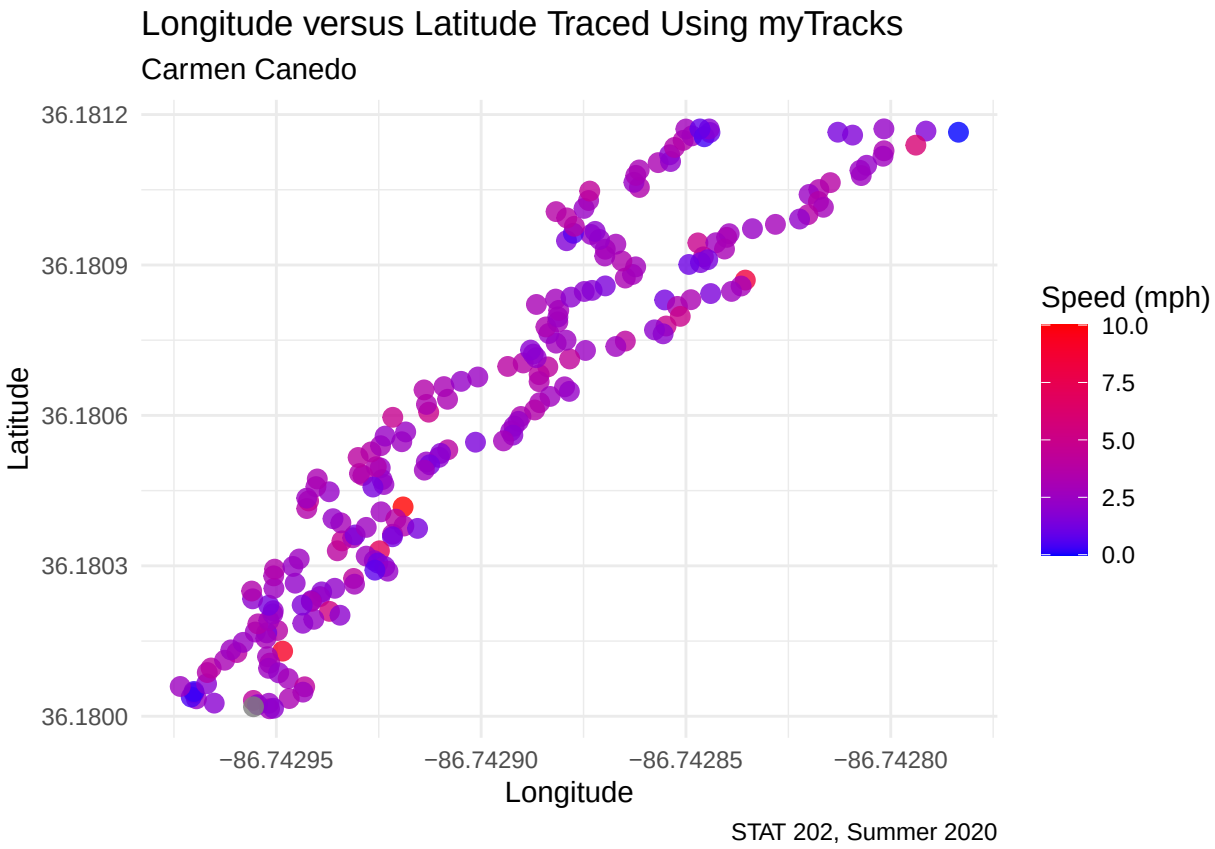
- The standard deviation is larger for latitude than it is for longitude.
- This tells us that the latitude moves farther from the mean than the longitude does. Thus, we can expect the path to go north to south.

Creating latitude v. longitude scatter plot

Next, we are going to create a scatter plot of the longitude and latitude points that are colored based on my walking speed in miles per hour (mph). The speed uses a blue to red scale where the slower I walk, the more blue the point appears, and as my speed increases, the point becomes more red. Because I aimed to maintain a steady pace during my walk, most of the points are purple.

```
lat_v_long <- walking %>%
  ggplot(aes(x = longitude, y = latitude)) +
  geom_point(alpha = 0.8, aes(color = speed), size = 3) +
  scale_color_gradient(low = "blue", high = "red") +
  theme_minimal() +
  labs(title = "Longitude versus Latitude Traced Using myTracks",
       subtitle = "Carmen Canedo",
       caption = "STAT 202, Summer 2020",
       x = "Longitude",
       y = "Latitude",
       color = "Speed (mph)")

lat_v_long
```



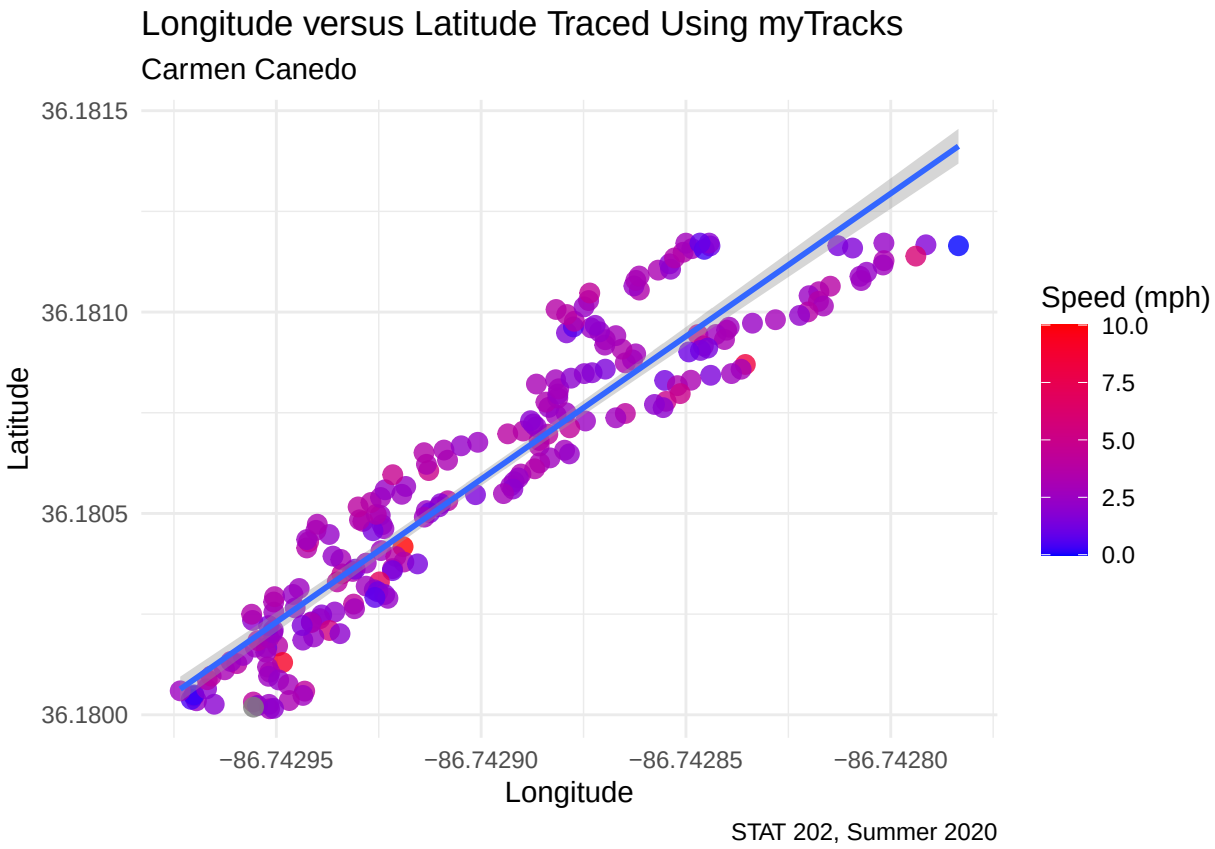
Adding Line of Best Fit

In order to approximate the location of the sidewalk, we are going to add a line of best fit.

```
lat_v_long <- lat_v_long +
  geom_smooth(method = "lm")

lat_v_long
```

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



Simple Linear Regression Results

Now, we will examine the linear regression more closely and determine the formula for the line of best fit.

```
# Calculating model
model <- lm(latitude ~ longitude, data = walking)

# Finding y-intercept and slope
coef(model)
```

```
## (Intercept)   longitude
##  651.652871    7.095362
```

```
# Finding correlation coefficient
cor(walking$longitude, walking$latitude)
```

```
## [1] 0.9379158
```

`coef()` returns the y-intercept and slope values that we can use to write out the formula for our linear regression below.

Formula for line of best fit: $latitude = 651.653 + 7.0954(longitude)$

`cor()` returns the correlation coefficient that will help us gauge whether our model actually represents the sidewalk well.

Analysis

- The regression line is a good tool to estimate the path I traveled because it is able to predict the y-value given the change in the x-value.
- The correlation coefficient ranges from 0 to 1, and since our correlation value is 0.938, we can ascertain that our trend line accurately portrays the location of the sidewalk.

Mapping the route

The exercises I referenced to create my own map can be found by clicking *here*.

Getting the data

In order to ensure that all the values work when mapped, this equation places the vectors correctly.

```
# Function to shift vectors
shift_vec <- function(vector, shift) {
  if (length(vec) <= abs(shift)) {
    rep(NA, length(vec))
  } else {
    if (shift >= 0) {
      c(rep(NA, shift), vec[1:(length(vec) - shift)])
    } else {
      c(vec[(abs(shift) + 1):length(vec)])
    }
  }
}
```

Reading in GPX file

Next we read in the GPX file itself instead of using the .csv file from above.

```
# Limits to 10 digits
options(digits = 10)

# Parsing the GPX file
parsed_file <- htmlTreeParse(file = "lab-1-raw-data.gpx",
                             error = function(...) {},
                             useInternalNodes = TRUE)

# Get all times and coordinates via the respective xpath
times <- xpathSApply(parsed_file, path = "//trkpt/time", xmlValue)
coords <- xpathSApply(parsed_file, path = "//trkpt", xmlAttrs)

# Extract latitude and longitude from the coordinates
lats <- as.numeric(coords["lat",])
lons <- as.numeric(coords["lon",])
```

Putting values into a dataframe

This allows us to have all of individual data points from the GPX file in one place, ready to be placed onto a map.

```
geodf <- data.frame(lat = lats, lon = lons, time = times)
```

Querying map background

I am using my Google API to access the static map used below. It is centered on the street where I walked, and I zoomed in so that my path will be visible.

```
street <- get_map(location = "409 N 15th St., Lockeland Springs, Nashville, Tennessee",
                  zoom = 19,
                  maptype = "roadmap")
```

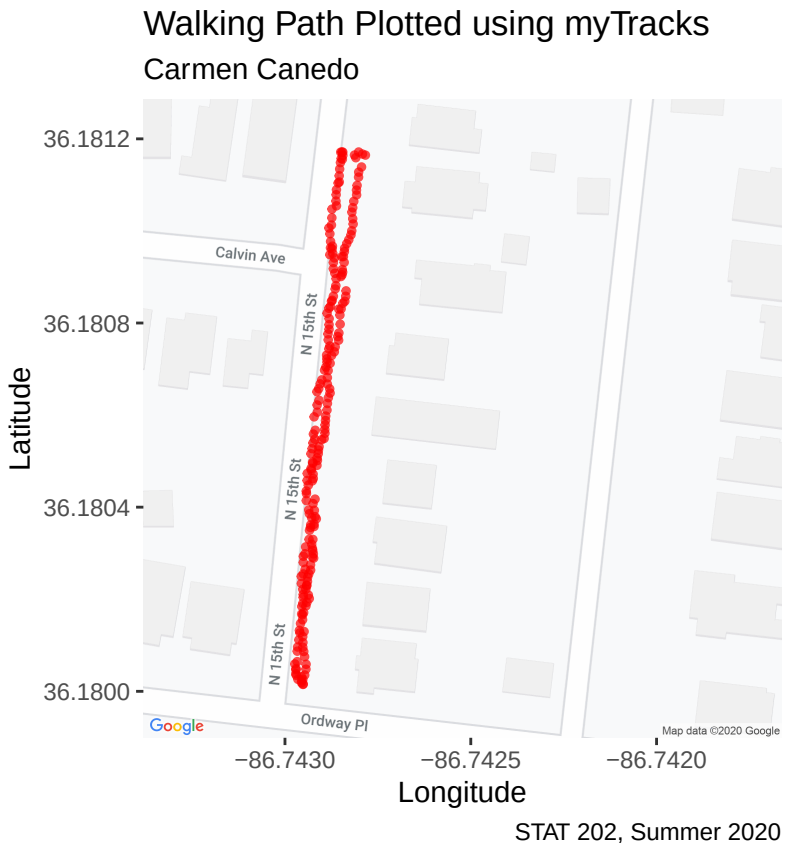
Finished product

Finally, we can put all the data together and plot the points onto the map using the `ggmap` package.

```
# Plotting points
path <- ggmap(street) +
  geom_point(data = geodf,
             aes(x = lon, y = lat),
             size = 1,
             alpha = 0.7,
             color = "red")

# Adding details
path <- path +
  labs(x = "Longitude",
       y = "Latitude",
       title = "Walking Path Plotted using myTracks",
       subtitle = "Carmen Canedo",
       caption = "STAT 202, Summer 2020")

# Printing map
path
```



Summary

This lab allowed me to apply statistical skills in a real life setting. I enjoy using R and Python, and though I have experience with them in other settings, it was really informative to use them to make statistical calculations. The biggest thing I learned was how to produce static maps using my very own Google API!