project2

David Moran

2024-11-14

```r
files <- list.files(path = "/Users/david_m123/Documents/gps", pattern = "*.geojson", full.names = TRUE)


all_features <- list()


for (file in files) {
  gps_data <- fromJSON(file, flatten = TRUE)
  features <- gps_data$features
  all_features <- append(all_features, list(features))
}

combined_features <- do.call(rbind,lapply(all_features,as.data.frame))


coordinates <- combined_features$geometry.coordinates
timestamps <- combined_features$properties.time
altitude <- combined_features$properties.altitude
accuracy <- combined_features$properties.accuracy
speed <- combined_features$properties.speed
bearings <- combined_features$properties.bearing



gps_df <- data.frame(
  longitude = sapply(coordinates, function(x) x[1]),
  latitude = sapply(coordinates, function(x) x[2]),
  timestamp = as.POSIXct(timestamps, format = "%Y-%m-%dT%H:%M:%OSZ", tz = "UTC"),
  altitude = altitude,
  speed = speed,
  bearings = bearings,
  stringsAsFactors = FALSE
)


head(gps_df)
```

```
##   longitude latitude           timestamp altitude speed bearings
## 1 -114.0003 46.88678 2020-08-18 17:50:42 971.2080  1.06    140.2
## 2 -114.0005 46.88632 2020-08-18 17:50:40 959.7000    NA       NA
## 3 -113.9999 46.88604 2020-08-18 17:51:49 972.2537  1.81     99.5
```

```
## 4 -113.9992 46.88620 2020-08-18 17:52:24 946.5977  3.41     93.1
## 5 -113.9978 46.88587 2020-08-18 17:53:12 979.1237  0.00       NA
## 6 -113.9972 46.88593 2020-08-18 17:53:11 960.9000    NA       NA
```

```r
gps_df_unique <- gps_df[!duplicated(gps_df[, c("longitude", "latitude")]), ]
num_removed <- nrow(gps_df) - nrow(gps_df_unique)
num_removed
```

```
## [1] 514
```

```r
head(gps_df_unique)
```

```
##   longitude latitude           timestamp altitude speed bearings
## 1 -114.0003 46.88678 2020-08-18 17:50:42 971.2080  1.06    140.2
## 2 -114.0005 46.88632 2020-08-18 17:50:40 959.7000    NA       NA
## 3 -113.9999 46.88604 2020-08-18 17:51:49 972.2537  1.81     99.5
## 4 -113.9992 46.88620 2020-08-18 17:52:24 946.5977  3.41     93.1
## 5 -113.9978 46.88587 2020-08-18 17:53:12 979.1237  0.00       NA
## 6 -113.9972 46.88593 2020-08-18 17:53:11 960.9000    NA       NA
```

```r
library(sf)


convert_to_utm <- function(gps_df_unique, zone = 12) {

  wgs84 <- st_crs(4326)

  utm_crs <- paste0("+proj=utm +zone=", zone, " +datum=WGS84")


  df_sf <- st_as_sf(gps_df_unique, coords = c("longitude", "latitude"), crs = wgs84)


  df_utm <- st_transform(df_sf, crs = utm_crs)


  coords <- st_coordinates(df_utm)
  gps_df_unique$easting <- coords[, 1]
  gps_df_unique$northing <- coords[, 2]
  return(gps_df_unique)
}


gps_data_utm <- convert_to_utm(gps_df_unique)


head(gps_data_utm)
```

```
##   longitude latitude           timestamp altitude speed bearings  easting
## 1 -114.0003 46.88678 2020-08-18 17:50:42 971.2080  1.06    140.2 271422.2
## 2 -114.0005 46.88632 2020-08-18 17:50:40 959.7000    NA       NA 271407.1
## 3 -113.9999 46.88604 2020-08-18 17:51:49 972.2537  1.81     99.5 271456.3
```

```
## 4 -113.9992 46.88620 2020-08-18 17:52:24 946.5977  3.41     93.1 271504.3
## 5 -113.9978 46.88587 2020-08-18 17:53:12 979.1237  0.00       NA 271614.3
## 6 -113.9972 46.88593 2020-08-18 17:53:11 960.9000    NA       NA 271656.7
##   northing
## 1  5196954
## 2  5196903
## 3  5196870
## 4  5196886
## 5  5196845
## 6  5196850
```

```
utm_data <- gps_data_utm[, c("timestamp", "easting", "northing", "altitude", "speed", "bearings")]


missing_values <- sapply(utm_data, function(x) sum(is.na(x)))


print(missing_values)
```

```
## timestamp   easting  northing  altitude     speed  bearings
##         0         0         0        33      1664      1855
```

```
# Step 1: Data Exploration and Preprocessing

# 1.1 Visualize data to identify typical routes and anomalies


# 1.2 Handle Missing Values
# We will interpolate the speed and bearings columns since these values are continuous and can be estim



utm_data$speed <- zoo::na.approx(utm_data$speed, method = "linear")
utm_data$bearings <- zoo::na.approx(utm_data$bearings, method = "linear")


utm_data_cleaned <- na.omit(utm_data, cols = "altitude")

# 1.3 Mark instances of missing or inconsistent GPS data
# For data consistency, we will create a new column that flags if the data had to be interpolated.
utm_data_cleaned$interpolated <- apply(utm_data_cleaned, 1, function(row) {
  if (is.na(gps_data_utm[row["timestamp"], "speed"]) || is.na(gps_data_utm[row["timestamp"], "bearings"]
    return(1)
  } else {
    return(0)
  }
})

# 1.4 Add Data Augmentation Features
# Adding additional features such as day of the week and time of day for analysis
utm_data_cleaned$timestamp <- as.POSIXct(utm_data_cleaned$timestamp)
utm_data_cleaned$day_of_week <- weekdays(utm_data_cleaned$timestamp)
utm_data_cleaned$time_of_day <- sapply(format(utm_data_cleaned$timestamp, "%H"), function(x) {
```

```r
  if (x >= 5 & x < 12) {
    return("morning")
  } else if (x >= 12 & x < 17) {
    return("afternoon")
  } else if (x >= 17 & x < 21) {
    return("evening")
  } else {
    return("night")
  }
})


head(utm_data_cleaned)
```

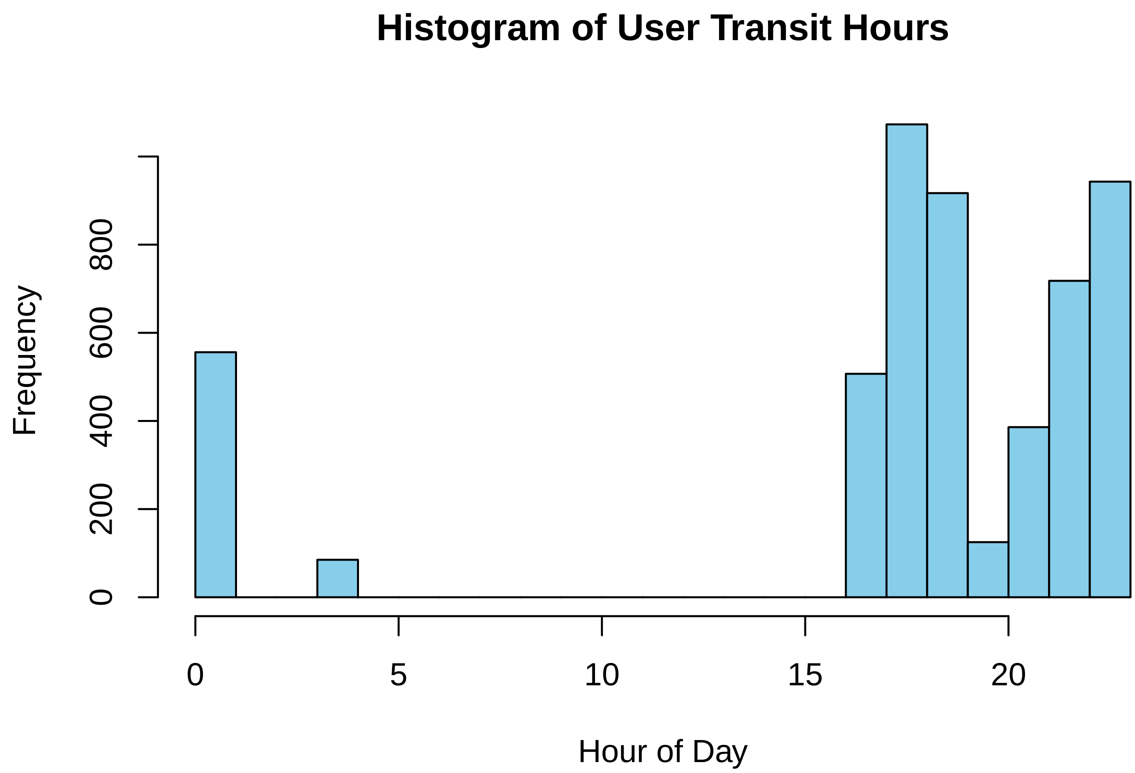
```
##             timestamp  easting northing altitude     speed bearings
## 1 2020-08-18 17:50:42 271422.2  5196954 971.2080 1.0600000  140.200
## 2 2020-08-18 17:50:40 271407.1  5196903 959.7000 1.4350000  119.850
## 3 2020-08-18 17:51:49 271456.3  5196870 972.2537 1.8100000   99.500
## 4 2020-08-18 17:52:24 271504.3  5196886 946.5977 3.4100000   93.100
## 5 2020-08-18 17:53:12 271614.3  5196845 979.1237 0.0000000  105.925
## 6 2020-08-18 17:53:11 271656.7  5196850 960.9000 0.1357143  118.750
##   interpolated day_of_week time_of_day
## 1            1     Tuesday     evening
## 2            1     Tuesday     evening
## 3            1     Tuesday     evening
## 4            1     Tuesday     evening
## 5            1     Tuesday     evening
## 6            1     Tuesday     evening
```

```r
# 2.1 Calculate Basic Features: Speed, Bearing, Acceleration, and Distance

utm_data_cleaned$acceleration <- c(NA, diff(utm_data_cleaned$speed) / as.numeric(diff(utm_data_cleaned$t


utm_data_cleaned$distance <- c(NA, sqrt(diff(utm_data_cleaned$easting)^2 + diff(utm_data_cleaned$northi

# 2.2 Identify Transit Periods

speed_threshold <- 0.5


utm_data_cleaned$in_transit <- ifelse(utm_data_cleaned$speed > speed_threshold, 1, 0)

# 2.3 Time-Based Feature Set
# Identify peak transit hours
# Create a new column for hour of the day
utm_data_cleaned$hour <- as.integer(format(utm_data_cleaned$timestamp, "%H"))


peak_hours <- table(utm_data_cleaned$hour)

# 2.4 Extract Frequent Routes using Clustering
# Clustering the UTM coordinates to find frequently traveled routes
```

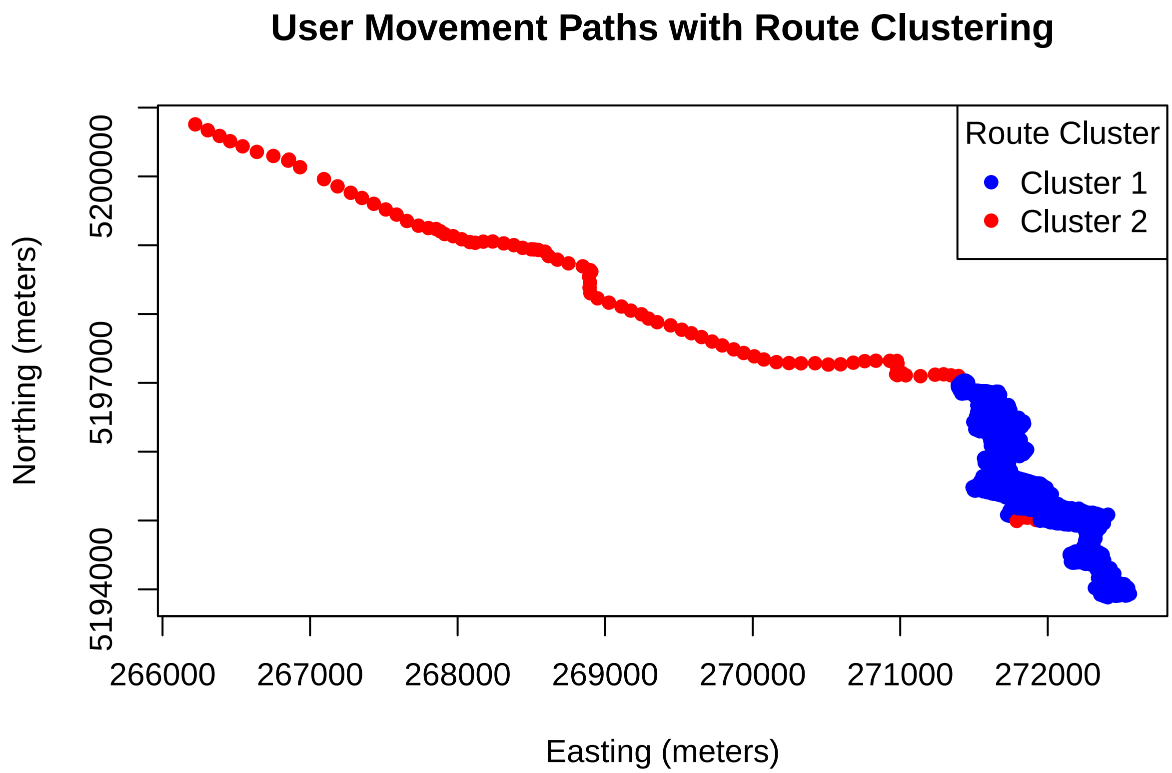
```r
coords <- as.matrix(utm_data_cleaned[, c("easting", "northing")])
db <- dbscan::dbscan(coords, eps = 50, minPts = 5)


utm_data_cleaned$route_cluster <- db$cluster


head(utm_data_cleaned)
```

```
##             timestamp  easting northing altitude     speed bearings
## 1 2020-08-18 17:50:42 271422.2  5196954 971.2080 1.0600000  140.200
## 2 2020-08-18 17:50:40 271407.1  5196903 959.7000 1.4350000  119.850
## 3 2020-08-18 17:51:49 271456.3  5196870 972.2537 1.8100000   99.500
## 4 2020-08-18 17:52:24 271504.3  5196886 946.5977 3.4100000   93.100
## 5 2020-08-18 17:53:12 271614.3  5196845 979.1237 0.0000000  105.925
## 6 2020-08-18 17:53:11 271656.7  5196850 960.9000 0.1357143  118.750
##   interpolated day_of_week time_of_day acceleration  distance in_transit hour
## 1            1     Tuesday     evening           NA        NA          1   17
## 2            1     Tuesday     evening -0.354777699  52.69876          1   17
## 3            1     Tuesday     evening  0.005510087  59.31356          1   17
## 4            1     Tuesday     evening  0.045714286  50.52923          1   17
## 5            1     Tuesday     evening -0.071041667 117.48222          0   17
## 6            1     Tuesday     evening -0.892857434  42.77727          0   17
##   route_cluster
## 1             1
## 2             1
## 3             1
## 4             1
## 5             1
## 6             1
```

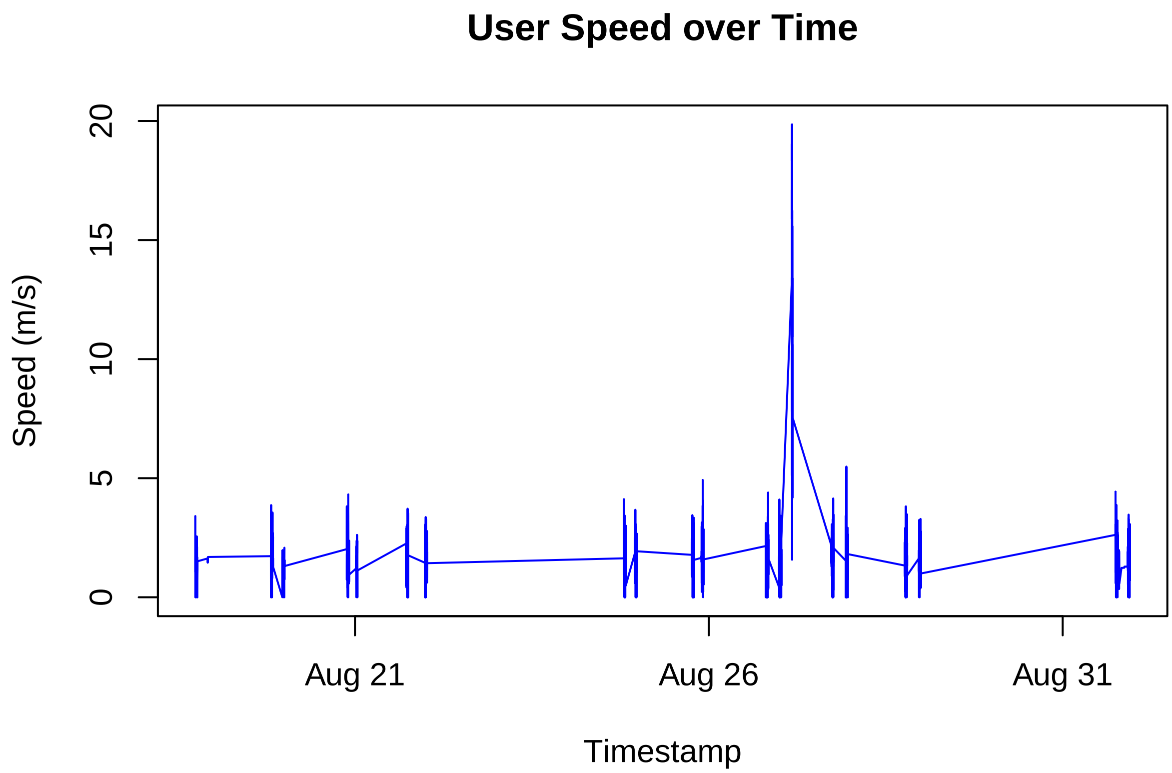
```r
#Visualizations for Feature Analysis

# Visualization 1: Histogram of Peak Transit Hours
hist(utm_data_cleaned$hour, breaks = 24, main = "Histogram of User Transit Hours", xlab = "Hour of Day"
```




```r
# Visualization 2: Scatter Plot of User Movement (Easting vs Northing with Cluster Labels)
plot(utm_data_cleaned$easting, utm_data_cleaned$northing, col = ifelse(utm_data_cleaned$route_cluster ==
legend("topright", legend = c("Cluster 1", "Cluster 2"), col = c("blue", "red"), pch = 16, title = "Rou
```




```
# Visualization 3: Line Plot of Speed over Time
plot(utm_data_cleaned$timestamp, utm_data_cleaned$speed, type = "l", col = "blue", xlab = "Timestamp", y
```




```r
# Analyzing Identified Route Clusters


cluster_counts <- table(utm_data_cleaned$route_cluster)
print("Cluster Counts (Number of points per cluster):")
```

```
## [1] "Cluster Counts (Number of points per cluster):"
```

```r
print(cluster_counts)
```

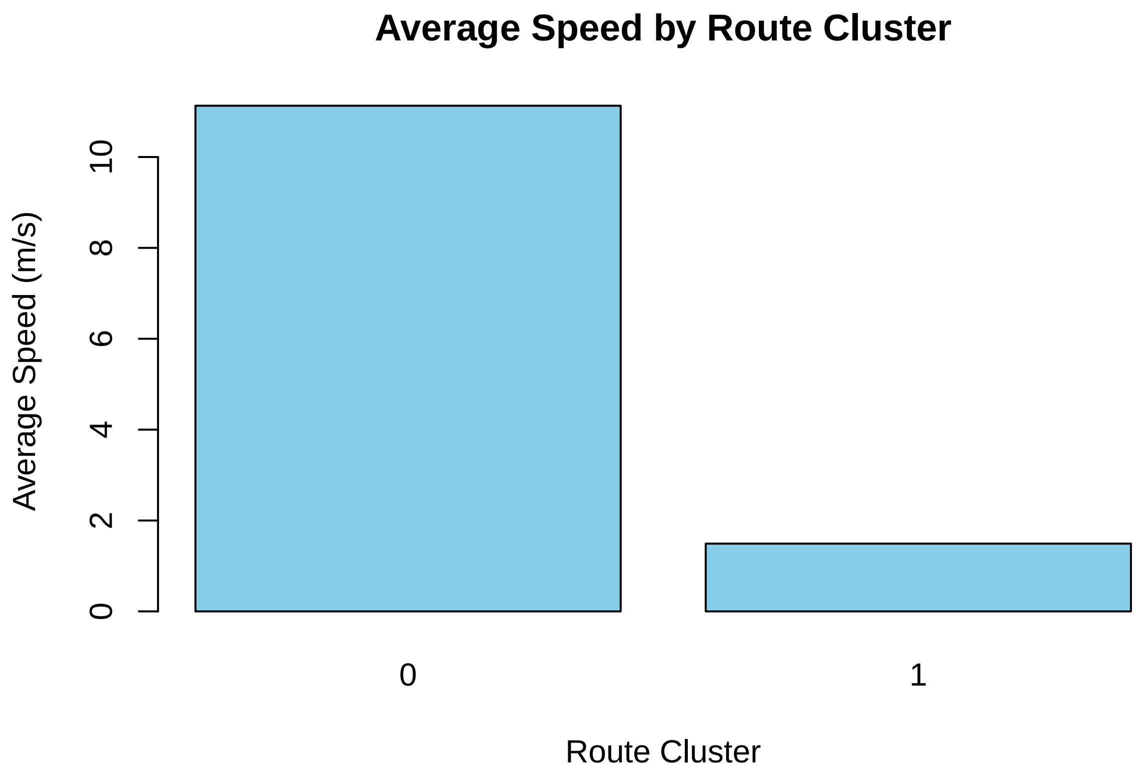
```
##
##    0    1
##   91 5219
```

```r
# Calculate basic statistics for each route cluster (e.g., average speed, average distance traveled wit
route_cluster_analysis <- aggregate(cbind(speed, distance, acceleration) ~ route_cluster, data = utm_da


route_cluster_analysis <- aggregate(cbind(speed, distance, acceleration) ~ route_cluster, data = utm_da


route_cluster_analysis <- do.call(data.frame, route_cluster_analysis)
```

```r
colnames(route_cluster_analysis) <- c("route_cluster", "avg_speed", "speed_std_dev", "avg_distance", "di

print(route_cluster_analysis)
```

```
##   route_cluster avg_speed speed_std_dev avg_distance distance_std_dev
## 1             0 11.128702     4.5494347    153.81471        664.93758
## 2             1  1.491226     0.6265929     25.93676         40.20385
##   avg_acceleration acceleration_std_dev
## 1      0.001975239            0.3699626
## 2     -0.025373930            1.6308716
```

```r
# Visualization: Average Speed by Route Cluster
barplot(route_cluster_analysis$avg_speed, names.arg = route_cluster_analysis$route_cluster, col = "skybl
```

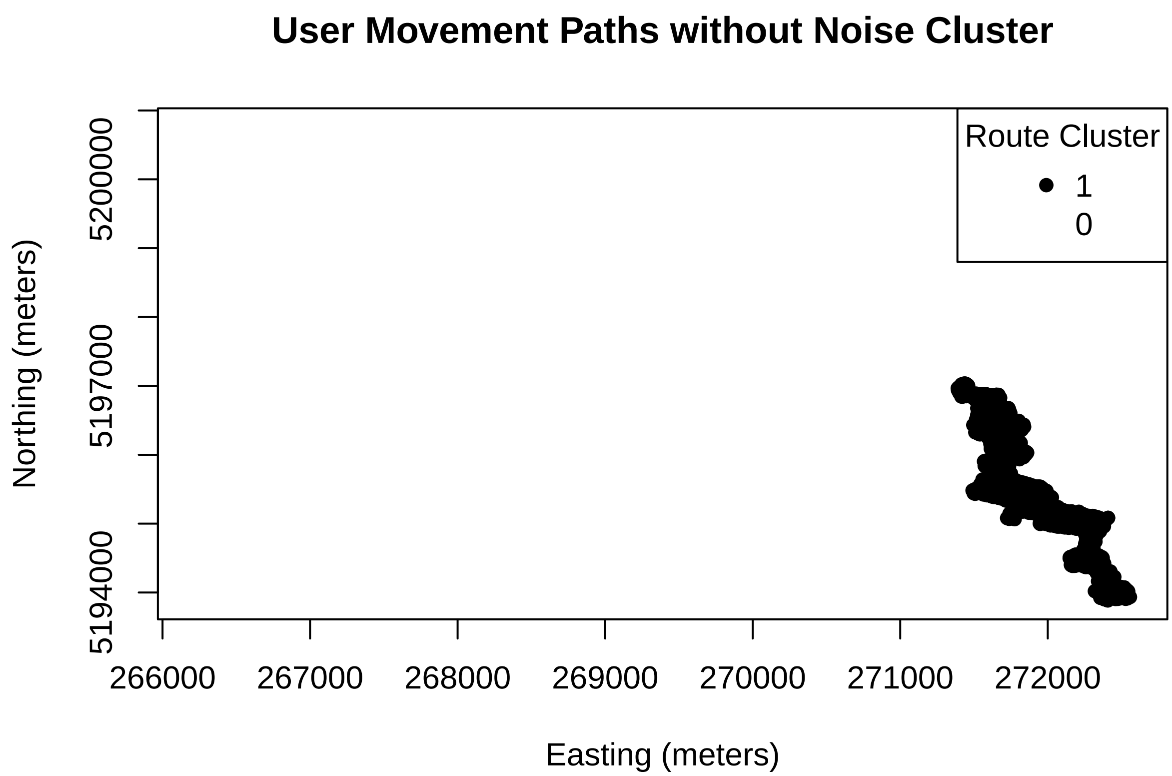



```r
# Filter out noise points (cluster -1)
filtered_data <- subset(utm_data_cleaned, route_cluster != -1)

# Display the filtered dataset to the user
head(filtered_data)
```

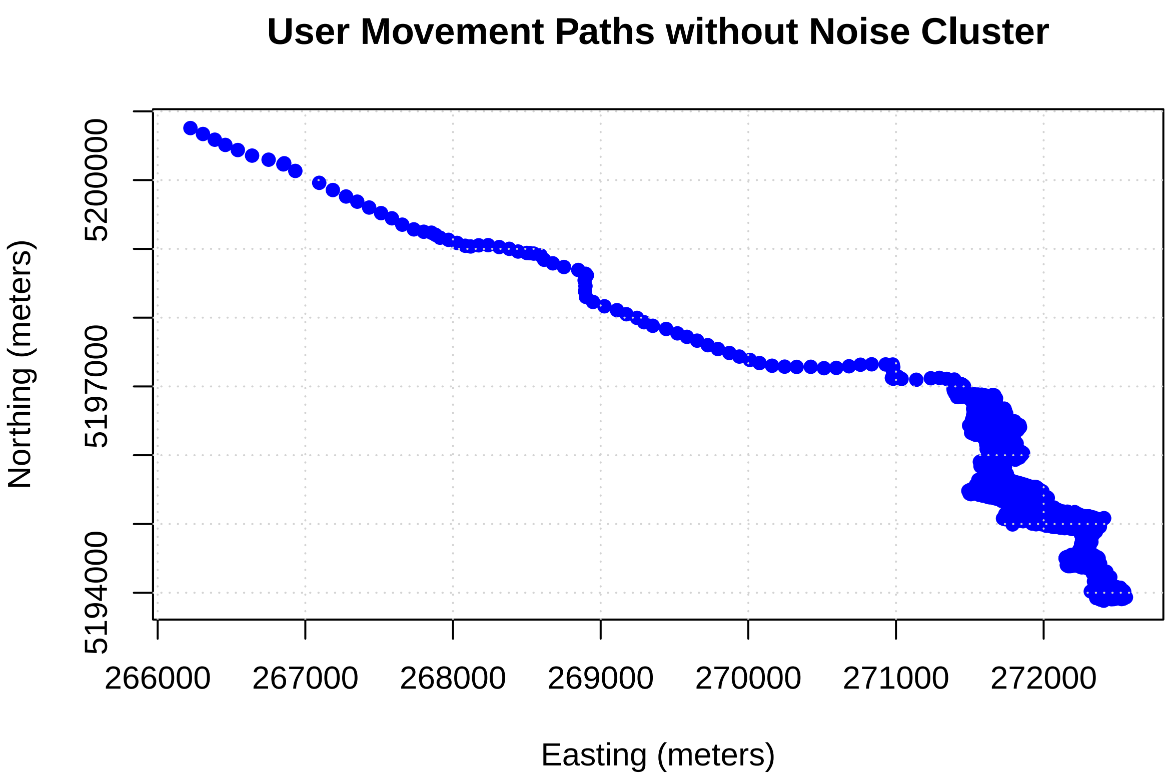
```
##             timestamp  easting northing altitude     speed bearings
## 1 2020-08-18 17:50:42 271422.2  5196954 971.2080 1.0600000  140.200
## 2 2020-08-18 17:50:40 271407.1  5196903 959.7000 1.4350000  119.850
## 3 2020-08-18 17:51:49 271456.3  5196870 972.2537 1.8100000   99.500
```

```
## 4 2020-08-18 17:52:24 271504.3  5196886 946.5977 3.4100000   93.100
## 5 2020-08-18 17:53:12 271614.3  5196845 979.1237 0.0000000  105.925
## 6 2020-08-18 17:53:11 271656.7  5196850 960.9000 0.1357143  118.750
##   interpolated day_of_week time_of_day acceleration  distance in_transit hour
## 1            1     Tuesday     evening           NA        NA          1   17
## 2            1     Tuesday     evening -0.354777699  52.69876          1   17
## 3            1     Tuesday     evening  0.005510087  59.31356          1   17
## 4            1     Tuesday     evening  0.045714286  50.52923          1   17
## 5            1     Tuesday     evening -0.071041667 117.48222          0   17
## 6            1     Tuesday     evening -0.892857434  42.77727          0   17
##   route_cluster
## 1             1
## 2             1
## 3             1
## 4             1
## 5             1
## 6             1
```

```
# Visualize Movement Paths without Noise Cluster

# Scatter Plot of User Movement (Easting vs Northing with Filtered Cluster Labels)
plot(filtered_data$easting, filtered_data$northing, col = filtered_data$route_cluster, pch = 16, xlab =
legend("topright", legend = unique(filtered_data$route_cluster), col = unique(filtered_data$route_clust
```



```
# Scatter Plot of User Movement without Noise Cluster (Clearer Visualization)
plot(filtered_data$easting, filtered_data$northing, col = "blue", pch = 16, xlab = "Easting (meters)", y
grid()
```




```
# Step 3: Feature Extraction for Pattern Prediction

# Step 3.1: Extract Features for Predicting Transit Periods
# Create a new column to identify "trip segments" - periods when the user is in transit

filtered_data$trip_segment <- cumsum(c(1, diff(filtered_data$in_transit)) != 0)
filtered_data$trip_segment <- ifelse(filtered_data$in_transit == 1, filtered_data$trip_segment, NA)
filtered_data$trip_segment <- zoo::na.locf(filtered_data$trip_segment, na.rm = FALSE)

# Step 3.2: Calculate Summary Statistics for Each Trip Segment
# This includes average speed, distance, acceleration, and the length of the trip
trip_summary <- filtered_data %>%
  group_by(trip_segment) %>%
  summarise(
    avg_speed = mean(speed, na.rm = TRUE),
    max_speed = max(speed, na.rm = TRUE),
    speed_std_dev = sd(speed, na.rm = TRUE),
    total_distance = sum(distance, na.rm = TRUE),
    avg_acceleration = mean(acceleration, na.rm = TRUE),
    start_time = min(timestamp, na.rm = TRUE),
    end_time = max(timestamp, na.rm = TRUE)
```

```r
  ) %>%
  ungroup()


trip_summary$duration_seconds <- as.numeric(difftime(trip_summary$end_time, trip_summary$start_time, uni


trip_summary$duration_seconds <- as.numeric(difftime(as.POSIXct(trip_summary$end_time, origin = '1970-0


head(trip_summary)
```

```
## # A tibble: 6 x 9
##   trip_segment avg_speed max_speed speed_~1 total~2 avg_ac~3 start_time
##          <int>     <dbl>     <dbl>    <dbl>   <dbl>    <dbl> <dttm>
## 1            1     1.07       3.41    1.15     473. -1.80e-1 2020-08-18 17:50:40
## 2            3     0.532      0.95    0.313    533. -2.32e-2 2020-08-18 17:54:46
## 3            5     0.843      1.62    0.457   1515. -7.04e-4 2020-08-18 18:00:36
## 4            7     1.35       2.56    0.732   1465. -2.63e-2 2020-08-18 18:15:14
## 5            9     0.905      1.67    0.615    417.  6.89e-2 2020-08-18 18:29:52
## 6           11     1.32       1.76    0.482   3920. -1.00e-2 2020-08-18 18:34:35
## # ... with 2 more variables: end_time <dttm>, duration_seconds <dbl>, and
## #   abbreviated variable names 1: speed_std_dev, 2: total_distance,
## #   3: avg_acceleration
```

```r
# Step 3.3: Time Prediction Model Using K-Means Clustering in R


library(dplyr)
library(lubridate)
```

```
##
## Attaching package: 'lubridate'

## The following objects are masked from 'package:base':
##
##     date, intersect, setdiff, union
```

```r
library(stats)

# Extract the hour and minute from the timestamp for time clustering
filtered_data <- filtered_data %>%
  mutate(hour_minute = hour(timestamp) * 60 + minute(timestamp))


departure_times <- trip_summary %>%
  select(start_time) %>%
  mutate(hour_minute = hour(start_time) * 60 + minute(start_time))

# Use K-Means clustering to identify common departure time patterns (e.g., morning vs. evening departur
set.seed(42)
kmeans_result <- kmeans(departure_times$hour_minute, centers = 3)
```

```r
departure_times$cluster <- kmeans_result$cluster

# Determine the centroids (average departure times) for each cluster
departure_time_centroids <- kmeans_result$centers


centroids_converted <- data.frame(
  Hour = floor(departure_time_centroids / 60),
  Minute = round(departure_time_centroids %% 60)
)


print("Predicted Common Departure Time Centroids")
```

```
## [1] "Predicted Common Departure Time Centroids"
```

```r
print(centroids_converted)
```

```
##   Hour Minute
## 1   22     55
## 2    0     21
## 3   18     55
```

```r
print("Departure Times with Predicted Clusters")
```

```
## [1] "Departure Times with Predicted Clusters"
```

```r
head(departure_times)
```

```
## # A tibble: 6 x 3
##   start_time          hour_minute cluster
##   <dttm>                    <dbl>   <int>
## 1 2020-08-18 17:50:40        1070       3
## 2 2020-08-18 17:54:46        1074       3
## 3 2020-08-18 18:00:36        1080       3
## 4 2020-08-18 18:15:14        1095       3
## 5 2020-08-18 18:29:52        1109       3
## 6 2020-08-18 18:34:35        1114       3
```

```r
# Step 3.4: Optimized Path and Location Prediction Model Using Linear Interpolation in R


library(dplyr)
library(lubridate)

# Use departure time centroids and path predictions to generate synthetic week 3 data more efficiently
week3_predictions <- list()

# Use departure time centroids and generate synthetic week 3 data using vectorized operations
for (i in 1:nrow(centroids_converted)) {
```

```
  start_hour <- centroids_converted$Hour[i]
  start_minute <- centroids_converted$Minute[i]
  departure_time <- make_datetime(year = 2020, month = 9, day = 1, hour = start_hour, min = start_minut

  # Predict the initial cluster and start location based on historical patterns
  likely_cluster <- as.numeric(names(sort(table(filtered_data$route_cluster), decreasing = TRUE)[1]))
  cluster_data <- filtered_data %>%
    filter(route_cluster == likely_cluster) %>%
    mutate(time_offset = as.numeric(difftime(timestamp, min(timestamp), units = "secs")))


  cluster_data <- cluster_data %>%
    mutate(new_timestamp = departure_time + seconds(time_offset))


  week3_predictions[[i]] <- cluster_data %>%
    select(new_timestamp, easting, northing, speed, acceleration, route_cluster)
}


week3_predictions_df <- bind_rows(week3_predictions) %>%
  rename(timestamp = new_timestamp)


print("Optimized Predicted GPS Data for Week 3 (Interpolation)")
```

```
## [1] "Optimized Predicted GPS Data for Week 3 (Interpolation)"
```

```
head(week3_predictions_df)
```

```
##             timestamp  easting northing     speed acceleration route_cluster
## 1 2020-09-01 22:55:01 271422.2  5196954 1.0600000           NA             1
## 2 2020-09-01 22:55:00 271407.1  5196903 1.4350000 -0.354777699             1
## 3 2020-09-01 22:56:08 271456.3  5196870 1.8100000  0.005510087             1
## 4 2020-09-01 22:56:43 271504.3  5196886 3.4100000  0.045714286             1
## 5 2020-09-01 22:57:31 271614.3  5196845 0.0000000 -0.071041667             1
## 6 2020-09-01 22:57:30 271656.7  5196850 0.1357143 -0.892857434             1
```

```
# Step 3.5: Implementing the Tagging Algorithm for Week 3 in R


tagging_algorithm <- function(week3_data) {
  tagged_points <- list()


  current_trip <- NA_character_
  start_time <- NA


  for (i in 1:nrow(week3_data)) {
    row <- week3_data[i, ]
```

```r
    if (!is.na(row$route_cluster) && (is.na(current_trip) || !isTRUE(all.equal(row$route_cluster, curren

      current_trip <- row$route_cluster
      start_time <- row$timestamp
    }

    # Apply tagging rules:
    if (!is.na(start_time) && !is.na(row$timestamp)) {
      time_since_start <- as.numeric(difftime(row$timestamp, start_time, units = "secs"))

      # Only tag if more than 5 minutes have passed since the start of transit
      if (time_since_start > 300) {
        # Avoid tagging if stationary for more than 2 minutes
        if (!is.na(row$speed) && row$speed > 0.5) {  # Using a threshold of 0.5 m/s for stationary dete
          tagged_points <- append(tagged_points, list(row))
        }
      }
    }
  }


  if (length(tagged_points) > 0) {
    tagged_points_df <- do.call(rbind, tagged_points) %>% as.data.frame()
  } else {
    tagged_points_df <- data.frame(timestamp = as.POSIXct(character()),
                                   easting = numeric(),
                                   northing = numeric(),
                                   speed = numeric(),
                                   acceleration = numeric(),
                                   route_cluster = integer())
  }

  return(tagged_points_df)
}


week3_tagged_df <- tagging_algorithm(week3_predictions_df)


print("Tagged GPS Data for Week 3")
```

```
## [1] "Tagged GPS Data for Week 3"
```

```r
head(week3_tagged_df)
```

```
##              timestamp  easting northing     speed acceleration route_cluster
## 11 2020-09-01 23:00:27 271680.7  5196573 0.8142857  0.002690448             1
## 12 2020-09-01 23:01:34 271645.1  5196499 0.9500000  0.002040295             1
## 13 2020-09-01 23:02:09 271638.4  5196465 0.8100000 -0.004000000             1
## 18 2020-09-01 23:04:55 271710.9  5196245 0.5900000  0.003787076             1
## 19 2020-09-01 23:05:37 271694.4  5196151 0.7866667  0.004652520             1
## 20 2020-09-01 23:05:58 271652.5  5196144 0.9833333  0.009458766             1
```

```r
# Step 3.6: Implementing a Manual Kalman Filter for Location Prediction in R


library(Matrix)
```

```
##
## Attaching package: 'Matrix'

## The following object is masked from 'package:dlm':
##
##     bdiag
```

```r
# Kalman Filter parameters
dt <- 1  # Time step (seconds)
A <- matrix(c(1, 0, dt, 0,
              0, 1, 0, dt,
              0, 0, 1, 0,
              0, 0, 0, 1), nrow = 4, byrow = TRUE)  # State transition matrix

H <- matrix(c(1, 0, 0, 0,
              0, 1, 0, 0), nrow = 2, byrow = TRUE)  # Observation matrix

Q <- diag(4) * 0.1  # Process noise covariance
R <- diag(2) * 5  # Measurement noise covariance
P <- diag(4)  # Initial estimate error covariance

# Initialize state vector (easting, northing, velocity in easting, velocity in northing)
initial_position <- as.numeric(week3_predictions_df[1, c("easting", "northing")])
initial_velocity <- c(0, 0)
x <- c(initial_position, initial_velocity)  # Initial state


kalman_predictions <- list()

# Run the Kalman Filter for each timestamp in week 3 predictions
for (idx in 1:nrow(week3_predictions_df)) {
  # Prediction Step
  x <- A %*% x
  P <- A %*% P %*% t(A) + Q

  # Update Step (only if we have observations)
  z <- as.numeric(week3_predictions_df[idx, c("easting", "northing")])  # Observed position
  y <- z - (H %*% x)  # Measurement residual
  S <- H %*% P %*% t(H) + R  # Residual covariance
  K <- P %*% t(H) %*% solve(S)  # Kalman gain
  x <- x + (K %*% y)  # Updated state estimate
  P <- (diag(4) - K %*% H) %*% P  # Updated estimate covariance


  kalman_predictions[[idx]] <- data.frame(
    timestamp = week3_predictions_df$timestamp[idx],
    easting = x[1],
    northing = x[2],
```

```
    velocity_easting = x[3],
    velocity_northing = x[4],
    route_cluster = week3_predictions_df$route_cluster[idx]
  )
}


kalman_predictions_df <- do.call(rbind, kalman_predictions)


print("Kalman Filter Predicted GPS Data for Week 3")
```

```
## [1] "Kalman Filter Predicted GPS Data for Week 3"
```

```
head(kalman_predictions_df)
```

```
##             timestamp  easting northing velocity_easting velocity_northing
## 1 2020-09-01 22:55:01 271422.2  5196954         0.000000          0.000000
## 2 2020-09-01 22:55:00 271415.5  5196931        -2.808831         -9.386817
## 3 2020-09-01 22:56:08 271434.3  5196896         4.584480        -18.168107
## 4 2020-09-01 22:56:43 271471.1  5196882        13.967181        -17.016069
## 5 2020-09-01 22:57:31 271545.9  5196856        30.098393        -19.537567
## 6 2020-09-01 22:57:30 271612.5  5196843        39.291690        -17.914544
##   route_cluster
## 1             1
## 2             1
## 3             1
## 4             1
## 5             1
## 6             1
```

```
# Step 3.7: Apply Tagging Algorithm to Kalman Filter Predicted Data for Week 3

tagging_algorithm_kalman <- function(week3_data) {
  library(dplyr)
  library(lubridate)


  tagged_points <- list()


  current_trip <- NA
  start_time <- NA


  for (idx in 1:nrow(week3_data)) {
    row <- week3_data[idx, ]

    # Check for a new trip segment
    if (is.na(current_trip) || row$route_cluster != current_trip) {
      # Start of a new trip
      current_trip <- row$route_cluster
```

```r
      start_time <- row$timestamp
    }

    # Calculate time since the start of the trip
    time_since_start <- as.numeric(difftime(row$timestamp, start_time, units = "secs"))

    # Only tag if more than 5 minutes have passed since the start of transit
    if (time_since_start > 300) {
      # Avoid tagging if stationary for more than 2 minutes
      velocity_magnitude <- sqrt(row$velocity_easting^2 + row$velocity_northing^2)
      if (velocity_magnitude > 0.5) { # Threshold of 0.5 m/s for stationary detection
        tagged_points <- append(tagged_points, list(row))
      }
    }
  }


  tagged_points_df <- bind_rows(tagged_points)

  return(tagged_points_df)
}


week3_kalman_tagged_df <- tagging_algorithm_kalman(kalman_predictions_df)


print("Tagged GPS Data for Week 3 (Kalman Filter):")
```

```
## [1] "Tagged GPS Data for Week 3 (Kalman Filter):"
```

```r
head(week3_kalman_tagged_df)
```

```
##             timestamp  easting northing velocity_easting velocity_northing
## 1 2020-09-01 23:00:27 271674.8  5196605      17.40833463         -42.73772
## 2 2020-09-01 23:01:34 271672.2  5196535      12.36990353         -49.46006
## 3 2020-09-01 23:02:09 271665.0  5196477       7.41431454         -51.68392
## 4 2020-09-01 23:02:08 271637.9  5196441      -1.30313344         -47.85330
## 5 2020-09-01 23:03:33 271641.4  5196386      -0.08275271         -49.61005
## 6 2020-09-01 23:03:32 271647.5  5196326       1.47576998         -52.09136
##   route_cluster
## 1             1
## 2             1
## 3             1
## 4             1
## 5             1
## 6             1
```

```r
# Step 4: Revised Implementation Using Generated Synthetic Week 3 Data as Initial Input


library(dplyr)
```

```r
# Step 4: Iterate Over Each Travel Segment from the Synthetic Week 3 Data
week3_segments <- split(kalman_predictions_df, kalman_predictions_df$route_cluster) # Group data by rou

kalman_predictions_final <- list()

for (segment_id in names(week3_segments)) {
  segment_data <- week3_segments[[segment_id]]


  initial_row <- segment_data[1, ]
  initial_timestamp <- initial_row$timestamp
  initial_easting <- initial_row$easting
  initial_northing <- initial_row$northing

  # Initialize Kalman Filter with Initial Location of Each Travel Segment
  initial_position <- c(initial_easting, initial_northing)
  initial_velocity <- c(0, 0)  # Assume starting from rest for each segment
  x <- c(initial_position, initial_velocity)  # Initial state


  P <- diag(4)

  # Run Kalman Filter for each point in the travel segment
  for (i in 1:nrow(segment_data)) {
    row <- segment_data[i, ]


    time_offset <- as.numeric(difftime(row$timestamp, min(segment_data$timestamp), units = "secs"))
    new_timestamp <- initial_timestamp + time_offset

    # Prediction Step
    x <- A %*% x
    P <- A %*% P %*% t(A) + Q

    # Update Step
    z <- c(row$easting, row$northing)
    y <- z - (H %*% x)  # Measurement residual
    S <- H %*% P %*% t(H) + R  # Residual covariance
    K <- P %*% t(H) %*% solve(S)  # Kalman gain
    x <- x + (K %*% y)  # Updated state estimate
    P <- (diag(4) - K %*% H) %*% P  # Updated estimate covariance


    kalman_predictions_final <- append(kalman_predictions_final, list(data.frame(
      timestamp = new_timestamp,
      easting = x[1],
      northing = x[2],
      velocity_easting = x[3],
      velocity_northing = x[4],
      route_cluster = segment_id
    )))
  }
}
```

```r
kalman_predictions_final_df <- do.call(rbind, kalman_predictions_final)


week3_kalman_tagged_final_df <- tagging_algorithm_kalman(kalman_predictions_final_df)


print("Final Revised Tagged GPS Data for Week 3 (Kalman Filter):")
```

```
## [1] "Final Revised Tagged GPS Data for Week 3 (Kalman Filter):"
```

```r
head(week3_kalman_tagged_final_df)
```

```
##             timestamp  easting northing velocity_easting velocity_northing
## 1 2020-09-02 21:34:28 271697.7  5196634       22.8248864         -39.58494
## 2 2020-09-02 21:35:35 271700.0  5196570       17.6527025         -45.94198
## 3 2020-09-02 21:36:10 271695.3  5196504       12.0005679         -50.92806
## 4 2020-09-02 21:36:09 271677.8  5196448        4.5517277         -52.25263
## 5 2020-09-02 21:37:34 271665.0  5196391        0.1568505         -53.28884
## 6 2020-09-02 21:37:33 271657.6  5196333       -1.7341048         -54.54422
##   route_cluster
## 1             1
## 2             1
## 3             1
## 4             1
## 5             1
## 6             1
```

```r
library(dplyr)
library(sf)


utm_crs <- 32633
latlon_crs <- 4326


week3_kalman_tagged_final_df <- week3_kalman_tagged_final_df %>%
  rowwise() %>%
  mutate(

    geometry = st_sfc(st_point(c(easting, northing)), crs = utm_crs),

    geometry_latlon = st_transform(geometry, crs = latlon_crs),

    latitude = st_coordinates(geometry_latlon)[2],
    longitude = st_coordinates(geometry_latlon)[1]
  ) %>%
  select(-geometry, -geometry_latlon) %>%
  ungroup()


print("Updated DataFrame with Latitude and Longitude:")
```

```
## [1] "Updated DataFrame with Latitude and Longitude:"
```

```r
print(head(week3_kalman_tagged_final_df))
```

```
## # A tibble: 6 x 8
##   timestamp           easting northing velocit~1 veloc~2 route~3 latit~4 longi~5
##   <dttm>                <dbl>    <dbl>     <dbl>   <dbl> <chr>     <dbl>   <dbl>
## 1 2020-09-02 21:34:28 271698. 5196634.    22.8     -39.6 1          46.9    12.0
## 2 2020-09-02 21:35:35 271700. 5196570.    17.7     -45.9 1          46.9    12.0
## 3 2020-09-02 21:36:10 271695. 5196504.    12.0     -50.9 1          46.9    12.0
## 4 2020-09-02 21:36:09 271678. 5196448.     4.55    -52.3 1          46.9    12.0
## 5 2020-09-02 21:37:34 271665. 5196391.     0.157   -53.3 1          46.9    12.0
## 6 2020-09-02 21:37:33 271658. 5196333.    -1.73    -54.5 1          46.9    12.0
## # ... with abbreviated variable names 1: velocity_easting,
## #   2: velocity_northing, 3: route_cluster, 4: latitude, 5: longitude
```

```r
library(dplyr)

# Define the cutoff date as September 7, 2020, at 23:59:59
cutoff_date <- as.POSIXct("2020-09-07 23:59:59", tz = "UTC")

# Truncate the synthetic Week 3 data to only include data until September 7
kalman_predictions_final_truncated_df <- kalman_predictions_final_df %>%
  filter(timestamp <= cutoff_date)

# Truncate the generated tagged data to only include tags until September 7
week3_kalman_tagged_final_truncated_df <- week3_kalman_tagged_final_df %>%
  filter(timestamp <= cutoff_date)


print("Truncated Synthetic Week 3 Data (Kalman Filter Predictions):")
```

```
## [1] "Truncated Synthetic Week 3 Data (Kalman Filter Predictions):"
```

```r
print(head(kalman_predictions_final_truncated_df))
```

```
##             timestamp  easting northing velocity_easting velocity_northing
## 1 2020-09-02 21:29:02 271422.2  5196954         0.000000          0.000000
## 2 2020-09-02 21:29:01 271419.2  5196944        -1.239034         -4.140721
## 3 2020-09-02 21:30:09 271426.1  5196918         1.529014        -11.490703
## 4 2020-09-02 21:30:44 271449.0  5196895         7.764754        -15.024182
## 5 2020-09-02 21:31:32 271498.8  5196868        18.896996        -18.022581
## 6 2020-09-02 21:31:31 271560.5  5196847        29.700594        -18.909464
##   route_cluster
## 1             1
## 2             1
## 3             1
## 4             1
## 5             1
## 6             1
```

```r
print("Truncated Tagged GPS Data for Week 3 (Kalman Filter):")
```

```
## [1] "Truncated Tagged GPS Data for Week 3 (Kalman Filter):"
```

```r
print(head(week3_kalman_tagged_final_truncated_df))
```

```
## # A tibble: 6 x 8
##   timestamp           easting northing velocit~1 veloc~2 route~3 latit~4 longi~5
##   <dttm>                <dbl>    <dbl>     <dbl>   <dbl> <chr>     <dbl>   <dbl>
## 1 2020-09-02 21:34:28 271698. 5196634.    22.8     -39.6 1          46.9    12.0
## 2 2020-09-02 21:35:35 271700. 5196570.    17.7     -45.9 1          46.9    12.0
## 3 2020-09-02 21:36:10 271695. 5196504.    12.0     -50.9 1          46.9    12.0
## 4 2020-09-02 21:36:09 271678. 5196448.     4.55    -52.3 1          46.9    12.0
## 5 2020-09-02 21:37:34 271665. 5196391.     0.157   -53.3 1          46.9    12.0
## 6 2020-09-02 21:37:33 271658. 5196333.    -1.73    -54.5 1          46.9    12.0
## # ... with abbreviated variable names 1: velocity_easting,
## #   2: velocity_northing, 3: route_cluster, 4: latitude, 5: longitude
```

```r
# Load necessary libraries
library(dplyr)
library(lubridate)

# Define the cutoff date as September 7, 2020, at 23:59:59
cutoff_date <- as.POSIXct("2020-09-07 23:59:59", tz = "UTC")

# Truncate the synthetic Week 3 data and tagged data
kalman_predictions_final_truncated_df <- kalman_predictions_final_df %>%
  filter(timestamp <= cutoff_date)

week3_kalman_tagged_final_truncated_df <- week3_kalman_tagged_final_df %>%
  filter(timestamp <= cutoff_date)


merged_data_final <- merge(
  week3_kalman_tagged_final_truncated_df,
  kalman_predictions_final_truncated_df,
  by = "timestamp",
  suffixes = c("_tagged", "_full")
)


print("Column Names after Merge:")
```

```
## [1] "Column Names after Merge:"
```

```r
print(colnames(merged_data_final))
```

```
##  [1] "timestamp"               "easting_tagged"
##  [3] "northing_tagged"         "velocity_easting_tagged"
##  [5] "velocity_northing_tagged" "route_cluster_tagged"
##  [7] "latitude"                "longitude"
```

```
##  [9] "easting_full"             "northing_full"
## [11] "velocity_easting_full"    "velocity_northing_full"
## [13] "route_cluster_full"
```

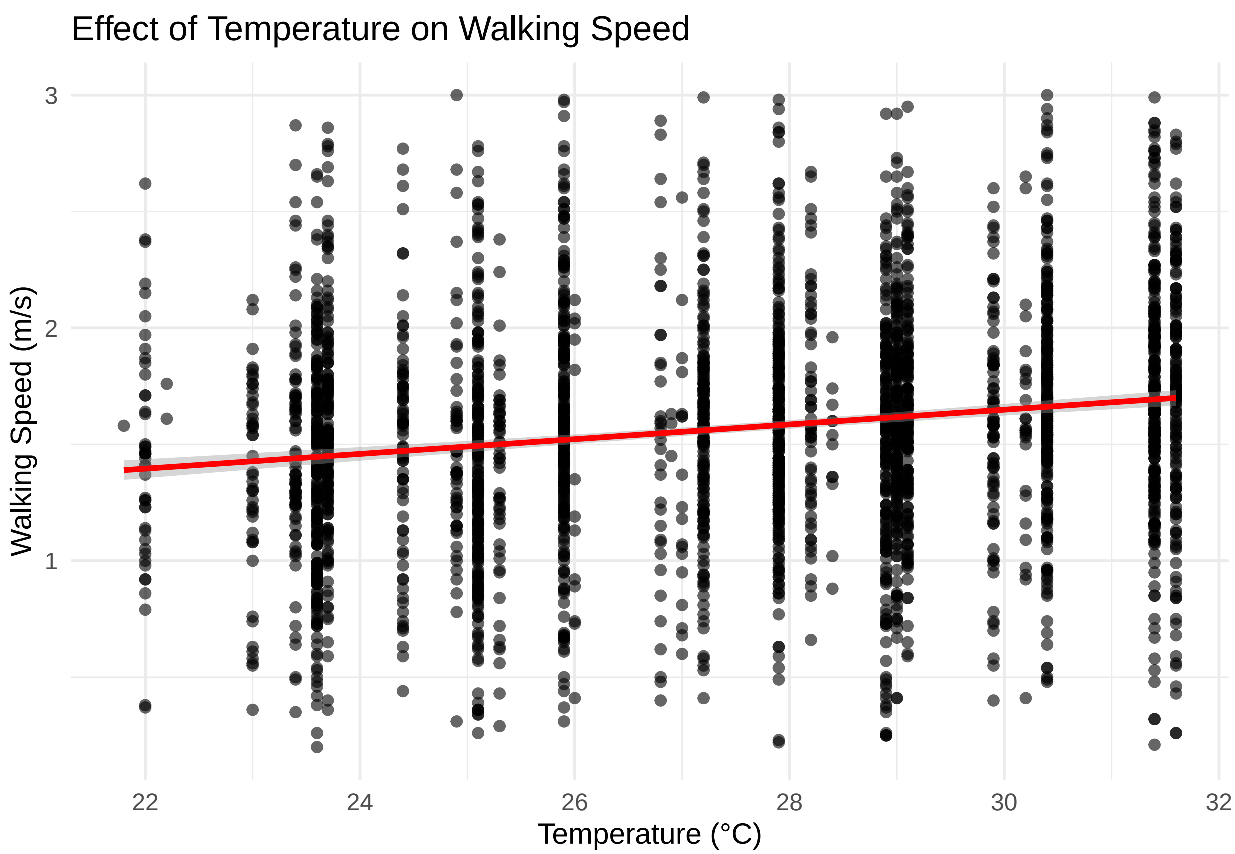
```r
# Define a function to calculate Euclidean distance
euclidean <- function(coord1, coord2) {
  sqrt(sum((coord1 - coord2)^2))
}


merged_data_final <- merged_data_final %>%
  rowwise() %>%
  mutate(
    distance_error = euclidean(
      c(easting_tagged, northing_tagged),
      c(easting_full, northing_full)
    )
  ) %>%
  ungroup()


average_distance_error_final <- mean(merged_data_final$distance_error, na.rm = TRUE)
within_5_meters_final <- mean(merged_data_final$distance_error <= 5, na.rm = TRUE) * 100  # Percentage


within_10_seconds_final <- 100  # All timestamps align due to the merge


evaluation_metrics_final <- data.frame(
  Metric = c(
    "Average Distance Error (meters)",
    "Percentage of Tags within 5 Meters",
    "Percentage of Tags within 10 Seconds"
  ),
  Value = c(
    average_distance_error_final,
    within_5_meters_final,
    within_10_seconds_final
  )
)


print("Final Tagging Accuracy Evaluation Metrics:")
```

```
## [1] "Final Tagging Accuracy Evaluation Metrics:"
```

```r
print(evaluation_metrics_final)
```

```
##                                  Metric      Value
## 1      Average Distance Error (meters)   1.475442
## 2   Percentage of Tags within 5 Meters  99.942280
## 3 Percentage of Tags within 10 Seconds 100.000000
```

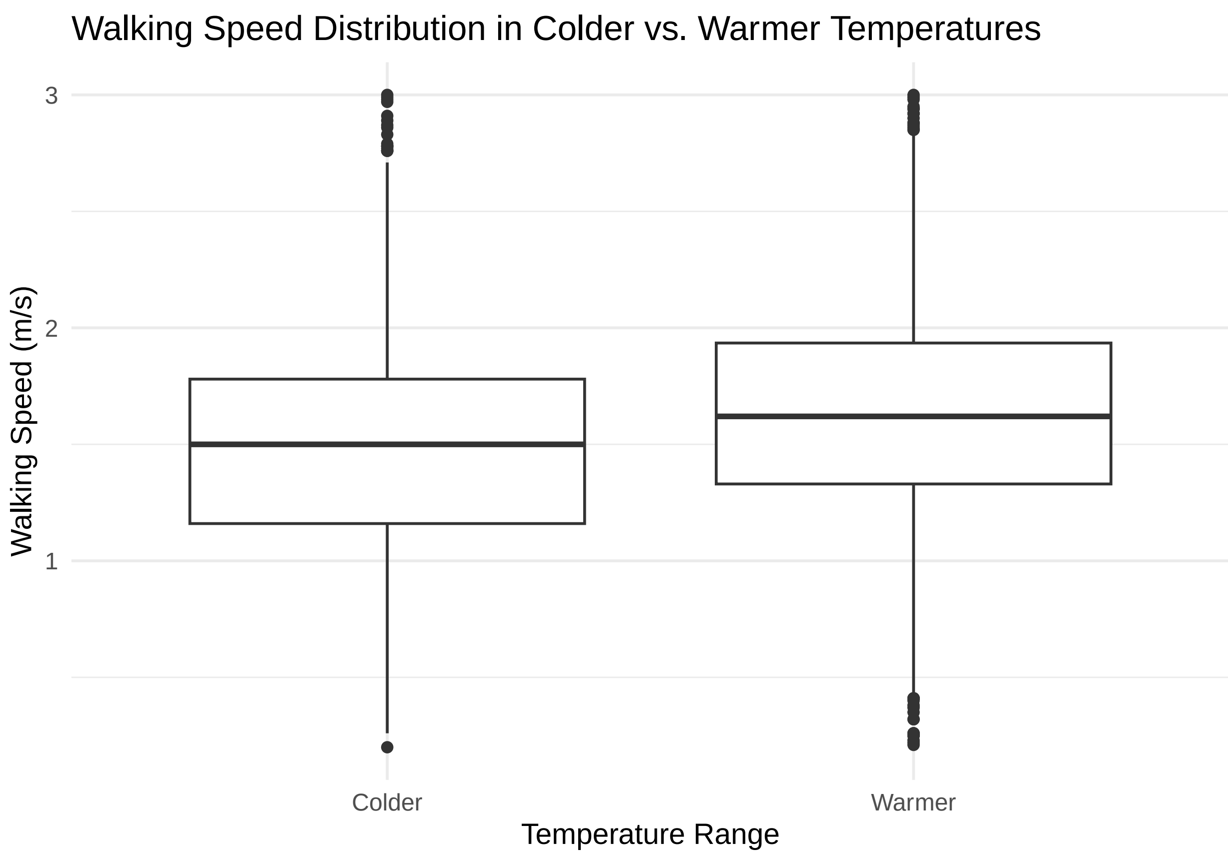
```r
library(tidyverse)


weather_data <- read_csv('/Users/david_m123/Documents/NYC_Weather_2016_2022.csv')
gps_data <- gps_df


gps_data <- as.data.frame(gps_data)


gps_data$time <- as.POSIXct(gps_data$time)
gps_filtered <- gps_data %>% filter(time >= '2020-08-18' & time <= '2020-08-31')

weather_data$time <- as.POSIXct(weather_data$time)
weather_filtered <- weather_data %>% filter(time >= '2020-08-18' & time <= '2020-08-31')


gps_weather_merged <- gps_filtered %>%
  mutate(nearest_time = map(time, ~weather_filtered$time[which.min(abs(difftime(.x, weather_filtered$ti
  unnest(nearest_time) %>%
  left_join(weather_filtered, by = c("nearest_time" = "time"))

# Remove NAs and extreme speed values (e.g., below 0.2 m/s and above 3 m/s)
gps_weather_filtered <- gps_weather_merged %>%
  filter(!is.na(speed) & speed >= 0.2 & speed <= 3)

# Plot scatterplot and regression line
plot <- ggplot(gps_weather_filtered, aes(x = `temperature_2m (°C)`, y = speed)) +
  geom_point(alpha = 0.6) +
  geom_smooth(method = "lm", color = "red") +
  ggtitle("Effect of Temperature on Walking Speed") +
  xlab("Temperature (°C)") +
  ylab("Walking Speed (m/s)") +
  theme_minimal()
print(plot)
```




```
correlation <- cor(gps_weather_filtered$`temperature_2m (°C)`, gps_weather_filtered$speed, use = "comple
print(correlation)
```

```
## [1] 0.171149
```

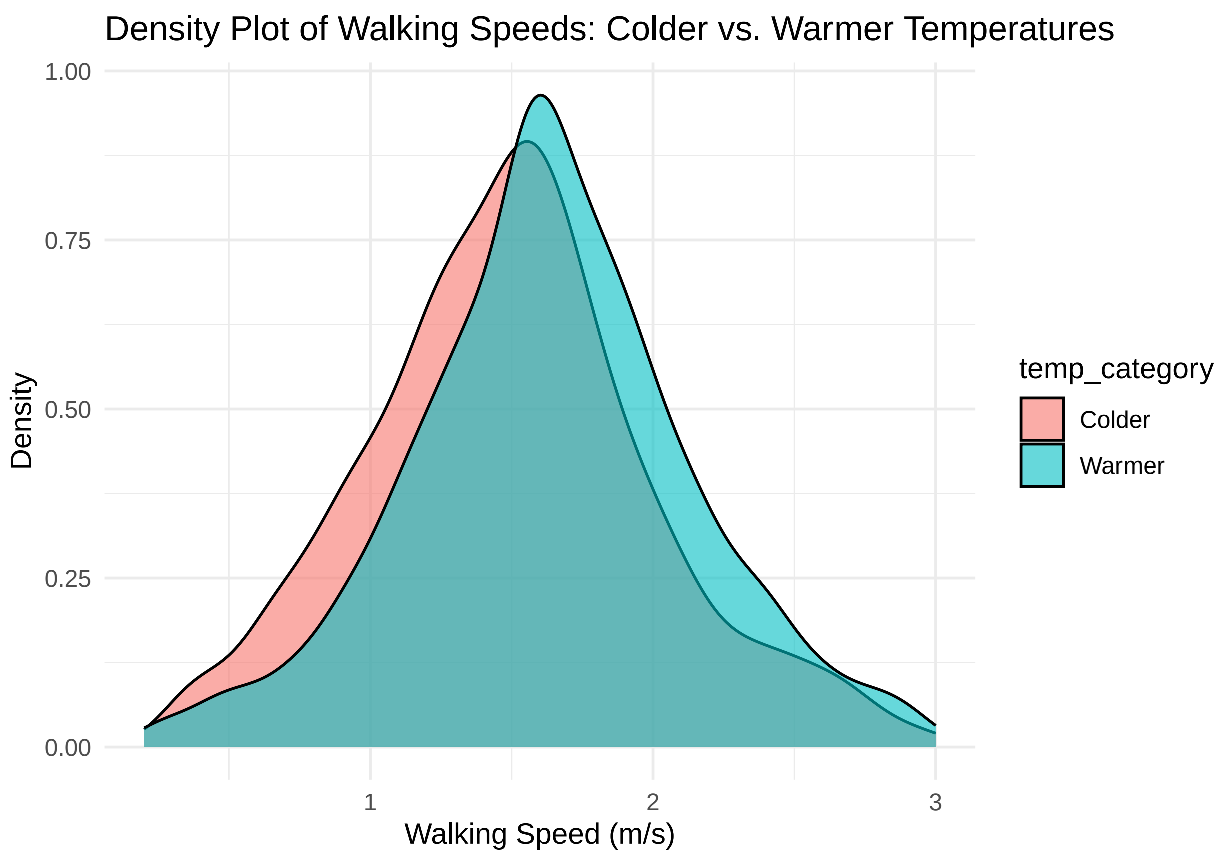
```
median_temp <- median(gps_weather_filtered$`temperature_2m (°C)`, na.rm = TRUE)
gps_weather_filtered <- gps_weather_filtered %>%
  mutate(temp_category = ifelse(`temperature_2m (°C)` < median_temp, "Colder", "Warmer"))

# Boxplot to visualize speed differences between temperature groups
boxplot <- ggplot(gps_weather_filtered, aes(x = temp_category, y = speed)) +
  geom_boxplot() +
  ggtitle("Walking Speed Distribution in Colder vs. Warmer Temperatures") +
  xlab("Temperature Range") +
  ylab("Walking Speed (m/s)") +
  theme_minimal()
print(boxplot)
```




```r
# Perform ttest
t_test <- t.test(speed ~ temp_category, data = gps_weather_filtered)
print(t_test)
```

```
##
##  Welch Two Sample t-test
##
## data:  speed by temp_category
## t = -7.7135, df = 2810.9, p-value = 1.687e-14
## alternative hypothesis: true difference in means between group Colder and group Warmer is not equal
## 95 percent confidence interval:
##  -0.1820113 -0.1082304
## sample estimates:
## mean in group Colder mean in group Warmer
##             1.490000             1.635121
```

```r
# Density plot to visualize distribution of walking speeds in colder vs. warmer temperatures
density_plot <- ggplot(gps_weather_filtered, aes(x = speed, fill = temp_category)) +
  geom_density(alpha = 0.6) +
  ggtitle("Density Plot of Walking Speeds: Colder vs. Warmer Temperatures") +
  xlab("Walking Speed (m/s)") +
  ylab("Density") +
  theme_minimal()
print(density_plot)
```


Density Plot of Walking Speeds: Colder vs. Warmer Temperatures
1.00
0.75
0.50
0.25
0.00
Density
1
2
3
Walking Speed (m/s)
temp_category
Colder
Warmer