R-Code des Semesterprojektes - Unbeabsichtigte Begegnungen und Bewegungsmuster: Eine Analyse von Bewegungsdaten innerhalb einer Gemeinde

Project for Patterns and trends in environmental data MSc ENR FS24

Serafin Martig und Nadja Pfister (Arbeitsteilung 50:50)

Libraries

```
# Load necessary libraries
library(sf)
library(jsonlite)
library(dplyr)
library(tmap)
library(purrr)
library(ggplot2)
library(lubridate)
library(leaflet)
library(tmap)
library(ggmap)
library(ggspatial)
library(prettymapr)
library(osmdata)
library(readr)
# install.packages("osmdata")
```

#Daten einlesen

```
# Load and process your data
#B
```

```r
records_json_S <- jsonlite::read_json("C:/_Data/Master/PaT_24/Projectwork/RProjectwork/Ausgar
records_S <- records_json_S[[1]] |>
  mutate( Person = "B")


#A
records_json_N <- jsonlite::read_json("C:/_Data/Master/PaT_24/Projectwork/RProjectwork/Ausgar
records_N <- records_json_N[[1]] |>
  mutate( Person = "A") %>%
  mutate(Year = year(ymd_hms(timestamp))) %>%
  filter(Year %in% c(2022, 2023, 2024))

#
records_N2 <- records_json_N[[1]] |>
  mutate( Person = "A") %>%
  mutate(Year = year(ymd_hms(timestamp))) |>
  mutate(Month = month(ymd_hms(timestamp))) %>%
  filter(Year %in% c(2024)) |>
  filter(Month %in% c(6))



#
# # zusammenführen
records <- bind_rows(records_N, records_S)



# Convert the data to an sf object and prepare coordinates
records_sf_both <- records %>%
  mutate(
    lat = latitudeE7 / 1e7,
    lon = longitudeE7 / 1e7,
    lat_copy = latitudeE7 / 1e7,  # Kopie der Latitude
    lon_copy = longitudeE7 / 1e7,  # Kopie der Longitude
    timestamp = as.POSIXct(timestamp, origin = "1970-01-01", tz = "UTC"),
    activity = map_chr(activity, ~ .x[[1]]$type %||% "UNKNOWN") # Assuming there is always at
  ) %>%
  st_as_sf(coords = c("lon", "lat"), crs = 4326)
```

```r
names(records_sf_both)
```

```
 [1] "latitudeE7"        "longitudeE7"       "accuracy"
 [4] "source"            "deviceTag"         "deviceDesignation"
 [7] "timestamp"         "activity"          "altitude"
[10] "verticalAccuracy"  "platformType"      "osLevel"
[13] "serverTimestamp"   "deviceTimestamp"   "batteryCharging"
[16] "formFactor"        "velocity"          "heading"
[19] "locationMetadata"  "inferredLocation"  "placeId"
[22] "activeWifiScan"    "Person"            "Year"
[25] "lat_copy"          "lon_copy"          "geometry"
```

#Daten filtern

```r
str(records_sf_both)
```

```
Classes 'sf' and 'data.frame':  318933 obs. of  27 variables:
 $ latitudeE7       : int  467422864 467423301 467423301 467423301 467423301 467423301 467423
 $ longitudeE7      : int  97779715 97778860 97778860 97778860 97778860 97778860 97774770 977
 $ accuracy         : int  6 7 7 7 7 7 15 20 20 18 ...
 $ source           : chr  "GPS" "GPS" "GPS" "GPS" ...
 $ deviceTag        : int  467305824 467305824 467305824 467305824 467305824 467305824 467305
 $ deviceDesignation: chr  NA NA NA NA ...
 $ timestamp        : POSIXct, format: "2022-01-01" "2022-01-01" ...
 $ activity         : chr  "UNKNOWN" "UNKNOWN" "UNKNOWN" "UNKNOWN" ...
 $ altitude         : int  NA NA NA NA NA NA NA NA NA NA ...
 $ verticalAccuracy : int  NA NA NA NA NA NA NA NA NA NA ...
 $ platformType     : chr  NA NA NA NA ...
 $ osLevel          : int  NA NA NA NA NA NA NA NA NA NA ...
 $ serverTimestamp  : chr  NA NA NA NA ...
 $ deviceTimestamp  : chr  NA NA NA NA ...
 $ batteryCharging  : logi  NA NA NA NA NA NA ...
 $ formFactor       : chr  NA NA NA NA ...
 $ velocity         : int  NA NA NA NA NA NA NA NA NA NA ...
 $ heading          : int  NA NA NA NA NA NA NA NA NA NA ...
 $ locationMetadata :List of 318933
  ..$ : NULL
  ..$ : NULL
  ..$ : NULL
```

```
..$ : NULL
..$ : NULL
..$ : NULL
..$ : NULL
..$ : NULL
..$ : NULL
..$ : NULL
..$ : NULL
..$ : NULL
..$ : NULL
..$ : NULL
..$ : NULL
..$ : NULL
..$ : NULL
..$ : NULL
..$ : NULL
..$ : NULL
..$ : NULL
..$ : NULL
..$ : NULL
..$ : NULL
..$ : NULL
..$ : NULL
..$ : NULL
..$ : NULL
..$ : NULL
..$ : NULL
..$ : NULL
..$ : NULL
..$ : NULL
..$ : NULL
..$ : NULL
..$ : NULL
..$ : NULL
..$ : NULL
..$ : NULL
..$ : NULL
..$ : NULL
..$ : NULL
..$ : NULL
..$ : NULL
..$ : NULL
..$ : NULL
```

```
..$ : NULL
..$ : NULL
..$ : NULL
..$ : NULL
..$ : NULL
..$ : NULL
..$ : NULL
..$ : NULL
..$ : NULL
..$ : NULL
..$ : NULL
..$ : NULL
..$ : NULL
..$ : NULL
..$ : NULL
..$ : NULL
..$ : NULL
..$ : NULL
..$ : NULL
..$ : NULL
..$ : NULL
..$ : NULL
..$ : NULL
..$ : NULL
..$ : NULL
..$ : NULL
..$ : NULL
..$ : NULL
..$ : NULL
..$ : NULL
..$ : NULL
..$ : NULL
..$ : NULL
..$ : NULL
..$ : NULL
..$ : NULL
..$ : NULL
..$ : NULL
..$ : NULL
..$ : NULL
..$ : NULL
..$ : NULL
..$ : NULL
```

```
  ..$ : NULL
  ..$ : NULL
  ..$ : NULL
  ..$ : NULL
  ..$ : NULL
  ..$ : NULL
  ..$ : NULL
  ..$ : NULL
  ..$ : NULL
  ..$ : NULL
  .. [list output truncated]
 $ inferredLocation :List of 318933
  ..$ : NULL
  ..$ : NULL
  ..$ : NULL
  ..$ : NULL
  ..$ : NULL
  ..$ : NULL
  ..$ : NULL
  ..$ : NULL
  ..$ : NULL
  ..$ : NULL
  ..$ : NULL
  ..$ : NULL
  ..$ : NULL
  ..$ : NULL
  ..$ : NULL
  ..$ : NULL
  ..$ : NULL
  ..$ : NULL
  ..$ : NULL
  ..$ : NULL
  ..$ : NULL
  ..$ : NULL
  ..$ : NULL
  ..$ : NULL
  ..$ : NULL
  ..$ : NULL
  ..$ : NULL
  ..$ : NULL
  ..$ : NULL
  ..$ : NULL
  ..$ : NULL
```

```
..$ : NULL
..$ : NULL
..$ : NULL
..$ : NULL
..$ : NULL
..$ : NULL
..$ : NULL
..$ : NULL
..$ : NULL
..$ : NULL
..$ : NULL
..$ : NULL
..$ : NULL
..$ : NULL
..$ : NULL
..$ : NULL
..$ : NULL
..$ : NULL
..$ : NULL
..$ : NULL
..$ : NULL
..$ : NULL
..$ : NULL
..$ : NULL
..$ : NULL
..$ : NULL
..$ : NULL
..$ : NULL
..$ : NULL
..$ : NULL
..$ : NULL
..$ : NULL
..$ : NULL
..$ : NULL
..$ : NULL
..$ : NULL
..$ : NULL
..$ : NULL
..$ : NULL
..$ : NULL
..$ : NULL
..$ : NULL
..$ : NULL
```

```
 ..$ : NULL
 ..$ : NULL
 ..$ : NULL
 ..$ : NULL
 ..$ : NULL
 ..$ : NULL
 ..$ : NULL
 ..$ : NULL
 ..$ : NULL
 ..$ : NULL
 ..$ : NULL
 ..$ : NULL
 ..$ : NULL
 ..$ : NULL
 ..$ : NULL
 ..$ : NULL
 ..$ : NULL
 ..$ : NULL
 ..$ : NULL
 ..$ : NULL
 ..$ : NULL
 ..$ : NULL
 ..$ : NULL
 ..$ : NULL
 .. [list output truncated]
$ placeId          : chr  NA NA NA NA ...
$ activeWifiScan   :'data.frame':  318933 obs. of  1 variable:
 ..$ accessPoints:List of 318933
 .. ..$ : NULL
 .. ..$ : NULL
 .. ..$ : NULL
 .. ..$ : NULL
 .. ..$ : NULL
 .. ..$ : NULL
 .. ..$ : NULL
 .. ..$ : NULL
 .. ..$ : NULL
 .. ..$ : NULL
 .. ..$ : NULL
 .. ..$ : NULL
 .. ..$ : NULL
 .. ..$ : NULL
 .. ..$ : NULL
```

```
.. ..$ : NULL
.. ..$ : NULL
.. ..$ : NULL
.. ..$ : NULL
.. ..$ : NULL
.. ..$ : NULL
.. ..$ : NULL
.. ..$ : NULL
.. ..$ : NULL
.. ..$ : NULL
.. ..$ : NULL
.. ..$ : NULL
.. ..$ : NULL
.. ..$ : NULL
.. ..$ : NULL
.. ..$ : NULL
.. ..$ : NULL
.. ..$ : NULL
.. ..$ : NULL
.. ..$ : NULL
.. ..$ : NULL
.. ..$ : NULL
.. ..$ : NULL
.. ..$ : NULL
.. ..$ : NULL
.. ..$ : NULL
.. ..$ : NULL
.. ..$ : NULL
.. ..$ : NULL
.. ..$ : NULL
.. ..$ : NULL
.. ..$ : NULL
.. ..$ : NULL
.. ..$ : NULL
.. ..$ : NULL
.. ..$ : NULL
.. ..$ : NULL
.. ..$ : NULL
.. ..$ : NULL
.. ..$ : NULL
.. ..$ : NULL
.. ..$ : NULL
.. ..$ : NULL
```

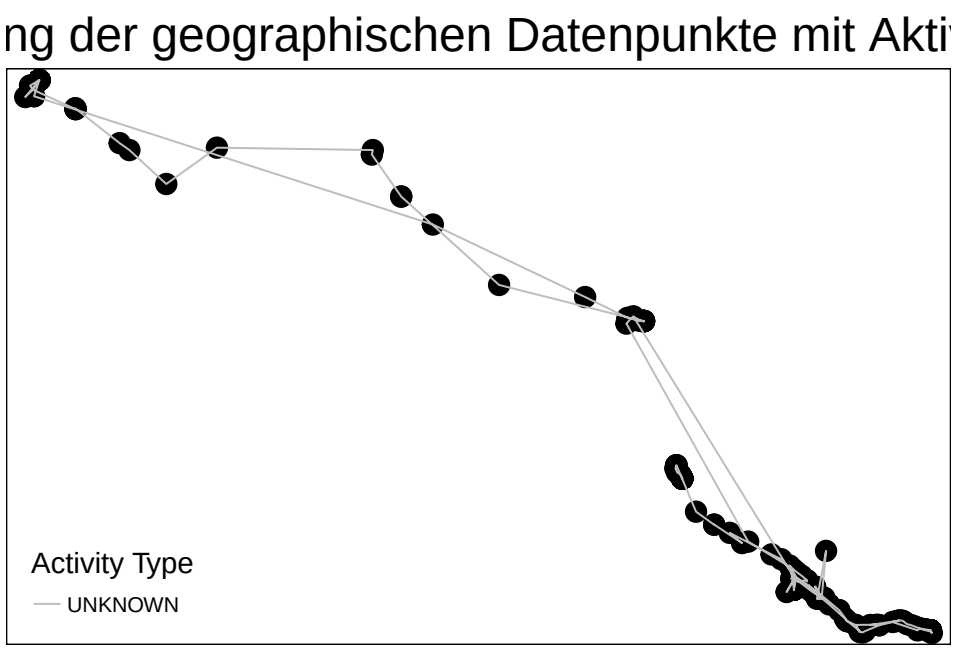
```
.. ..$ : NULL
.. ..$ : NULL
.. ..$ : NULL
.. ..$ : NULL
.. ..$ : NULL
.. ..$ : NULL
.. ..$ : NULL
.. ..$ : NULL
.. ..$ : NULL
.. ..$ : NULL
.. ..$ : NULL
.. ..$ : NULL
.. ..$ : NULL
.. ..$ : NULL
.. ..$ : NULL
.. ..$ : NULL
.. ..$ : NULL
.. ..$ : NULL
.. ..$ : NULL
.. ..$ : NULL
.. ..$ : NULL
.. ..$ : NULL
.. ..$ : NULL
.. ..$ : NULL
.. ..$ : NULL
.. ..$ : NULL
.. ..$ : NULL
.. ..$ : NULL
.. ..$ : NULL
.. ..$ : NULL
.. ..$ : NULL
.. ..$ : NULL
.. ..$ : NULL
.. ..$ : NULL
.. ..$ : NULL
.. ..$ : NULL
.. ..$ : NULL
.. ..$ : NULL
.. ..$ : NULL
.. ..$ : NULL
.. ..$ : NULL
.. ..$ : NULL
.. .. [list output truncated]
```

```
 $ Person           : chr  "A" "A" "A" "A" ...
 $ Year             : num  2022 2022 2022 2022 2022 ...
 $ lat_copy         : num  46.7 46.7 46.7 46.7 46.7 ...
 $ lon_copy         : num  9.78 9.78 9.78 9.78 9.78 ...
 $ geometry         :sfc_POINT of length 318933; first list element:  'XY' num  9.78 46.74
 - attr(*, "sf_column")= chr "geometry"
 - attr(*, "agr")= Factor w/ 3 levels "constant","aggregate",..: NA NA NA NA NA NA NA NA NA N
  ..- attr(*, "names")= chr [1:26] "latitudeE7" "longitudeE7" "accuracy" "source" ...
```

```r
# Filtern nach dem Jahr 2024
records_sf <- records_sf_both %>%
  filter(year(timestamp) == 2024)
```

#Plot Activity an einzelnem Tag

```r
# Filter Date
records_filtered <- records_sf_both %>%
  filter(as.Date(timestamp) == as.Date("2024-04-08"))

# Transform point data to a line representation
records_lines <- records_filtered %>%
  arrange(timestamp) %>%
  summarise(geometry = st_combine(geometry), activity = first(activity), .groups = "drop") %>
  st_cast("LINESTRING")  # Cast MULTIPOINT to LINESTRING

# Define a color palette for activity types
activity_colors <- c("ON_FOOT" = "green", "TILTING" = "orange", "UNKNOWN" = "grey", "STILL" =

# Visualize the point and line data on the map
# tmap <-
  tm_shape(records_filtered) +
  tm_dots(size = 0.5, col = "black") +
  tm_shape(records_lines) +
  tm_lines(col = "activity", palette = activity_colors, title.col = "Activity Type") +
  tm_basemap(server = "OpenStreetMap") +
  tm_layout(main.title = "Visualisierung der geographischen Datenpunkte mit Aktivitätsniveau
            main.title.position = "center")
```

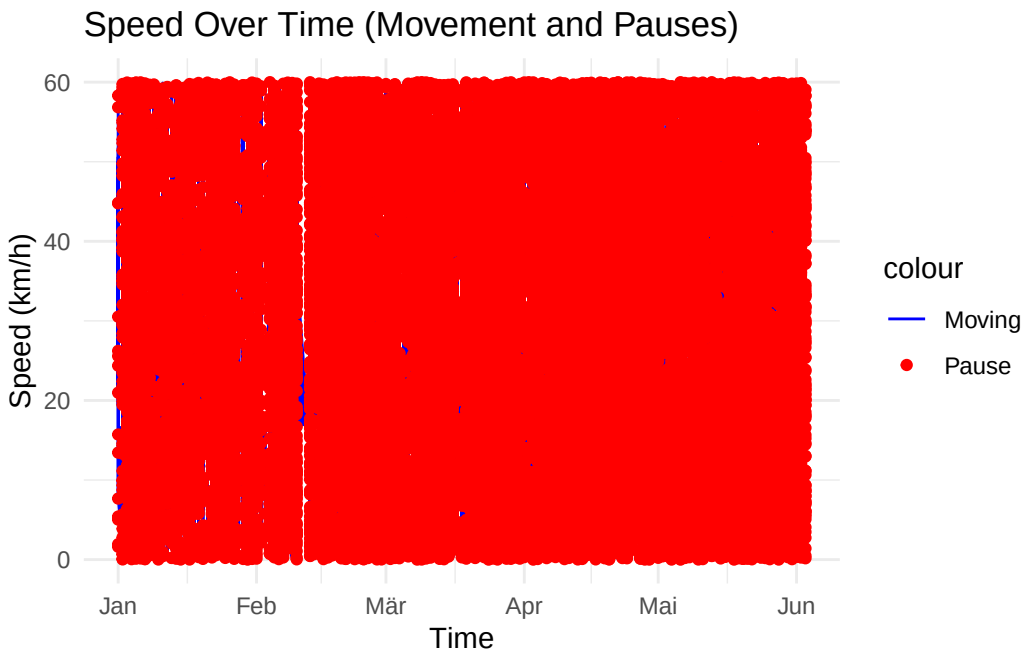



```
# Print the map
# print(tmap)
```

#Geschwindigkeit über Zeit mit Pausen » es müssen noch auf Tage oder Zeitperiode definiert werden

```
library(ggplot2)
library(dplyr)



# Hinzufügen einer simulierten Geschwindigkeit für das Beispiel
records_sf <- records_sf %>%
  mutate(
    speed = runif(n = n(), min = 0, max = 60),  # Zufällige Geschwindigkeit zwischen 0 und 60
    activity = if_else(runif(n = n()) > 0.5, "moving", "pause")  # Zufällige Aktivitätszuweis
  )

# Plot für Geschwindigkeit über Zeit mit Pausen
# moving_speed_plot_with_pauses <-
  ggplot() +
  geom_line(data = filter(records_sf, activity == "moving"),
```

```
            aes(x = timestamp, y = speed, color = "Moving")) +
  geom_point(data = filter(records_sf, activity == "pause"),
             aes(x = timestamp, y = speed, color = "Pause")) +
  scale_color_manual(values = c("Moving" = "blue", "Pause" = "red")) +
  labs(title = "Speed Over Time (Movement and Pauses)", x = "Time", y = "Speed (km/h)") +
  theme_minimal()
```




```
# Drucken des Plots
# print(moving_speed_plot_with_pauses)
```

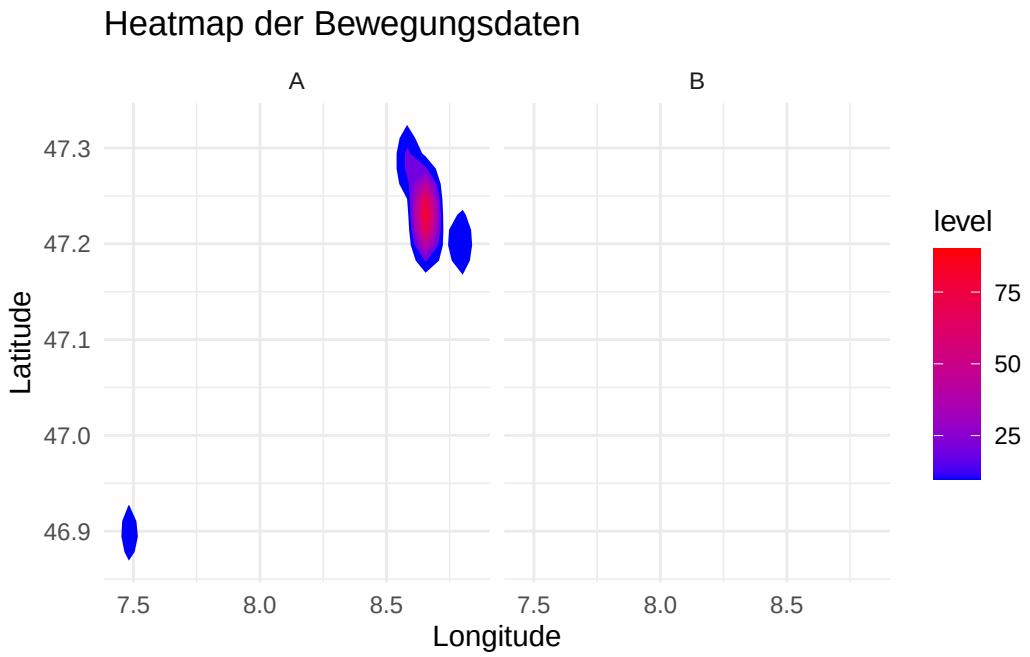
#Heatmap

```
names(records_sf)
```

```
 [1] "latitudeE7"         "longitudeE7"        "accuracy"
 [4] "source"             "deviceTag"          "deviceDesignation"
 [7] "timestamp"          "activity"           "altitude"
[10] "verticalAccuracy"   "platformType"       "osLevel"
[13] "serverTimestamp"    "deviceTimestamp"    "batteryCharging"
[16] "formFactor"         "velocity"           "heading"
[19] "locationMetadata"   "inferredLocation"   "placeId"
```

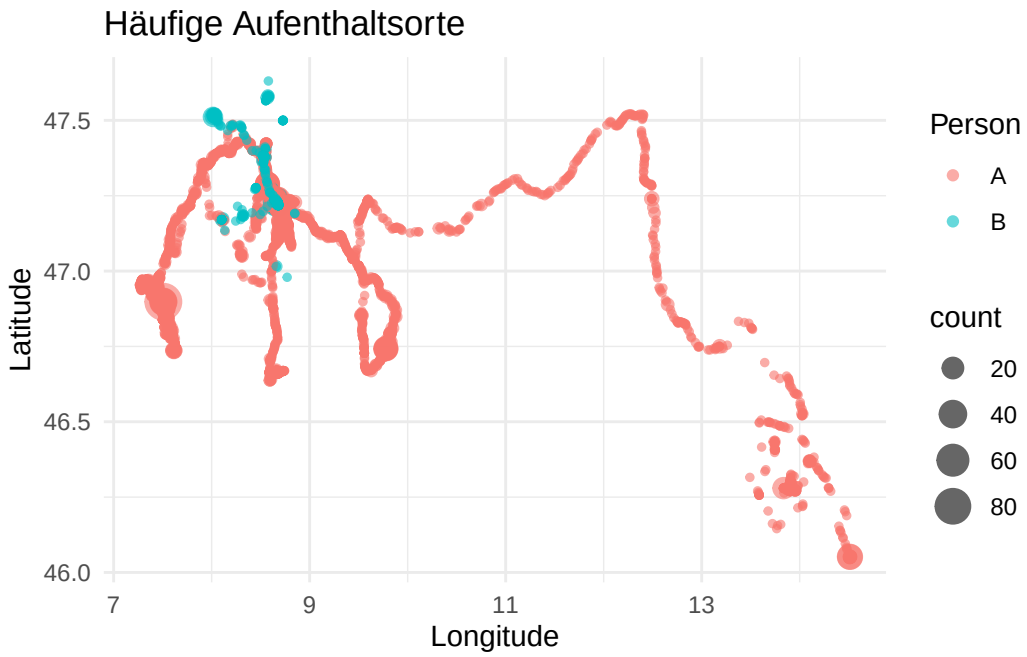
```
[22] "activeWifiScan"    "Person"            "Year"
[25] "lat_copy"          "lon_copy"          "geometry"
[28] "speed"
```

```
# Erstellen einer Heatmap der Bewegungsdaten
# heatmap_movement <-
  ggplot(records_sf, aes(x = lon_copy , y = lat_copy )) +
  stat_density2d(aes(fill = after_stat(level)), geom = "polygon") +
  scale_fill_gradient(low = "blue", high = "red") +
  labs(title = "Heatmap der Bewegungsdaten", x = "Longitude", y = "Latitude") +
  theme_minimal() +
    facet_grid( . ~ Person)
```




```
# Drucken der Heatmap
# print(heatmap_movement)



# Summarisieren von häufigen Aufenthaltsorten basierend auf deinen vorhandenen Koordinaten
  # install.packages("data.table")
  library(data.table)
```

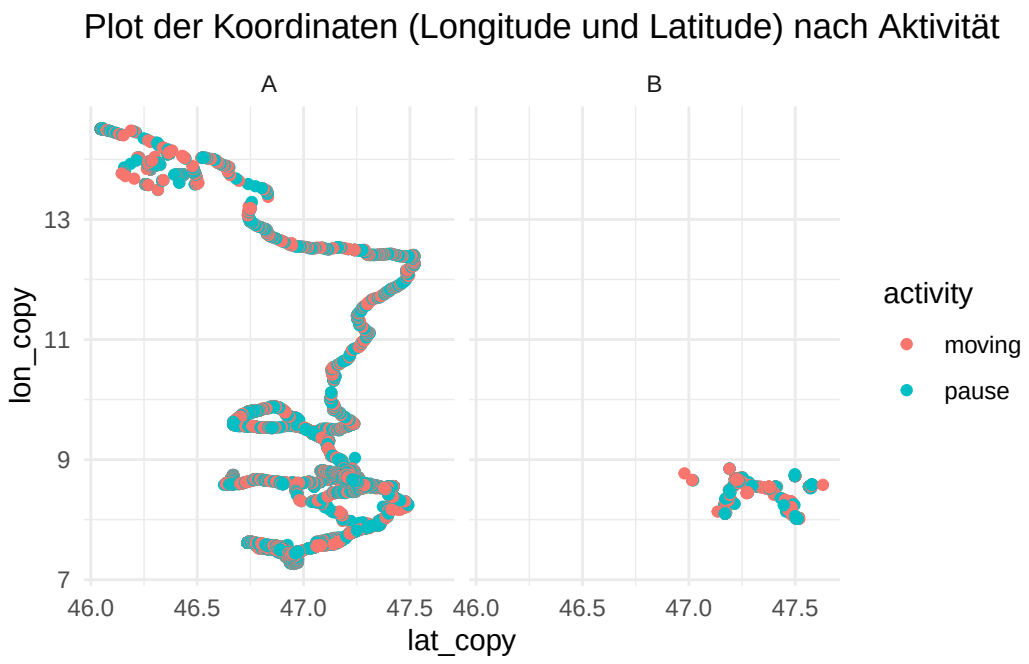
```r
setDT(records_sf)

frequent_locations <- records_sf[activity == "moving",
                                 .(count = .N),
                                 by = .(Person, lon_copy, lat_copy)][order(-count)]


# frequent_locations <-
#   records_sf %>%
#   filter(activity == "moving") |>
#   group_by(Person, lon_copy, lat_copy) %>%
#   summarise(count = n(), .groups = 'drop') %>%
#   arrange(desc(count))

# Visualisierung der häufigen Aufenthaltsorte
# frequent_locations_plot <-
  ggplot(frequent_locations, aes(x = lon_copy, y = lat_copy, size = count, color = Person)) +
  geom_point(alpha = 0.6) +
  labs(title = "Häufige Aufenthaltsorte", x = "Longitude", y = "Latitude") +
  theme_minimal()
```



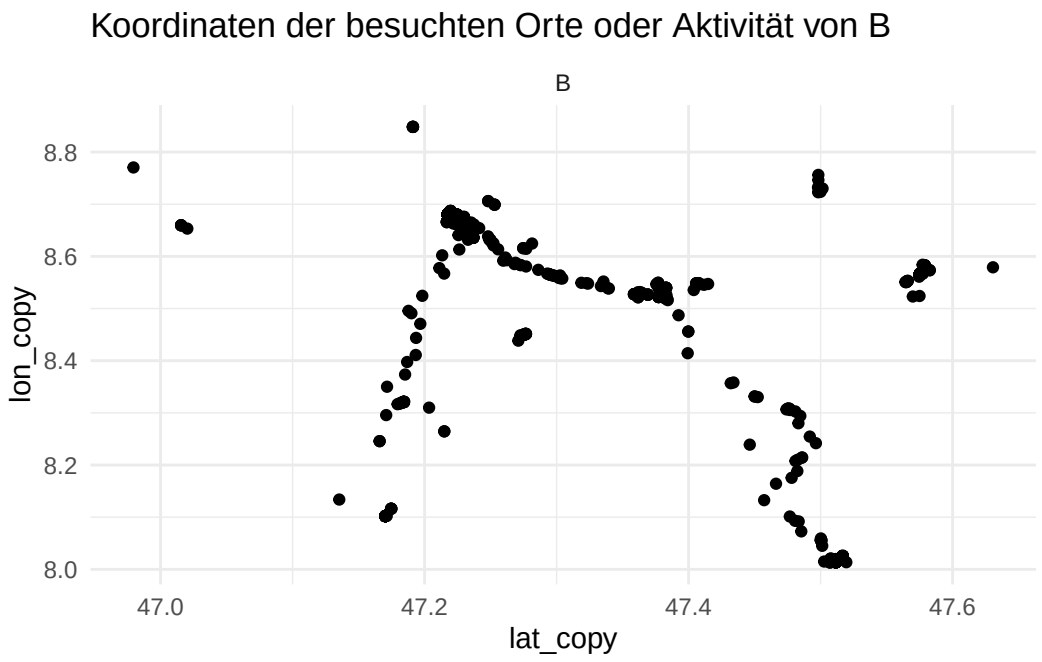
```
# # Drucken des Plots
# print(frequent_locations_plot)
#
#
# # Print the plot
# print(frequent_locations_plot)
#
#
# # Print the heatmap
# print(heatmap_movement)
```

#Plots

```
 ##  Koordinaten ##
# Plot der Longitude und Latitude
ggplot(records_sf, aes(x = lat_copy, y = lon_copy, color = activity)) +
  geom_point() +
  labs(title = "Plot der Koordinaten (Longitude und Latitude) nach Aktivität") +
  theme_minimal() +
  facet_grid( . ~ Person )
```

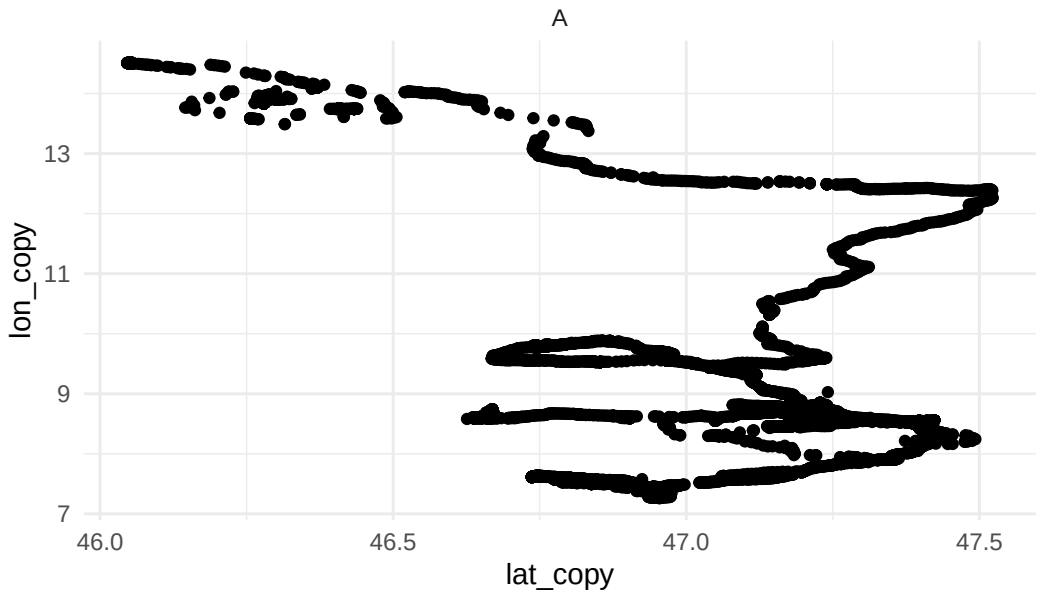


```r
# Plot der Longitude und Latitude

#B
records_sf |>
  filter(Person == "B") |>
ggplot(aes(x = lat_copy, y = lon_copy)) +
  geom_point() +
  labs(title = "Koordinaten der besuchten Orte oder Aktivität von B") +
  theme_minimal() +
  facet_grid( . ~ Person )
```

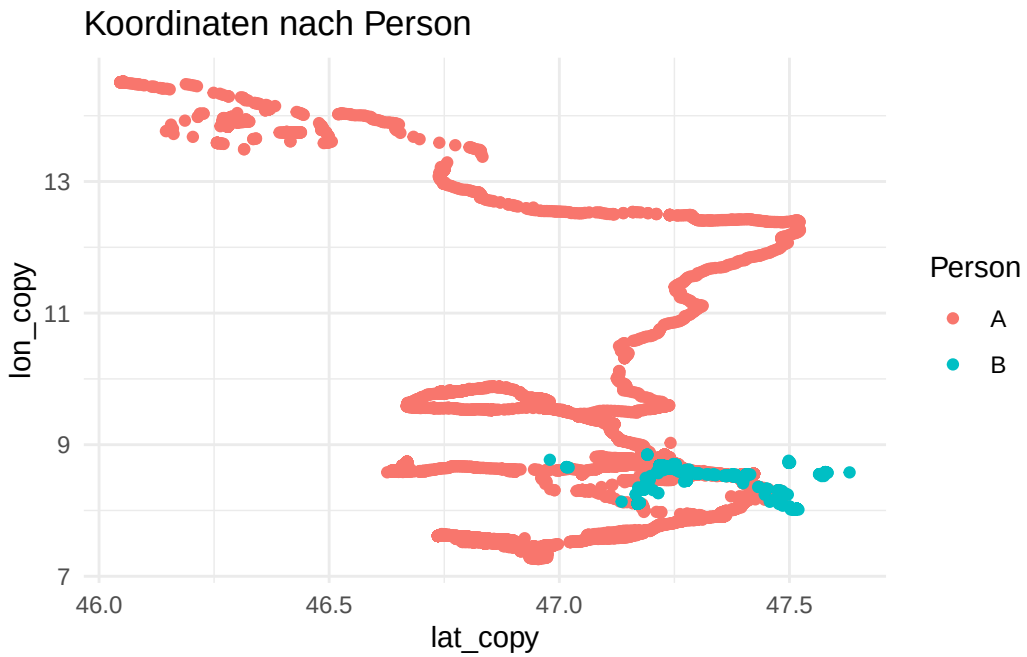



```r
#A
records_sf |>
  filter(Person == "A") |>
ggplot(aes(x = lat_copy, y = lon_copy)) +
  geom_point() +
  labs(title = "Koordinaten der besuchten Orte oder Aktivität von A") +
  theme_minimal() +
  facet_grid( . ~ Person )
```

Koordinaten der besuchten Orte oder Aktivität von A



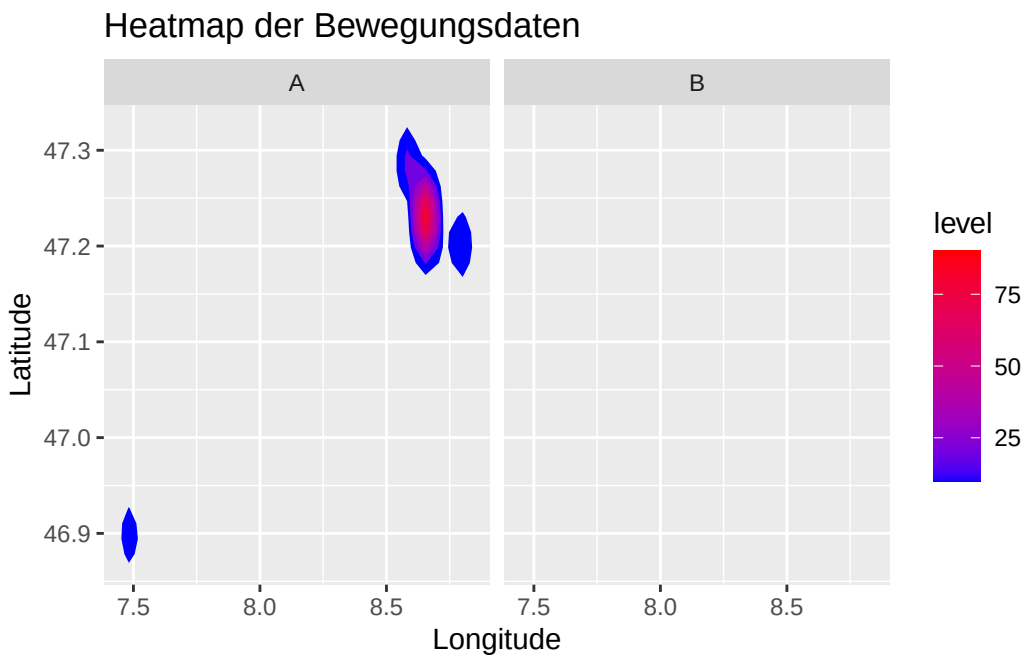

```
# Plot der Longitude und Latitude
ggplot(records_sf, aes(x = lat_copy, y = lon_copy, color = Person)) +
  geom_point() +
  labs(title = "Koordinaten nach Person") +
  theme_minimal()
```



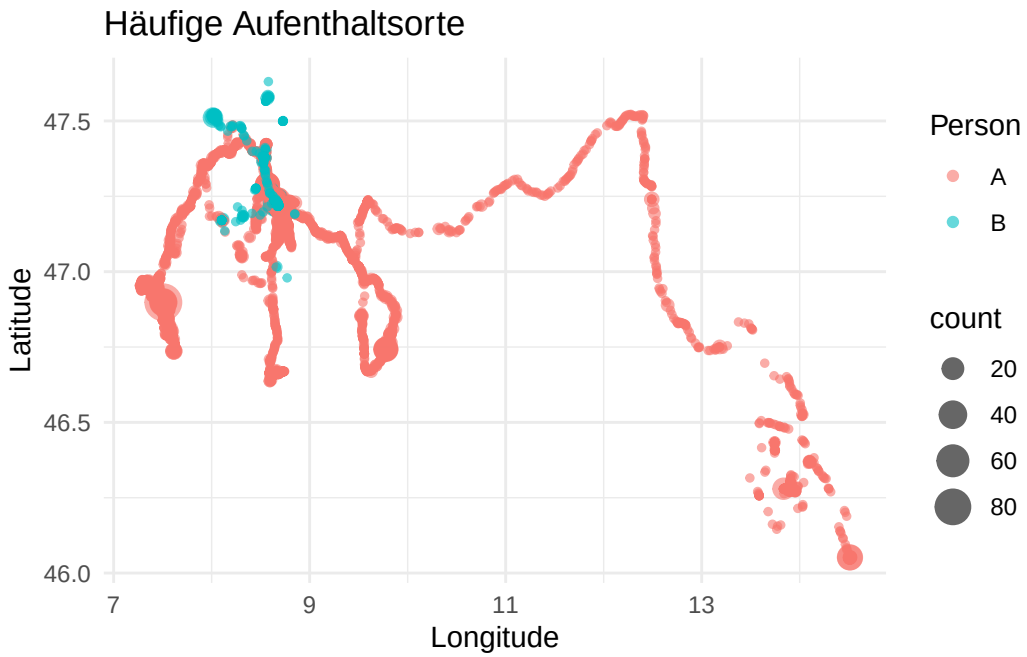

```
 ##  Zeitlich ##

# #mit Geschwindigkeit
# # # Plot der Longitude und Latitude
# ggplot(records_sf, aes(x = day, y = Speed_kmh, color = Person )) +
#   geom_point() +
#   labs(title = "Erreichte Geschwindigkeiten der Personen") +
#   theme_minimal()
#
# ggplot(records_sf, aes(x = day, y = Speed_kmh, color = Person )) +
#   geom_boxplot() +
#   labs(title = "Erreichte Geschwindigkeiten der Personen") +
#   theme_minimal()
#
#
# ggplot(records_sf, aes(x = Person, y = Speed_kmh)) +
#   geom_boxplot() +
#   labs(title = "Erreichte Geschwindigkeiten der Personen über Untersuchungszeitraum") +
#   theme_minimal()
```

```
### Plot mit x-Achse Zeit, y-Achse Koordinate (Long, Lat) und color = Person ###
#wie kann man Longitude und Latitude auf der y-Achse darstellen?




## Anzahl Aktivitäten


# ggplot(records_sf, aes(x = lat_copy, y = lon_copy, color = Person)) +
#   geom_point() +
#   geom_text(aes(label = Placemark), vjust = -1, size = 3) +  # Punkte mit Text aus der Spal
#   labs(title = "Plot der Koordinaten (Longitude und Latitude) nach Person") +
#   theme_minimal()




# Add plotting for Speed Over Time where movement occurred, highlighting pauses
# moving_speed_plot_with_pauses <-

# ggplot() +
#   geom_line(data = filter(records_sf, activity == "moving"),
#             aes(x = time, y = speed, color = activity)) +
#   geom_point(data = filter(records_sf, activity == "pause"),
#              aes(x = time, y = speed, color = activity)) +
#   scale_color_manual(values = c("moving" = "blue", "pause" = "red")) +
#   labs(title = "Speed Over Time (Movement and Pauses)", x = "Time", y = "Speed (km/h)") +
#   theme_minimal() +
#   facet_grid(  Person  ~ .)




#Hotspot
ggplot(records_sf, aes(x = lon_copy, y = lat_copy)) +
  stat_density2d(aes(fill = ..level..), geom = "polygon") +
  scale_fill_gradient(low = "blue", high = "red") +
  labs(title = "Heatmap der Bewegungsdaten", x = "Longitude", y = "Latitude") +
```

```
  facet_grid(  .  ~ Person)
```




```
#
#
# # Häufige Aufenthaltsorte
# frequent_locations <- records_sf %>%
#   group_by(Person, lon_copy, lat_copy) %>%
#   summarise(count = n()) %>%
#   arrange(desc(count))
#
#
#

#dieserplot
# Visualisierung der häufigen Aufenthaltsorte
ggplot(frequent_locations, aes(x = lon_copy, y = lat_copy, size = count, color = Person)) +
  geom_point(alpha = 0.6) +
  labs(title = "Häufige Aufenthaltsorte", x = "Longitude", y = "Latitude") +
  theme_minimal()
```



#Meeting_Points

```
# # Define the file path for the CSV file
# csv_path <- "C:/Users/A/OneDrive - ZHAW/General/Projektarbeit/R/PatTre/Data/csv_data_moving
#
# # Read the CSV file
# csv_data <- read_csv(csv_path)

library(sf)
library(readr)
library(ggmap)
library(ggplot2)
library(osmdata)
library(dplyr)
library(grid)
library(lubridate)

# Convert the time column to POSIXct
records_sf_Timepoint <- records_sf %>%
  mutate(Time_Point = as.POSIXct(deviceTimestamp, format = "%Y-%m-%dT%H:%M:%OS", tz = "UTC"))

str(records_sf_Timepoint$Time_Point)
```

```
 POSIXct[1:53086], format: "2024-01-03 07:38:28" "2024-01-03 07:38:28" "2024-01-03 07:38:28"
```

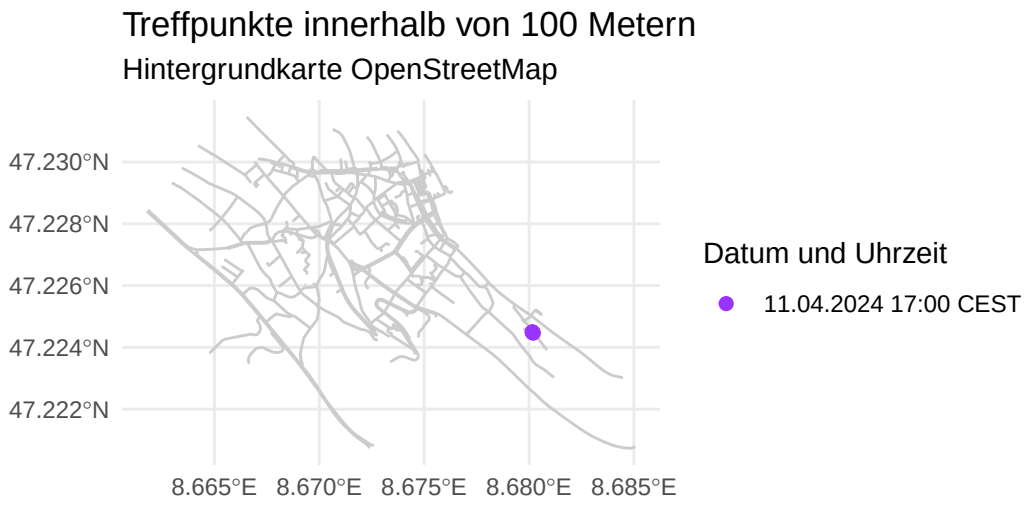
```r
# Round the time to the nearest 15 minutes
records_sf_Timepoint_rounded <- records_sf_Timepoint %>%
  mutate(Time_Rounded = round_date(Time_Point, unit = "15 minutes"))

# Split data into separate data frames for each person
B_data <- records_sf_Timepoint_rounded %>% filter(Person == "B") |>   as.data.frame()
A_data <- records_sf_Timepoint_rounded %>% filter(Person == "A") |> as.data.frame()

# Join the datasets by the rounded time
joined_data <- inner_join(B_data, A_data, by = "Time_Rounded", suffix = c(".B", ".A"))


names(joined_data)
```

```
 [1] "latitudeE7.B"        "longitudeE7.B"       "accuracy.B"
 [4] "source.B"            "deviceTag.B"         "deviceDesignation.B"
 [7] "timestamp.B"         "activity.B"          "altitude.B"
[10] "verticalAccuracy.B"  "platformType.B"      "osLevel.B"
[13] "serverTimestamp.B"   "deviceTimestamp.B"   "batteryCharging.B"
[16] "formFactor.B"        "velocity.B"          "heading.B"
[19] "locationMetadata.B"  "inferredLocation.B"  "placeId.B"
[22] "activeWifiScan.B"    "Person.B"            "Year.B"
[25] "lat_copy.B"          "lon_copy.B"          "geometry.B"
[28] "speed.B"             "Time_Point.B"        "Time_Rounded"
[31] "latitudeE7.A"        "longitudeE7.A"       "accuracy.A"
[34] "source.A"            "deviceTag.A"         "deviceDesignation.A"
[37] "timestamp.A"         "activity.A"          "altitude.A"
[40] "verticalAccuracy.A"  "platformType.A"      "osLevel.A"
[43] "serverTimestamp.A"   "deviceTimestamp.A"   "batteryCharging.A"
[46] "formFactor.A"        "velocity.A"          "heading.A"
[49] "locationMetadata.A"  "inferredLocation.A"  "placeId.A"
[52] "activeWifiScan.A"    "Person.A"            "Year.A"
[55] "lat_copy.A"          "lon_copy.A"          "geometry.A"
[58] "speed.A"             "Time_Point.A"
```

```r
# Calculate Euclidean distances between concurrent observations
joined_data <- joined_data %>%
  mutate(Distance = sqrt((lon_copy.B - lon_copy.A)^2 +
                           (lat_copy.B - lat_copy.A)^2))
```

```r
# Radius der Erde in Kilometern
earth_radius <- 6371

# Berechnung der Haversine-Distanz um von Grad auf Kilometer zu kommen
joined_data <- joined_data %>%
  mutate(
    Distance_km = 2 * earth_radius * asin(sqrt(
      sin((lat_copy.A - lat_copy.B) * pi / 180 / 2)^2 +
      cos(lat_copy.B * pi / 180) * cos(lat_copy.A * pi / 180) *
      sin((lon_copy.A - lon_copy.B) * pi / 180 / 2)^2
    ))
  )


### Meet als Radius von 250 Metern ###
# Determine if the persons are spatially close enough to constitute a meet (within 250 meters
joined_data <- joined_data %>%
  mutate(Meet = Distance_km <= 0.075)

# Filter the joined dataset to only include meets
meets_data <- joined_data %>% filter(Meet == TRUE)

# Create a table with the meeting points, coordinates, and times
meeting_table <- meets_data %>%
  mutate(Meeting_Longitude = (lon_copy.B + lon_copy.A) / 2,
         Meeting_Latitude = (lat_copy.B + lat_copy.B) / 2) %>%
  select(Time_Rounded, Meeting_Longitude, Meeting_Latitude)

# Convert the time column to POSIXct
records_sf_Timepoint <- records_sf %>%
  mutate(Time_Point = as.POSIXct(deviceTimestamp, format = "%Y-%m-%dT%H:%M:%OS", tz = "UTC"))

# Convert POSIXct to Zurich time zone and format for the legend
meeting_table$Formatted_Time <- meeting_table$Time_Rounded %>%
  with_tz("Europe/Zurich") %>%
  format("%d.%m.%Y %H:%M %Z")

# Get bounding box for the plot area based on meeting points
bbox <- st_bbox(c(xmin = min(meeting_table$Meeting_Longitude),
                  xmax = max(meeting_table$Meeting_Longitude),
                  ymin = min(meeting_table$Meeting_Latitude),
                  ymax = max(meeting_table$Meeting_Latitude)), crs = st_crs(4326))
```

```r
# Fetch OpenStreetMap data
osm_map <- opq(bbox = bbox) %>%
  add_osm_feature(key = "highway") %>%
  osmdata_sf()

#Ausreisser entfernen
meeting_table_filtered <- meeting_table %>%
  filter(Formatted_Time != "08.04.2024 08:15 CEST")

# Convert meeting_table to an sf object
sf_meeting_table <- st_as_sf(meeting_table_filtered, coords = c("Meeting_Longitude", "Meeting
final_plot <- ggplot() +
  geom_sf(data = osm_map$osm_lines, color = "grey80", size = 0.3) +
  geom_sf(data = sf_meeting_table, aes(color = Formatted_Time), size = 2) +
  scale_color_manual(values = c("#9933FF", "#228B22", "#9933FF", "#9933FF"),
                     name = "Treffzeitpunkt",
                     guide = guide_legend(title = "Datum und Uhrzeit")) +
  labs(title = "Treffpunkte innerhalb von 100 Metern",
       subtitle = "Hintergrundkarte OpenStreetMap") +
  theme_minimal() +
  theme(legend.position = "right")

# Print the final plot
print(final_plot)
```



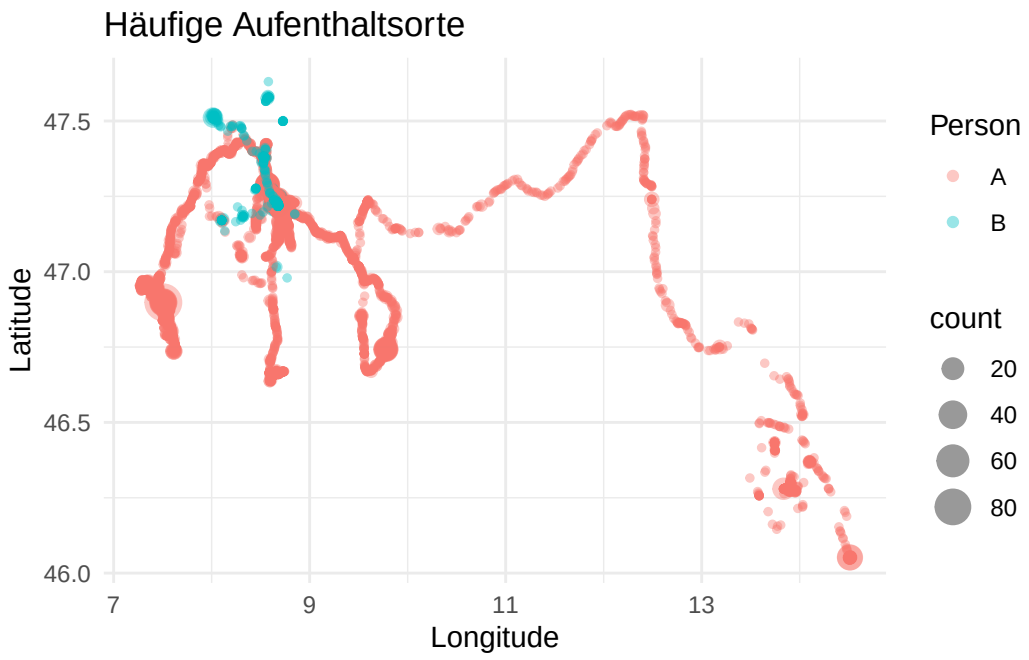
#Plot Vergleich zwischen Personen

```
frequent_locations
```

```
       Person lon_copy lat_copy count
       <char>    <num>    <num> <int>
    1:      A 7.506927 46.89821    87
    2:      A 8.664210 47.23103    45
    3:      A 7.506990 46.89821    39
    4:      A 8.664309 47.23090    38
    5:      A 7.507190 46.89817    36
   ---
20606:      B 8.663929 47.22479     1
20607:      B 8.663781 47.22490     1
20608:      B 8.663828 47.22473     1
20609:      B 8.666793 47.22562     1
20610:      B 8.664004 47.22463     1
```

```
# Visualisierung der häufigen Aufenthaltsorte
# frequent_locations_plot <-
  ggplot(frequent_locations, aes(x = lon_copy, y = lat_copy, size = count, color = Person)) +
  geom_point(alpha = 0.4) +
```

```
  labs(title = "Häufige Aufenthaltsorte", x = "Longitude", y = "Latitude") +
  theme_minimal()
```

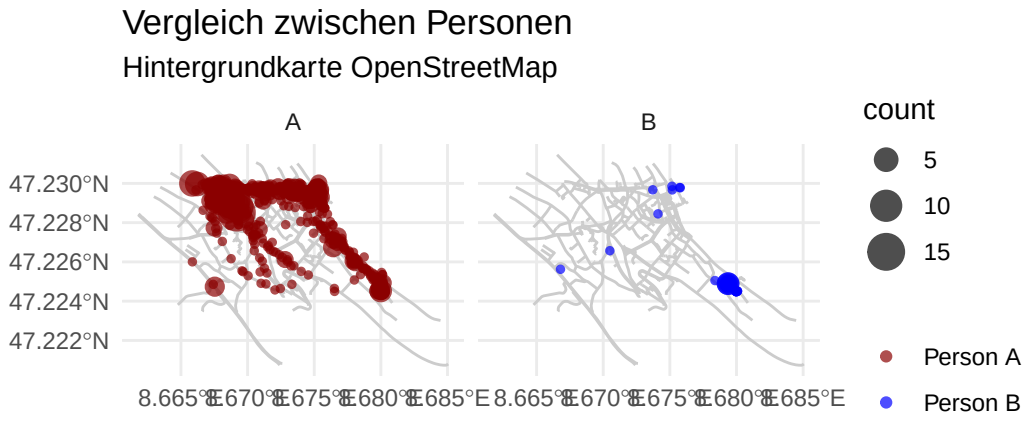



```
bbox
```

```
     xmin      ymin      xmax      ymax
 8.665727 47.224463  8.680202 47.230027
```

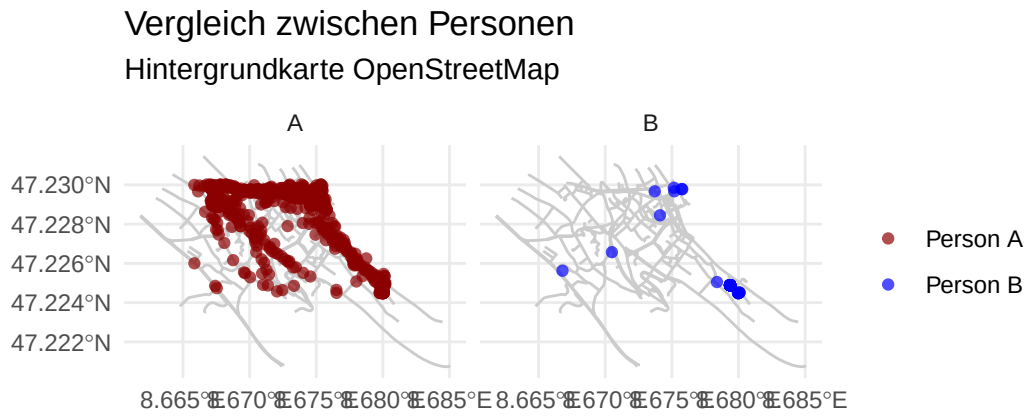
```
# Daten in ein sf-Objekt umwandeln
frequent_locations_sf <- st_as_sf(frequent_locations, coords = c("lon_copy", "lat_copy"), crs

# Daten anhand der Bounding Box zuschneiden
frequent_locations_cropped <- st_crop(frequent_locations_sf, bbox)


# Plot erstellen
ggplot() +
  geom_sf(data = osm_map$osm_lines, color = "grey80", size = 0.3) +
  geom_sf(data = frequent_locations_cropped, aes(size = count, color = Person), alpha = 0.7)
  scale_color_manual(values = c("A" = "darkred", "B" = "blue"),
                     name = "Person",
```

```
                      labels = c("A" = "Person A", "B" = "Person B"),
                      guide = guide_legend(title = NULL)) +
  labs(title = "Vergleich zwischen Personen",
       subtitle = "Hintergrundkarte OpenStreetMap") +
  theme_minimal() +
  theme(legend.position = "right") +
  facet_grid(. ~ Person)
```



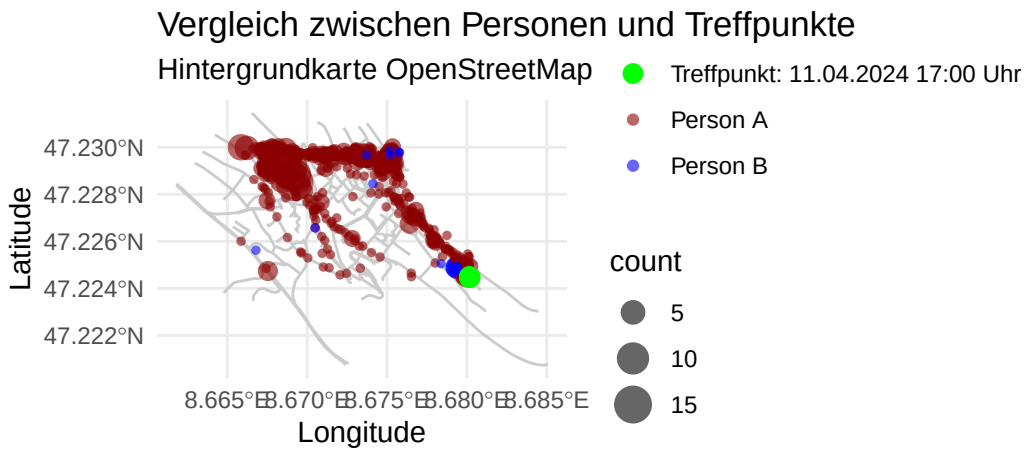

```
frequent_locations_cropped_n <- frequent_locations_cropped |> filter(Person == "B")


ggplot() +
  geom_sf(data = osm_map$osm_lines, color = "grey80", size = 0.3) +
  geom_sf(data = frequent_locations_cropped, aes(color = Person), alpha = 0.7) +
  scale_color_manual(values = c("A" = "darkred", "B" = "blue"),
                      name = NULL,
                      labels = c("A" = "Person A", "B" = "Person B")) +
  scale_size_continuous(name = "Count") +
  labs(title = "Vergleich zwischen Personen",
       subtitle = "Hintergrundkarte OpenStreetMap") +
  theme_minimal() +
  theme(legend.position = "right") +
```

```
  guides(color = guide_legend(order = 1),
         size = guide_legend(order = 2)) +
  facet_grid(. ~ Person)
```




```
final_plot <- ggplot() +
  geom_sf(data = osm_map$osm_lines, color = "grey80", size = 0.3) +
  geom_sf(data = sf_meeting_table, aes(color = Formatted_Time), size = 2) +
  scale_color_manual(values = c("#9933FF", "#228B22", "#9933FF", "#9933FF"),
                     name = "Treffzeitpunkt",
                     guide = guide_legend(title = "Datum und Uhrzeit")) +
  labs(title = "Treffpunkte innerhalb von 100 Metern",
       subtitle = "Hintergrundkarte OpenStreetMap") +
  theme_minimal() +
  theme(legend.position = "right")



# Kombinierter Plot erstellen
ggplot() +
  geom_sf(data = osm_map$osm_lines, color = "grey80", size = 0.3) +
  geom_sf(data = frequent_locations_cropped, aes(size = count, color = Person), alpha = 0.6)
  geom_sf(data = sf_meeting_table, aes(color = Formatted_Time), size = 3) +
```

```
  scale_color_manual(values = c("A" = "darkred", "B" = "blue", "2022-01-01 12:00" = "#D2FF33"
                       labels = c("A" = "Person A", "B" = "Person B", "08.04.2024 08:15 CEST"
                      guide = guide_legend(order = 1,title = NULL)) +
  scale_shape_manual(values = c(Person = 16, Formatted_Time = 17), guide = guide_legend(order
  labs(title = "Vergleich zwischen Personen und Treffpunkte",
       subtitle = "Hintergrundkarte OpenStreetMap",
       x = "Longitude",
       y = "Latitude") +
  theme_minimal() +
  theme(legend.position = "right")
```



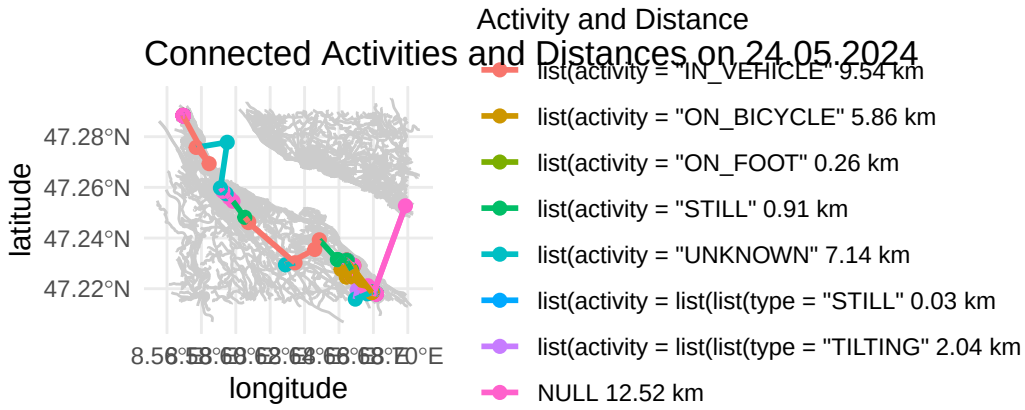

```
# ggplot() +
#   geom_sf(data = osm_map$osm_lines, color = "grey80", size = 0.3) +
#   geom_sf(data = frequent_locations_cropped, aes(size = count, color = Person, shape = Type
#   geom_sf(data = sf_meeting_table, aes(color = Formatted_Time, shape = Type), size = 3) +
#   scale_color_manual(values = c("A" = "#FF5733", "B" = "#0066FF", "2022-01-01 12:00" = "#D2
#                       labels = c("A" = "Person A", "B" = "Person B", "08.04.2024 08:15 CEST"
#                       guide = guide_legend(order = 1, title = NULL)) +
#   scale_shape_manual(values = c("Person" = 16, "Treffpunkt" = 17), guide = guide_legend(orde
#   labs(title = "Vergleich zwischen Personen und Treffpunkte",
#        subtitle = "Hintergrundkarte OpenStreetMap",
#        x = "Longitude",
```

```
#         y = "Latitude") +
#   theme_minimal() +
#   theme(legend.position = "right")
```

Connected acitivities and Distances on a Day > Map

```
# Load necessary libraries
library(dplyr)
library(tidyr)
library(ggplot2)
library(sf)
library(ggspatial)
library(scales)
library(osmdata)
library(geosphere)

# Assuming you have loaded the data into records_N dataframe
# records_N <- read.csv("path_to_your_data.csv") # Adjust the path accordingly

# Filter records by timestamp for the specific date
records_filtered <- records_N %>%
  filter(grepl("2024-05-24", timestamp))

# Parse the activity column
records_filtered <- records_filtered %>%
  mutate(activity = gsub("list\\(list\\(type = c\\(|\\), confidence = c\\(|\\)\\)\\)", "", ac
  separate(activity, into = c("activity1", "activity2"), sep = ", confidence = ") %>%
  separate(activity1, into = c("activity1", "activity1_confidence"), sep = ", ") %>%
  separate(activity2, into = c("activity2", "activity2_confidence"), sep = ", ")

# Convert lat/lon from E7 to decimal degrees
records_filtered <- records_filtered %>%
  mutate(latitude = latitudeE7 / 1e7,
         longitude = longitudeE7 / 1e7)

# Remove activities "Null" and "Still"
records_filtered <- records_filtered %>%
  filter(!activity1 %in% c("Null", "Still"))
```

```r
# Convert to spatial dataframe
sf_records_filtered <- st_as_sf(records_filtered, coords = c("longitude", "latitude"), crs =

# Calculate distances between consecutive points
records_filtered <- records_filtered %>%
  arrange(timestamp) %>%
  mutate(dist = distHaversine(cbind(longitude, latitude), cbind(lag(longitude), lag(latitude
  replace_na(list(dist = 0))  # Replace NA distances with 0

# Summarize distance per activity
activity_distances <- records_filtered %>%
  group_by(activity1) %>%
  summarize(total_distance = sum(dist, na.rm = TRUE))

# Create labels for the legend
records_filtered <- records_filtered %>%
  left_join(activity_distances, by = "activity1") %>%
  mutate(legend_label = paste(activity1, round(total_distance, 2), "km"))

# Get bounding box for the plot area
bbox <- st_bbox(sf_records_filtered)

# Get OpenStreetMap data
osm_map <- opq(bbox = c(bbox["xmin"], bbox["ymin"], bbox["xmax"], bbox["ymax"])) %>%
  add_osm_feature(key = "highway") %>%
  osmdata_sf()

# Plot using ggplot2
ggplot() +
  geom_sf(data = osm_map$osm_lines, color = "grey80", size = 0.3) + # Use osm_lines for a mo
  geom_point(data = records_filtered, aes(x = longitude, y = latitude, color = legend_label)
  geom_path(data = records_filtered, aes(x = longitude, y = latitude, group = 1, color = leg
  theme_minimal() +
  labs(title = "Connected Activities and Distances on 24.05.2024",
       fill = "Activity and Distance",
       color = "Activity and Distance")
```



Connected acitivities and Distances on a Month > Map mit Hintergrund CH shp

```
# # QUELLE HINTERGRUNDKARTE:
# # https://www.swisstopo.admin.ch/de/landschaftsmodell-swissboundaries3d#swissBOUNDARIES3D--
#
#
# # Load necessary libraries
# library(dplyr)
# library(tidyr)
# library(ggplot2)
# library(sf)
# library(geosphere)
#
# # Set SHAPE_RESTORE_SHX config option
# Sys.setenv(SHAPE_RESTORE_SHX = "YES")
#
# # Load the Swiss boundaries shapefile
# swiss_boundaries <- st_read("C:/Users/A/OneDrive - ZHAW/General/Projektarbeit/Ausgangsdater
#
```

```
# # Ensure geometries are valid
# swiss_boundaries <- st_make_valid(swiss_boundaries)
#
# # Check the CRS of the shapefile
# print(st_crs(swiss_boundaries))
#
# # Assuming you have loaded the data into records_N dataframe
# # records_N <- read.csv("path_to_your_data.csv") # Adjust the path accordingly
#
# # Filter records by timestamp for the entire month of April 2024
# records_filtered <- records_N %>%
#   filter(grepl("2024-04", timestamp))
#
# # Parse the activity column
# records_filtered <- records_filtered %>%
#   mutate(activity = gsub("list\\(list\\(type = c\\(|\\), confidence = c\\(|\\)\\)\\)", "",
#   separate(activity, into = c("activity1", "activity2"), sep = ", confidence = ") %>%
#   separate(activity1, into = c("activity1", "activity1_confidence"), sep = ", ") %>%
#   separate(activity2, into = c("activity2", "activity2_confidence"), sep = ", ")
#
# # Convert lat/lon from E7 to decimal degrees
# records_filtered <- records_filtered %>%
#   mutate(latitude = latitudeE7 / 1e7,
#          longitude = longitudeE7 / 1e7)
#
# # Remove activities "Null" and "Still"
# records_filtered <- records_filtered %>%
#   filter(!activity1 %in% c("Null", "Still"))
#
# # Convert to spatial dataframe
# sf_records_filtered <- st_as_sf(records_filtered, coords = c("longitude", "latitude"), crs
#
# # Transform the CRS of the swiss boundaries to match the records
# swiss_boundaries <- st_transform(swiss_boundaries, crs = st_crs(sf_records_filtered))
#
# # Calculate distances between consecutive points
# records_filtered <- records_filtered %>%
#   arrange(timestamp) %>%
#   mutate(dist = distHaversine(cbind(longitude, latitude), cbind(lag(longitude), lag(latitud
#   replace_na(list(dist = 0))  # Replace NA distances with 0
#
# # Summarize distance per activity
```

```
# activity_distances <- records_filtered %>%
#   group_by(activity1) %>%
#   summarize(total_distance = sum(dist, na.rm = TRUE))
#
# # Create labels for the legend
# records_filtered <- records_filtered %>%
#   left_join(activity_distances, by = "activity1") %>%
#   mutate(legend_label = paste(activity1, round(total_distance, 2), "km"))
#
# # Plot using ggplot2
# ggplot() +
#   geom_sf(data = swiss_boundaries, fill = NA, color = "grey50") +
#   geom_point(data = records_filtered, aes(x = longitude, y = latitude, color = legend_label
#   geom_path(data = records_filtered, aes(x = longitude, y = latitude, group = 1, color = le
#   theme_minimal() +
#   labs(title = "Connected Activities and Distances in April 2024",
#        fill = "Activity and Distance",
#        color = "Activity and Distance")
```

Connected acitivities and Distances 2023 > Map

```
#
# # QUELLE Hintergrundkarte: https://data.dtu.dk/articles/dataset/Shapefile_of_European_count
# # Verwendung REFNET für Transformation
#
# #Load necessary libraries
# library(dplyr)
# library(tidyr)
# library(ggplot2)
# library(sf)
# library(geosphere)
#
# # Set SHAPE_RESTORE_SHX config option
# Sys.setenv(SHAPE_RESTORE_SHX = "YES")
#
# # Load the Europe boundaries shapefile
# europe_boundaries <- st_read("C:/Users/A/OneDrive - ZHAW/General/Projektarbeit/Ausgangsdat
#
# # Ensure geometries are valid
```

```
# europe_boundaries <- st_make_valid(europe_boundaries)
#
# # Check the CRS of the shapefile
# print(st_crs(europe_boundaries))
#
# # Assuming you have loaded the data into records_N dataframe
# # records_N <- read.csv("path_to_your_data.csv") # Adjust the path accordingly
#
# # Filter records by timestamp for the entire month of April 2024
# records_filtered <- records_N %>%
#   filter(grepl("2023", timestamp))
#
# # Parse the activity column
# records_filtered <- records_filtered %>%
#   mutate(activity = gsub("list\\(list\\(type = c\\(|\\), confidence = c\\(|\\)\\)\\)", "",
#   separate(activity, into = c("activity1", "activity2"), sep = ", confidence = ") %>%
#   separate(activity1, into = c("activity1", "activity1_confidence"), sep = ", ") %>%
#   separate(activity2, into = c("activity2", "activity2_confidence"), sep = ", ")
#
# # Convert lat/lon from E7 to decimal degrees
# records_filtered <- records_filtered %>%
#   mutate(latitude = latitudeE7 / 1e7,
#          longitude = longitudeE7 / 1e7)
#
# # Remove activities "Null" and "Still"
# records_filtered <- records_filtered %>%
#   filter(!activity1 %in% c("Null", "Still"))
#
# # Convert to spatial dataframe
# sf_records_filtered <- st_as_sf(records_filtered, coords = c("longitude", "latitude"), crs
#
# # Transform the CRS of the Europe boundaries to match the records
# europe_boundaries <- st_transform(europe_boundaries, crs = st_crs(sf_records_filtered))
#
# # Calculate distances between consecutive points
# records_filtered <- records_filtered %>%
#   arrange(timestamp) %>%
#   mutate(dist = distHaversine(cbind(longitude, latitude), cbind(lag(longitude), lag(latitu
#   replace_na(list(dist = 0))  # Replace NA distances with 0
#
# # Summarize distance per activity
# activity_distances <- records_filtered %>%
```

```
#   group_by(activity1) %>%
#   summarize(total_distance = sum(dist, na.rm = TRUE))
#
# # Create labels for the legend
# records_filtered <- records_filtered %>%
#   left_join(activity_distances, by = "activity1") %>%
#   mutate(legend_label = paste(activity1, round(total_distance, 2), "km"))
#
# # Plot using ggplot2
# ggplot() +
#   geom_sf(data = europe_boundaries, fill = NA, color = "grey50") +
#   geom_point(data = records_filtered, aes(x = longitude, y = latitude, color = legend_label
#   geom_path(data = records_filtered, aes(x = longitude, y = latitude, group = 1, color = le
#   theme_minimal() +
#   labs(title = "Connected Activities and Distances 2023",
#        fill = "Activity and Distance",
#        color = "Activity and Distance")
```

Zusammenstellung Karte Europaweit

```
#
# # Load necessary libraries
# library(dplyr)
# library(tidyr)
# library(ggplot2)
# library(sf)
# library(scales)
# library(geosphere)
# library(lubridate)
# library(gridExtra)
# library(GGally)
# library(RColorBrewer)
# library(cluster)
# library(factoextra)
#
# # Load the Europe boundaries shapefile
# europe_boundaries <- st_read("C:/_Data/Master/PaT_24/Projectwork/RProjectwork/cma-project/
#
# # Ensure geometries are valid
```

```
# europe_boundaries <- st_make_valid(europe_boundaries)
#
# # Check the CRS of the shapefile
# print(st_crs(europe_boundaries))
#
# # Hilfsfunktion zur Klassifizierung der Tageszeit
# time_of_day <- function(hour) {
#   case_when(
#     hour >= 6 & hour < 10 ~ "Morgen (06-10 Uhr)",
#     hour >= 10 & hour < 14 ~ "Mittag (10-14 Uhr)",
#     hour >= 14 & hour < 18 ~ "Nachmittag (14-18 Uhr)",
#     hour >= 18 & hour < 22 ~ "Abend (18-22 Uhr)",
#     TRUE ~ "Nacht (22-06 Uhr)"
#   )
# }
#
# process_year_data <- function(records_filtered, year_label) {
#   # Filter records by timestamp for the specified year
#   records_filtered <- records_filtered %>%
#     filter(grepl(year_label, timestamp))
#
#   # Parse the activity column
#   records_filtered <- records_filtered %>%
#     mutate(activity = gsub("list\\(list\\(type = c\\(|\\), confidence = c\\(|\\)\\)", "", a
#     separate(activity, into = c("activity1", "activity2"), sep = ", confidence = ") %>%
#     separate(activity1, into = c("activity1", "activity1_confidence"), sep = ", ") %>%
#     separate(activity2, into = c("activity2", "activity2_confidence"), sep = ", ")
#
#   # Convert lat/lon from E7 to decimal degrees
#   records_filtered <- records_filtered %>%
#     mutate(latitude = latitudeE7 / 1e7,
#            longitude = longitudeE7 / 1e7)
#
#   # Remove activities "Null" and "Still"
#   records_filtered <- records_filtered %>%
#     filter(!activity1 %in% c("Null", "Still"))
#
#   # Remove rows with missing latitude or longitude
#   records_filtered <- records_filtered %>%
#     filter(!is.na(latitude) & !is.na(longitude))
#
#   # Further clean the activity1 field
```

```
#   records_filtered <- records_filtered %>%
#     mutate(activity1 = gsub("list\\(activity = \"|\"\\)", "", activity1)) %>%
#     filter(activity1 != "NULL")
#
#   # Convert to spatial dataframe
#   sf_records_filtered <- st_as_sf(records_filtered, coords = c("longitude", "latitude"), cr
#
#   # Calculate distances between consecutive points
#   records_filtered <- records_filtered %>%
#     arrange(timestamp) %>%
#     mutate(
#       dist = distHaversine(cbind(longitude, latitude), cbind(lag(longitude), lag(latitude))
#       timestamp = as.POSIXct(timestamp, format = "%Y-%m-%dT%H:%M:%S", tz = "UTC"),
#       year = year(timestamp),
#       month = month(timestamp, label = TRUE, abbr = FALSE), # Monate als Namen
#       day = wday(timestamp, label = TRUE, abbr = FALSE), # Wochentage als Namen
#       hour = hour(timestamp),
#       period = time_of_day(hour)
#     ) %>%
#     replace_na(list(dist = 0))  # Replace NA distances with 0
#
#   return(records_filtered)
# }
#
# # Assuming records_N is loaded
# records_2022 <- process_year_data(records_N, "2022")
# records_2023 <- process_year_data(records_N, "2023")
# records_2024 <- process_year_data(records_N, "2024")
#
# # Combine all years
# all_records <- bind_rows(records_2022, records_2023, records_2024)
#
# # Define the order of periods and days
# all_records$period <- factor(all_records$period, levels = c("Morgen (06-10 Uhr)", "Mittag
# all_records$day <- factor(all_records$day, levels = c("Montag", "Dienstag", "Mittwoch", "Do
#
# # Prepare data for clustering
# clustering_data <- all_records %>%
#   select(longitude, latitude, activity1) %>%
#   mutate(activity1 = as.numeric(factor(activity1)))
#
# # Perform k-means clustering
```

```
# set.seed(123)
# kmeans_result <- kmeans(clustering_data[, c("longitude", "latitude")], centers = 5)
#
# # Add cluster information to the data
# all_records$cluster <- factor(kmeans_result$cluster)
#
# # Clustering nach Jahr
# clustering_data_year <- all_records %>%
#   select(longitude, latitude, year)
#
# set.seed(123)
# kmeans_result_year <- kmeans(clustering_data_year[, c("longitude", "latitude")], centers =
#
# # Add year cluster information to the data
# all_records$cluster_year <- factor(kmeans_result_year$cluster)
#
# # Visualize clusters on map with Europe boundaries
# p1 <- ggplot() +
#   geom_sf(data = europe_boundaries, fill = "lightgrey", color = "white") +
#   geom_point(data = all_records, aes(x = longitude, y = latitude, color = cluster), size =
#   scale_color_brewer(palette = "Set3", name = "Cluster der Aktivitäten") +
#   labs(
#     title = "Geografische Cluster der Aktivitäten",
#     x = "Längengrad",
#     y = "Breitengrad"
#   ) +
#   theme_minimal()
#
# # Visualize year-based clusters on map with Europe boundaries
# p2 <- ggplot() +
#   geom_sf(data = europe_boundaries, fill = "lightgrey", color = "white") +
#   geom_point(data = all_records, aes(x = longitude, y = latitude, color = cluster_year), si
#   scale_color_brewer(palette = "Set3", name = "Cluster nach Jahr") +
#   labs(
#     title = "Geografische Cluster der Aktivitäten nach Jahr",
#     x = "Längengrad",
#     y = "Breitengrad"
#   ) +
#   theme_minimal()
#
# # Heatmap der Koordinaten nach Aktivität
# p3 <- ggplot() +
```

```
#   geom_sf(data = europe_boundaries, fill = "lightgrey", color = "white") +
#   geom_tile(data = all_records, aes(x = longitude, y = latitude, fill = activity1), size =
#   scale_fill_brewer(palette = "Set3", name = "Aktivität") +
#   labs(
#     title = "Heatmap der Koordinaten nach Aktivität",
#     x = "Längengrad",
#     y = "Breitengrad"
#   ) +
#   theme_minimal()
#
# # Heatmap der Koordinaten nach Jahr
# p4 <- ggplot() +
#   geom_sf(data = europe_boundaries, fill = "lightgrey", color = "white") +
#   geom_tile(data = all_records, aes(x = longitude, y = latitude, fill = factor(year)), size
#   scale_fill_brewer(palette = "Set3", name = "Jahr") +
#   labs(
#     title = "Heatmap der Koordinaten nach Jahr",
#     x = "Längengrad",
#     y = "Breitengrad"
#   ) +
#   theme_minimal()
#
# # Arrange plots in grid
# grid.arrange(
#   p1, p2, p3, p4,
#   ncol = 1
# )
```

Heatmap und Cluster Karte Europaweit nach Aktivitäten

```
# # Load necessary libraries
# library(dplyr)
# library(tidyr)
# library(ggplot2)
# library(sf)
# library(scales)
# library(geosphere)
# library(lubridate)
# library(gridExtra)
#
```

```
#
# library(GGally)
# library(RColorBrewer)
# library(cluster)
# library(factoextra)
#
# # Load the Europe boundaries shapefile
# europe_boundaries <- st_read("C:/_Data/Master/PaT_24/Projectwork/RProjectwork/cma-project/
#
# # Ensure geometries are valid
# europe_boundaries <- st_make_valid(europe_boundaries)
#
# # Check the CRS of the shapefile
# print(st_crs(europe_boundaries))
#
# # Hilfsfunktion zur Klassifizierung der Tageszeit
# time_of_day <- function(hour) {
#   case_when(
#     hour >= 6 & hour < 10 ~ "Morgen (06-10 Uhr)",
#     hour >= 10 & hour < 14 ~ "Mittag (10-14 Uhr)",
#     hour >= 14 & hour < 18 ~ "Nachmittag (14-18 Uhr)",
#     hour >= 18 & hour < 22 ~ "Abend (18-22 Uhr)",
#     TRUE ~ "Nacht (22-06 Uhr)"
#   )
# }
#
# process_year_data <- function(records_filtered, year_label) {
#   # Filter records by timestamp for the specified year
#   records_filtered <- records_filtered %>%
#     filter(grepl(year_label, timestamp))
#
#   # Parse the activity column
#   records_filtered <- records_filtered %>%
#     mutate(activity = gsub("list\\(list\\(type = c\\(|\\), confidence = c\\(|\\)\\)\\)", "", a
#     separate(activity, into = c("activity1", "activity2"), sep = ", confidence = ") %>%
#     separate(activity1, into = c("activity1", "activity1_confidence"), sep = ", ") %>%
#     separate(activity2, into = c("activity2", "activity2_confidence"), sep = ", ")
#
#   # Convert lat/lon from E7 to decimal degrees
#   records_filtered <- records_filtered %>%
#     mutate(latitude = latitudeE7 / 1e7,
#            longitude = longitudeE7 / 1e7)
```

```
#
#   # Remove activities "Null" and "Still"
#   records_filtered <- records_filtered %>%
#     filter(!activity1 %in% c("Null", "Still"))
#
#   # Remove rows with missing latitude or longitude
#   records_filtered <- records_filtered %>%
#     filter(!is.na(latitude) & !is.na(longitude))
#
#   # Further clean the activity1 field
#   records_filtered <- records_filtered %>%
#     mutate(activity1 = gsub("list\\(activity = \"|\"\\)", "", activity1)) %>%
#     filter(activity1 != "NULL")
#
#   # Convert to spatial dataframe
#   sf_records_filtered <- st_as_sf(records_filtered, coords = c("longitude", "latitude"), cr
#
#   # Calculate distances between consecutive points
#   records_filtered <- records_filtered %>%
#     arrange(timestamp) %>%
#     mutate(
#       dist = distHaversine(cbind(longitude, latitude), cbind(lag(longitude), lag(latitude))
#       timestamp = as.POSIXct(timestamp, format = "%Y-%m-%dT%H:%M:%S", tz = "UTC"),
#       year = year(timestamp),
#       month = month(timestamp, label = TRUE, abbr = FALSE), # Monate als Namen
#       day = wday(timestamp, label = TRUE, abbr = FALSE), # Wochentage als Namen
#       hour = hour(timestamp),
#       period = time_of_day(hour)
#     ) %>%
#     replace_na(list(dist = 0))  # Replace NA distances with 0
#
#   return(records_filtered)
# }
#
# # Assuming records_N is loaded
# records_2022 <- process_year_data(records_N, "2022")
# records_2023 <- process_year_data(records_N, "2023")
# records_2024 <- process_year_data(records_N, "2024")
#
# # Combine all years
# all_records <- bind_rows(records_2022, records_2023, records_2024)
#
```

```
# # Define the order of periods and days
# all_records$period <- factor(all_records$period, levels = c("Morgen (06-10 Uhr)", "Mittag
# all_records$day <- factor(all_records$day, levels = c("Montag", "Dienstag", "Mittwoch", "Do
#
# # Filter data to the area with high density of points
# # Adjust these limits based on your data to zoom in on the area of interest
# x_limits <- c(-10, 30)
# y_limits <- c(35, 65)
#
# # Heatmap der Koordinaten nach Jahr (gezoomt auf dichten Bereich)
# p_heatmap_zoom <- ggplot() +
#   geom_sf(data = europe_boundaries, fill = "lightgrey", color = "white") +
#   stat_density2d(data = all_records, aes(x = longitude, y = latitude, fill = after_stat(lev
#   scale_fill_viridis_c(option = "inferno", name = "Häufigkeit") +
#   scale_alpha(range = c(0.4, 0.8), guide = "none") +
#   coord_sf(xlim = x_limits, ylim = y_limits) +
#   labs(
#     title = "Heatmap der Koordinaten über die Jahre (gezoomt)",
#     x = "Längengrad",
#     y = "Breitengrad"
#   ) +
#   theme_minimal()
#
# # Heatmap der Koordinaten nach Jahr und Jahresweise facetting (gezoomt auf dichten Bereich)
# p_heatmap_facet_zoom <- ggplot() +
#   geom_sf(data = europe_boundaries, fill = "lightgrey", color = "white") +
#   stat_density2d(data = all_records, aes(x = longitude, y = latitude, fill = after_stat(lev
#   scale_fill_viridis_c(option = "inferno", name = "Häufigkeit") +
#   scale_alpha(range = c(0.4, 0.8), guide = "none") +
#   coord_sf(xlim = x_limits, ylim = y_limits) +
#   facet_wrap(~ year) +
#   labs(
#     title = "Heatmap der Koordinaten nach Jahr (gezoomt)",
#     x = "Längengrad",
#     y = "Breitengrad"
#   ) +
#   theme_minimal()
#
# # Arrange plots in grid
# grid.arrange(
#   p_heatmap_zoom, p_heatmap_facet_zoom,
#   ncol = 1
```

```
# )
```

Karte Europaweit Clustering

```
# # Load necessary libraries
# library(dplyr)
# library(tidyr)
# library(ggplot2)
# library(sf)
# library(geosphere)
# library(dbscan)
#
# # Set SHAPE_RESTORE_SHX config option
# Sys.setenv(SHAPE_RESTORE_SHX = "YES")
#
# # Load the Europe boundaries shapefile
# europe_boundaries <- st_read("C:/_Data/Master/PaT_24/Projectwork/RProjectwork/cma-project/
#
# # Ensure geometries are valid
# europe_boundaries <- st_make_valid(europe_boundaries)
#
# # Load the data
# # Assuming you have loaded the data into records_N dataframe
# # records_N <- read.csv("path_to_your_data.csv") # Adjust the path accordingly
#
# # Process data for each year
# process_data <- function(data, year) {
#   data %>%
#     filter(grepl(as.character(year), timestamp)) %>%
#     mutate(activity = gsub("list\\(list\\(type = c\\(|\\), confidence = c\\(|\\)\\)\\)", "
#     separate(activity, into = c("activity1", "activity2"), sep = ", confidence = ") %>%
#     separate(activity1, into = c("activity1", "activity1_confidence"), sep = ", ") %>%
#     separate(activity2, into = c("activity2", "activity2_confidence"), sep = ", ") %>%
#     mutate(latitude = latitudeE7 / 1e7,
#            longitude = longitudeE7 / 1e7) %>%
#     filter(!activity1 %in% c("Null", "Still"))
# }
#
# records_2022 <- process_data(records_N, 2022)
# records_2023 <- process_data(records_N, 2023)
```

```
# records_2024 <- process_data(records_N, 2024)
#
# # Function to perform clustering and create convex hulls
# create_clusters_and_hulls <- function(data) {
#   # Perform DBSCAN clustering
#   coords <- cbind(data$longitude, data$latitude)
#   clusters <- dbscan(coords, eps = 0.01, minPts = 5)$cluster
#
#   # Add clusters to data
#   data$cluster <- clusters
#
#   # Convert to spatial dataframe
#   sf_data <- st_as_sf(data, coords = c("longitude", "latitude"), crs = 4326)
#
#   # Create convex hulls
#   hulls <- sf_data %>%
#     group_by(cluster) %>%
#     summarize(geometry = st_union(geometry)) %>%
#     st_convex_hull()
#
#   list(data = data, hulls = hulls)
# }
#
# clusters_hulls_2022 <- create_clusters_and_hulls(records_2022)
# clusters_hulls_2023 <- create_clusters_and_hulls(records_2023)
# clusters_hulls_2024 <- create_clusters_and_hulls(records_2024)
#
# # Plot the data
# plot_data <- function(europe_boundaries, clusters_hulls, year) {
#   ggplot() +
#     geom_sf(data = europe_boundaries, fill = NA, color = "grey50") +
#     geom_point(data = clusters_hulls$data, aes(x = longitude, y = latitude, color = as.fact
#     geom_sf(data = clusters_hulls$hulls, fill = NA, color = "blue", linetype = "dashed") +
#     theme_minimal() +
#     labs(title = paste("Connected Activities and Distances", year),
#          fill = "Activity and Distance",
#          color = "Cluster")
# }
#
# plot_2022 <- plot_data(europe_boundaries, clusters_hulls_2022, 2022)
# plot_2023 <- plot_data(europe_boundaries, clusters_hulls_2023, 2023)
# plot_2024 <- plot_data(europe_boundaries, clusters_hulls_2024, 2024)
```

```
#
# # Print plots
# print(plot_2022)
# print(plot_2023)
# print(plot_2024)
#
```

Zurückgelegte Distanz/Tageszeit+Monat_nur Pro Monat_pro Monat und Aktivität (2022) > Diagramme

```
# # Load necessary libraries
# library(dplyr)
# library(tidyr)
# library(ggplot2)
# library(sf)
# library(scales)
# library(geosphere)
# library(lubridate)
#
# # Hilfsfunktion zur Klassifizierung der Tageszeit
# time_of_day <- function(hour) {
#   case_when(
#     hour >= 6 & hour < 10 ~ "Morgen (06-10 Uhr)",
#     hour >= 10 & hour < 14 ~ "Mittag (10-14 Uhr)",
#     hour >= 14 & hour < 18 ~ "Nachmittag (14-18 Uhr)",
#     hour >= 18 & hour < 22 ~ "Abend (18-22 Uhr)",
#     TRUE ~ "Nacht (22-06 Uhr)"
#   )
# }
#
# # Filter records by timestamp for the year 2022,2023, 2024
# records_filtered <- records_N %>%
#   mutate(Year = year(ymd_hms(timestamp))) %>%
#   filter(Year %in% c(2022, 2023, 2024))
#
#
# # Parse the activity column
# records_filtered <- records_filtered %>%
#   mutate(activity = gsub("list\\(list\\(type = c\\(|\\), confidence = c\\(|\\)\\)\\)", "",
```

```
#   separate(activity, into = c("activity1", "activity2"), sep = ", confidence = ") %>%
#   separate(activity1, into = c("activity1", "activity1_confidence"), sep = ", ") %>%
#   separate(activity2, into = c("activity2", "activity2_confidence"), sep = ", ")
#
# # Convert lat/lon from E7 to decimal degrees
# records_filtered <- records_filtered %>%
#   mutate(latitude = latitudeE7 / 1e7,
#          longitude = longitudeE7 / 1e7)
#
# # Remove activities "Null" and "Still"
# records_filtered <- records_filtered %>%
#   filter(!activity1 %in% c("Null", "Still"))
#
# # Remove rows with missing latitude or longitude
# records_filtered <- records_filtered %>%
#   filter(!is.na(latitude) & !is.na(longitude))
#
# # Convert to spatial dataframe
# sf_records_filtered <- st_as_sf(records_filtered, coords = c("longitude", "latitude"), crs
#
#
#
#
# # Calculate distances between consecutive points
# records_filtered <- records_filtered %>%
#   arrange(timestamp) %>%
#   mutate(
#     dist = distHaversine(cbind(longitude, latitude), cbind(lag(longitude), lag(latitude)))
#     timestamp = as.POSIXct(timestamp, format = "%Y-%m-%dT%H:%M:%S", tz = "UTC"),
#     year = year(timestamp),
#     month = month(timestamp, label = TRUE, abbr = FALSE), # Monate als Namen
#     hour = hour(timestamp),
#     period = time_of_day(hour)
#   ) %>%
#   replace_na(list(dist = 0))  # Replace NA distances with 0
#
# # Summarize distance per activity and per month
# activity_distances <- records_filtered %>%
#   group_by(activity1) %>%
#   summarize(total_distance = sum(dist, na.rm = TRUE))
#
# month_distances <- records_filtered %>%
```

```
#   group_by(month, Year) %>%
#   summarize(total_distance = sum(dist, na.rm = TRUE)) |>
#   filter(!(Year == 2024 & month == "Juni"))
#
# activity_month_distances <- records_filtered %>%
#   group_by(month, activity1) %>%
#   summarize(total_distance = sum(dist, na.rm = TRUE), .groups = 'drop')
#
# # Create labels for the legend
# records_filtered <- records_filtered %>%
#   left_join(activity_distances, by = "activity1") %>%
#   mutate(legend_label = paste(activity1, round(total_distance, 2), "km"))
#
# # Reihenfolge der Tageszeiten definieren
# levels_order <- c("Morgen (06-10 Uhr)", "Mittag (10-14 Uhr)", "Nachmittag (14-18 Uhr)", "Al
# records_filtered$period <- factor(records_filtered$period, levels = levels_order)
#
# # Datenanalyse für die zurückgelegte Distanz nach Tageszeit und Monat
# month_period_distance <- records_filtered %>%
#   group_by(month, period) %>%
#   summarize(total_distance = sum(dist, na.rm = TRUE), .groups = 'drop')
#
# # Visualisierung der zurückgelegten Distanz pro Tageszeit und Monat als Flächendiagramm
# ggplot(month_period_distance, aes(x = period, y = total_distance, fill = month, group = mon
#   geom_area(position = 'stack', alpha = 0.6) +
#   scale_fill_brewer(palette = "Set3") +
#   labs(
#     title = "Zurückgelegte Distanz pro Tageszeit und Monat (2022)",
#     subtitle = "Aufgeteilt nach Morgen, Mittag, Nachmittag, Abend und Nacht",
#     y = "Distanz (km)",
#     fill = "Monat"
#   ) +
#   theme_minimal() +
#   theme(axis.text.x = element_text(angle = 45, hjust = 1))
#
#
# month_distances
#
# ##### Visualisierung der zurückgelegten Distanz pro Monat als Balkendiagramm - verwendet in
# balken_year <-
#   ggplot(month_distances, aes(x = month, y = total_distance, fill = month)) +
#   geom_bar(stat = "identity") +
```

```
#   scale_fill_brewer(palette = "Set3") +
#   labs(
#     title = "Zurückgelegte Distanz pro Monat und Jahr",
#     y = "Distanz (km)",
#     fill = "Monat"
#   ) +
#   theme_minimal() +
#   theme(
#     axis.text.x = element_blank(),  # Entfernt die x-Achsen-Beschriftungen
#     axis.title.x = element_blank(),  # Entfernt den Titel der x-Achse
#     # Optional: Dreht die y-Achsen-Beschriftungen
#   ) +
#   scale_y_continuous(limits = c(0, 7000), breaks = seq(0, 7000, by = 1000)) + # Setzt den
#   facet_grid( . ~Year)
#
# print(balken_year)
#
# # Visualisierung der zurückgelegten Distanz pro Monat und Aktivität als gestapeltes Balken
# ggplot(activity_month_distances, aes(x = month, y = total_distance, fill = activity1)) +
#   geom_bar(stat = "identity", position = "stack") +
#   scale_fill_brewer(palette = "Set3") +
#   labs(
#     title = "Zurückgelegte Distanz pro Monat und Aktivität (2022)",
#     x = "Monat",
#     y = "Distanz (km)",
#     fill = "Aktivität"
#   ) +
#   theme_minimal() +
#   theme(axis.text.x = element_text(angle = 45, hjust = 1))
```

Zurückgelegte Distanz/Tageszeit+Monat_nur Pro Monat_pro Monat und Aktivität (2023) > Diagramme

```
# # Load necessary libraries
# library(dplyr)
# library(tidyr)
# library(ggplot2)
# library(sf)
# library(scales)
```

```
# library(geosphere)
# library(lubridate)
#
# # Hilfsfunktion zur Klassifizierung der Tageszeit
# time_of_day <- function(hour) {
#   case_when(
#     hour >= 6 & hour < 10 ~ "Morgen (06-10 Uhr)",
#     hour >= 10 & hour < 14 ~ "Mittag (10-14 Uhr)",
#     hour >= 14 & hour < 18 ~ "Nachmittag (14-18 Uhr)",
#     hour >= 18 & hour < 22 ~ "Abend (18-22 Uhr)",
#     TRUE ~ "Nacht (22-06 Uhr)"
#   )
# }
#
# # Filter records by timestamp for the year 2023
# records_filtered <- records_N %>%
#   filter(grepl("2023", timestamp))
#
# # Parse the activity column
# records_filtered <- records_filtered %>%
#   mutate(activity = gsub("list\\(list\\(type = c\\(|\\), confidence = c\\(|\\)\\)\\)", "",
#   separate(activity, into = c("activity1", "activity2"), sep = ", confidence = ") %>%
#   separate(activity1, into = c("activity1", "activity1_confidence"), sep = ", ") %>%
#   separate(activity2, into = c("activity2", "activity2_confidence"), sep = ", ")
#
# # Convert lat/lon from E7 to decimal degrees
# records_filtered <- records_filtered %>%
#   mutate(latitude = latitudeE7 / 1e7,
#          longitude = longitudeE7 / 1e7)
#
# # Remove activities "Null" and "Still"
# records_filtered <- records_filtered %>%
#   filter(!activity1 %in% c("Null", "Still"))
#
# # Convert to spatial dataframe
# sf_records_filtered <- st_as_sf(records_filtered, coords = c("longitude", "latitude"), crs
#
# # Calculate distances between consecutive points
# records_filtered <- records_filtered %>%
#   arrange(timestamp) %>%
#   mutate(
#     dist = distHaversine(cbind(longitude, latitude), cbind(lag(longitude), lag(latitude)))
```

```
#     timestamp = as.POSIXct(timestamp, format = "%Y-%m-%dT%H:%M:%S", tz = "UTC"),
#     year = year(timestamp),
#     month = month(timestamp, label = TRUE, abbr = FALSE), # Monate als Namen
#     hour = hour(timestamp),
#     period = time_of_day(hour)
#   ) %>%
#   replace_na(list(dist = 0))  # Replace NA distances with 0
#
# # Summarize distance per activity and per month
# activity_distances <- records_filtered %>%
#   group_by(activity1) %>%
#   summarize(total_distance = sum(dist, na.rm = TRUE))
#
# month_distances <- records_filtered %>%
#   group_by(month) %>%
#   summarize(total_distance = sum(dist, na.rm = TRUE))
#
# activity_month_distances <- records_filtered %>%
#   group_by(month, activity1) %>%
#   summarize(total_distance = sum(dist, na.rm = TRUE), .groups = 'drop')
#
# # Create labels for the legend
# records_filtered <- records_filtered %>%
#   left_join(activity_distances, by = "activity1") %>%
#   mutate(legend_label = paste(activity1, round(total_distance, 2), "km"))
#
# # Reihenfolge der Tageszeiten definieren
# levels_order <- c("Morgen (06-10 Uhr)", "Mittag (10-14 Uhr)", "Nachmittag (14-18 Uhr)", "Al
# records_filtered$period <- factor(records_filtered$period, levels = levels_order)
#
# # Datenanalyse für die zurückgelegte Distanz nach Tageszeit und Monat
# month_period_distance <- records_filtered %>%
#   group_by(month, period) %>%
#   summarize(total_distance = sum(dist, na.rm = TRUE), .groups = 'drop')
#
# # Visualisierung der zurückgelegten Distanz pro Tageszeit und Monat als Flächendiagramm
# ggplot(month_period_distance, aes(x = period, y = total_distance, fill = month, group = mon
#   geom_area(position = 'stack', alpha = 0.6) +
#   scale_fill_brewer(palette = "Set3") +
#   labs(
#     title = "Zurückgelegte Distanz pro Tageszeit und Monat (2023)",
#     subtitle = "Aufgeteilt nach Morgen, Mittag, Nachmittag, Abend und Nacht",
```

```
#     x = "Tageszeit",
#     y = "Distanz (km)",
#     fill = "Monat"
#   ) +
#   theme_minimal() +
#   theme(axis.text.x = element_text(angle = 45, hjust = 1))
#
# # Visualisierung der zurückgelegten Distanz pro Monat als Balkendiagramm
# ggplot(month_distances, aes(x = month, y = total_distance, fill = month)) +
#   geom_bar(stat = "identity") +
#   scale_fill_brewer(palette = "Set3") +
#   labs(
#     title = "Zurückgelegte Distanz pro Monat (2023)",
#     x = "Monat",
#     y = "Distanz (km)",
#     fill = "Monat"
#   ) +
#   theme_minimal() +
#   theme(axis.text.x = element_text(angle = 45, hjust = 1))
#
# # Visualisierung der zurückgelegten Distanz pro Monat und Aktivität als gestapeltes Balken
# ggplot(activity_month_distances, aes(x = month, y = total_distance, fill = activity1)) +
#   geom_bar(stat = "identity", position = "stack") +
#   scale_fill_brewer(palette = "Set3") +
#   labs(
#     title = "Zurückgelegte Distanz pro Monat und Aktivität (2023)",
#     x = "Monat",
#     y = "Distanz (km)",
#     fill = "Aktivität"
#   ) +
#   theme_minimal() +
#   theme(axis.text.x = element_text(angle = 45, hjust = 1))
#
#
```

Zurückgelegte Distanz/Tageszeit+Monat_nur Pro Monat_pro Monat und Aktivität (2024) > Diagramme

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

```
# library(tidyr)
# library(ggplot2)
# library(sf)
# library(scales)
# library(geosphere)
# library(lubridate)
#
# # Hilfsfunktion zur Klassifizierung der Tageszeit
# time_of_day <- function(hour) {
#   case_when(
#     hour >= 6 & hour < 10 ~ "Morgen (06-10 Uhr)",
#     hour >= 10 & hour < 14 ~ "Mittag (10-14 Uhr)",
#     hour >= 14 & hour < 18 ~ "Nachmittag (14-18 Uhr)",
#     hour >= 18 & hour < 22 ~ "Abend (18-22 Uhr)",
#     TRUE ~ "Nacht (22-06 Uhr)"
#   )
# }
#
# # Filter records by timestamp for the year 2024
# records_filtered <- records_N %>%
#   filter(grepl("2024", timestamp))
#
# # Parse the activity column
# records_filtered <- records_filtered %>%
#   mutate(activity = gsub("list\\(list\\(type = c\\(|\\), confidence = c\\(|\\)\\)\\)", "",
#   separate(activity, into = c("activity1", "activity2"), sep = ", confidence = ") %>%
#   separate(activity1, into = c("activity1", "activity1_confidence"), sep = ", ") %>%
#   separate(activity2, into = c("activity2", "activity2_confidence"), sep = ", ")
#
# # Convert lat/lon from E7 to decimal degrees
# records_filtered <- records_filtered %>%
#   mutate(latitude = latitudeE7 / 1e7,
#          longitude = longitudeE7 / 1e7)
#
# # Remove activities "Null" and "Still"
# records_filtered <- records_filtered %>%
#   filter(!activity1 %in% c("Null", "Still"))
#
# # Remove rows with missing latitude or longitude
# records_filtered <- records_filtered %>%
#   filter(!is.na(latitude) & !is.na(longitude))
#
```

```
# # Convert to spatial dataframe
# sf_records_filtered <- st_as_sf(records_filtered, coords = c("longitude", "latitude"), crs
#
# # Calculate distances between consecutive points
# records_filtered <- records_filtered %>%
#   arrange(timestamp) %>%
#   mutate(
#     dist = distHaversine(cbind(longitude, latitude), cbind(lag(longitude), lag(latitude)))
#     timestamp = as.POSIXct(timestamp, format = "%Y-%m-%dT%H:%M:%S", tz = "UTC"),
#     year = year(timestamp),
#     month = month(timestamp, label = TRUE, abbr = FALSE), # Monate als Namen
#     hour = hour(timestamp),
#     period = time_of_day(hour)
#   ) %>%
#   replace_na(list(dist = 0))  # Replace NA distances with 0
#
# # Summarize distance per activity and per month
# activity_distances <- records_filtered %>%
#   group_by(activity1) %>%
#   summarize(total_distance = sum(dist, na.rm = TRUE))
#
# month_distances <- records_filtered %>%
#   group_by(month) %>%
#   summarize(total_distance = sum(dist, na.rm = TRUE))
#
# activity_month_distances <- records_filtered %>%
#   group_by(month, activity1) %>%
#   summarize(total_distance = sum(dist, na.rm = TRUE), .groups = 'drop')
#
# # Create labels for the legend
# records_filtered <- records_filtered %>%
#   left_join(activity_distances, by = "activity1") %>%
#   mutate(legend_label = paste(activity1, round(total_distance, 2), "km"))
#
# # Reihenfolge der Tageszeiten definieren
# levels_order <- c("Morgen (06-10 Uhr)", "Mittag (10-14 Uhr)", "Nachmittag (14-18 Uhr)", "Al
# records_filtered$period <- factor(records_filtered$period, levels = levels_order)
#
# # Datenanalyse für die zurückgelegte Distanz nach Tageszeit und Monat
# month_period_distance <- records_filtered %>%
#   group_by(month, period) %>%
#   summarize(total_distance = sum(dist, na.rm = TRUE), .groups = 'drop')
```

```
#
# # Visualisierung der zurückgelegten Distanz pro Tageszeit und Monat als Flächendiagramm
# ggplot(month_period_distance, aes(x = period, y = total_distance, fill = month, group = mon
#   geom_area(position = 'stack', alpha = 0.6) +
#   scale_fill_brewer(palette = "Set3") +
#   labs(
#     title = "Zurückgelegte Distanz pro Tageszeit und Monat (2024)",
#     subtitle = "Aufgeteilt nach Morgen, Mittag, Nachmittag, Abend und Nacht",
#     x = "Tageszeit",
#     y = "Distanz (km)",
#     fill = "Monat"
#   ) +
#   theme_minimal() +
#   theme(axis.text.x = element_text(angle = 45, hjust = 1))
#
# # Visualisierung der zurückgelegten Distanz pro Monat als Balkendiagramm
# ggplot(month_distances, aes(x = month, y = total_distance, fill = month)) +
#   geom_bar(stat = "identity") +
#   scale_fill_brewer(palette = "Set3") +
#   labs(
#     title = "Zurückgelegte Distanz pro Monat (2024)",
#     x = "Monat",
#     y = "Distanz (km)",
#     fill = "Monat"
#   ) +
#   theme_minimal() +
#   theme(axis.text.x = element_text(angle = 45, hjust = 1))
#
# # Visualisierung der zurückgelegten Distanz pro Monat und Aktivität als gestapeltes Balkend
# ggplot(activity_month_distances, aes(x = month, y = total_distance, fill = activity1)) +
#   geom_bar(stat = "identity", position = "stack") +
#   scale_fill_brewer(palette = "Set3") +
#   labs(
#     title = "Zurückgelegte Distanz pro Monat und Aktivität (2024)",
#     x = "Monat",
#     y = "Distanz (km)",
#     fill = "Aktivität"
#   ) +
#   theme_minimal() +
#   theme(axis.text.x = element_text(angle = 45, hjust = 1))
#
```

Zusammenstellung Zurückgelegte Distanz/Tageszeit+Monat_nur Pro Monat_pro Monat und Aktivität (2022-2014) > Diagramme A

```
# # Load necessary libraries
# library(dplyr)
# library(tidyr)
# library(ggplot2)
# library(sf)
# library(scales)
# library(geosphere)
# library(lubridate)
# library(gridExtra)
#
# # Hilfsfunktion zur Klassifizierung der Tageszeit
# time_of_day <- function(hour) {
#   case_when(
#     hour >= 6 & hour < 10 ~ "Morgen (06-10 Uhr)",
#     hour >= 10 & hour < 14 ~ "Mittag (10-14 Uhr)",
#     hour >= 14 & hour < 18 ~ "Nachmittag (14-18 Uhr)",
#     hour >= 18 & hour < 22 ~ "Abend (18-22 Uhr)",
#     TRUE ~ "Nacht (22-06 Uhr)"
#   )
# }
#
# plot_distance_per_time_of_day_and_month <- function(records_filtered, year_label) {
#   # Filter records by timestamp for the specified year
#   records_filtered <- records_filtered %>%
#     filter(grepl(year_label, timestamp))
#
#   # Parse the activity column
#   records_filtered <- records_filtered %>%
#     mutate(activity = gsub("list\\(list\\(type = c\\(|\\), confidence = c\\(|\\)\\)\\)", "", a
#     separate(activity, into = c("activity1", "activity2"), sep = ", confidence = ") %>%
#     separate(activity1, into = c("activity1", "activity1_confidence"), sep = ", ") %>%
#     separate(activity2, into = c("activity2", "activity2_confidence"), sep = ", ")
#
#   # Convert lat/lon from E7 to decimal degrees
#   records_filtered <- records_filtered %>%
#     mutate(latitude = latitudeE7 / 1e7,
#            longitude = longitudeE7 / 1e7)
#
```

```
#   # Remove activities "Null" and "Still"
#   records_filtered <- records_filtered %>%
#     filter(!activity1 %in% c("Null", "Still"))
#
#   # Remove rows with missing latitude or longitude
#   records_filtered <- records_filtered %>%
#     filter(!is.na(latitude) & !is.na(longitude))
#
#   # Convert to spatial dataframe
#   sf_records_filtered <- st_as_sf(records_filtered, coords = c("longitude", "latitude"), cr
#
#   # Calculate distances between consecutive points
#   records_filtered <- records_filtered %>%
#     arrange(timestamp) %>%
#     mutate(
#       dist = distHaversine(cbind(longitude, latitude), cbind(lag(longitude), lag(latitude))
#       timestamp = as.POSIXct(timestamp, format = "%Y-%m-%dT%H:%M:%S", tz = "UTC"),
#       year = year(timestamp),
#       month = month(timestamp, label = TRUE, abbr = FALSE), # Monate als Namen
#       hour = hour(timestamp),
#       period = time_of_day(hour)
#     ) %>%
#     replace_na(list(dist = 0))  # Replace NA distances with 0
#
#   # Reorder periods
#   records_filtered$period <- factor(records_filtered$period, levels = c("Morgen (06-10 Uhr)
#
#   # Datenanalyse für die zurückgelegte Distanz nach Tageszeit und Monat
#   month_period_distance <- records_filtered %>%
#     group_by(month, period) %>%
#     summarize(total_distance = sum(dist, na.rm = TRUE), .groups = 'drop')
#
#   # Visualisierung der zurückgelegten Distanz pro Tageszeit und Monat als Flächendiagramm
#   p1 <- ggplot(month_period_distance, aes(x = period, y = total_distance, fill = month, gro
#     geom_area(position = 'stack', alpha = 0.6) +
#     scale_fill_brewer(palette = "Set3") +
#     labs(
#       title = paste("Zurückgelegte Distanz pro Tageszeit und Monat (", year_label, ")", sep
#       subtitle = "Aufgeteilt nach Morgen, Mittag, Nachmittag, Abend und Nacht",
#       x = "Tageszeit",
#       y = "Distanz (km)",
#       fill = "Monat"
```

```
#     ) +
#     theme_minimal() +
#     theme(axis.text.x = element_text(angle = 45, hjust = 1))
#
#   # Summarize distance per month
#   month_distances <- records_filtered %>%
#     group_by(month) %>%
#     summarize(total_distance = sum(dist, na.rm = TRUE))
#
#   # Visualisierung der zurückgelegten Distanz pro Monat als Balkendiagramm
#   p2 <- ggplot(month_distances, aes(x = month, y = total_distance, fill = month)) +
#     geom_bar(stat = "identity") +
#     scale_fill_brewer(palette = "Set3") +
#     labs(
#       title = paste("Zurückgelegte Distanz pro Monat (", year_label, ")", sep = ""),
#       x = "Monat",
#       y = "Distanz (km)",
#       fill = "Monat"
#     ) +
#     theme_minimal() +
#     theme(axis.text.x = element_text(angle = 45, hjust = 1))
#
#   # Summarize distance per activity and per month
#   activity_month_distances <- records_filtered %>%
#     group_by(month, activity1) %>%
#     summarize(total_distance = sum(dist, na.rm = TRUE), .groups = 'drop')
#
#   # Visualisierung der zurückgelegten Distanz pro Monat und Aktivität als gestapeltes Balk
#   p3 <- ggplot(activity_month_distances, aes(x = month, y = total_distance, fill = activity
#     geom_bar(stat = "identity", position = "stack") +
#     scale_fill_brewer(palette = "Set3") +
#     labs(
#       title = paste("Zurückgelegte Distanz pro Monat und Aktivität (", year_label, ")", sep
#       x = "Monat",
#       y = "Distanz (km)",
#       fill = "Aktivität"
#     ) +
#     theme_minimal() +
#     theme(axis.text.x = element_text(angle = 45, hjust = 1))
#
#   return(list(p1, p2, p3))
# }
```

```
#
# # Assuming records_N is loaded
# plots_2022 <- plot_distance_per_time_of_day_and_month(records_N, "2022")
# plots_2023 <- plot_distance_per_time_of_day_and_month(records_N, "2023")
# plots_2024 <- plot_distance_per_time_of_day_and_month(records_N, "2024")
#
# # Arrange plots in grid
# grid.arrange(
#   grobs = c(plots_2022, plots_2023, plots_2024),
#   ncol = 3
# )
```

#Auswertungen einfache

```
# # Load necessary libraries
# library(dplyr)
# library(tidyr)
# library(ggplot2)
# library(sf)
# library(scales)
# library(geosphere)
# library(lubridate)
# library(gridExtra)
# library(GGally)
# library(RColorBrewer)
#
# # Hilfsfunktion zur Klassifizierung der Tageszeit
# time_of_day <- function(hour) {
#   case_when(
#     hour >= 6 & hour < 10 ~ "Morgen (06-10 Uhr)",
#     hour >= 10 & hour < 14 ~ "Mittag (10-14 Uhr)",
#     hour >= 14 & hour < 18 ~ "Nachmittag (14-18 Uhr)",
#     hour >= 18 & hour < 22 ~ "Abend (18-22 Uhr)",
#     TRUE ~ "Nacht (22-06 Uhr)"
#   )
# }
#
# process_year_data <- function(records_filtered, year_label) {
#   # Filter records by timestamp for the specified year
#   records_filtered <- records_filtered %>%
#     filter(grepl(year_label, timestamp))
#
```

```
#   # Parse the activity column
#   records_filtered <- records_filtered %>%
#     mutate(activity = gsub("list\\(list\\(type = c\\(|\\), confidence = c\\(|\\)\\)\\)", "", a
#     separate(activity, into = c("activity1", "activity2"), sep = ", confidence = ") %>%
#     separate(activity1, into = c("activity1", "activity1_confidence"), sep = ", ") %>%
#     separate(activity2, into = c("activity2", "activity2_confidence"), sep = ", ")
#
#   # Convert lat/lon from E7 to decimal degrees
#   records_filtered <- records_filtered %>%
#     mutate(latitude = latitudeE7 / 1e7,
#            longitude = longitudeE7 / 1e7)
#
#   # Remove activities "Null" and "Still"
#   records_filtered <- records_filtered %>%
#     filter(!activity1 %in% c("Null", "Still"))
#
#   # Remove rows with missing latitude or longitude
#   records_filtered <- records_filtered %>%
#     filter(!is.na(latitude) & !is.na(longitude))
#
#   # Convert to spatial dataframe
#   sf_records_filtered <- st_as_sf(records_filtered, coords = c("longitude", "latitude"), cr
#
#   # Calculate distances between consecutive points
#   records_filtered <- records_filtered %>%
#     arrange(timestamp) %>%
#     mutate(
#       dist = distHaversine(cbind(longitude, latitude), cbind(lag(longitude), lag(latitude))
#       timestamp = as.POSIXct(timestamp, format = "%Y-%m-%dT%H:%M:%S", tz = "UTC"),
#       year = year(timestamp),
#       month = month(timestamp, label = TRUE, abbr = FALSE), # Monate als Namen
#       hour = hour(timestamp),
#       period = time_of_day(hour)
#     ) %>%
#     replace_na(list(dist = 0))  # Replace NA distances with 0
#
#   return(records_filtered)
# }
#
# # Assuming records_N is loaded
# records_2022 <- process_year_data(records_N, "2022")
# records_2023 <- process_year_data(records_N, "2023")
```

```
# records_2024 <- process_year_data(records_N, "2024")
#
# # Combine all years
# all_records <- bind_rows(records_2022, records_2023, records_2024)
#
# # Define the order of periods
# all_records$period <- factor(all_records$period, levels = c("Morgen (06-10 Uhr)", "Mittag
#
# # Analyse Verteilung, Korrelation und Trend
#
# # Verteilung der zurückgelegten Distanz nach Tageszeit und Jahr
# dist_by_period_year <- all_records %>%
#   group_by(year, period) %>%
#   summarize(total_distance = sum(dist, na.rm = TRUE), .groups = 'drop')
#
# p1 <- ggplot(dist_by_period_year, aes(x = period, y = total_distance, fill = factor(year)))
#   geom_bar(stat = "identity", position = "dodge") +
#   scale_fill_brewer(palette = "Set3") +
#   labs(
#     title = "Verteilung der zurückgelegten Distanz nach Tageszeit und Jahr",
#     x = "Tageszeit",
#     y = "Distanz (km)",
#     fill = "Jahr"
#   ) +
#   theme_minimal() +
#   theme(axis.text.x = element_text(angle = 45, hjust = 1))
#
# # Verteilung der zurückgelegten Distanz nach Monat und Jahr
# dist_by_month_year <- all_records %>%
#   group_by(year, month) %>%
#   summarize(total_distance = sum(dist, na.rm = TRUE), .groups = 'drop')
#
# p2 <- ggplot(dist_by_month_year, aes(x = month, y = total_distance, fill = factor(year))) +
#   geom_bar(stat = "identity", position = "dodge") +
#   scale_fill_brewer(palette = "Set3") +
#   labs(
#     title = "Verteilung der zurückgelegten Distanz nach Monat und Jahr",
#     x = "Monat",
#     y = "Distanz (km)",
#     fill = "Jahr"
#   ) +
#   theme_minimal() +
```

```
#   theme(axis.text.x = element_text(angle = 45, hjust = 1))
#
# # Korrelation zwischen Monat und zurückgelegter Distanz für jedes Jahr
# cor_by_year <- all_records %>%
#   group_by(year, month) %>%
#   summarize(total_distance = sum(dist, na.rm = TRUE), .groups = 'drop') %>%
#   group_by(year) %>%
#   summarize(correlation = cor(as.numeric(month), total_distance))
#
# p3 <- ggplot(cor_by_year, aes(x = factor(year), y = correlation, color = factor(year), grou
#   geom_line(size = 1.5) +
#   geom_point(size = 4) +
#   scale_color_brewer(palette = "Set3") +
#   labs(
#     title = "Korrelation zwischen Monat und zurückgelegter Distanz",
#     x = "Jahr",
#     y = "Korrelation",
#     color = "Jahr"
#   ) +
#   theme_minimal()
#
# # Korrelation zwischen Tageszeit und zurückgelegter Distanz für jedes Jahr
# cor_period_by_year <- all_records %>%
#   group_by(year, period) %>%
#   summarize(total_distance = sum(dist, na.rm = TRUE), .groups = 'drop') %>%
#   group_by(year) %>%
#   summarize(correlation = cor(as.numeric(as.factor(period)), total_distance))
#
# p4 <- ggplot(cor_period_by_year, aes(x = factor(year), y = correlation, color = factor(yea
#   geom_line(size = 1.5) +
#   geom_point(size = 4) +
#   scale_color_brewer(palette = "Set3") +
#   labs(
#     title = "Korrelation zwischen Tageszeit und zurückgelegter Distanz",
#     x = "Jahr",
#     y = "Korrelation",
#     color = "Jahr"
#   ) +
#   theme_minimal()
#
# # Trendanalyse der zurückgelegten Distanz über die Monate hinweg (2024) mit linearer Regres
# trend_2024 <- records_2024 %>%
```

```
#   group_by(month) %>%
#   summarize(total_distance = sum(dist, na.rm = TRUE), .groups = 'drop')
#
# p5 <- ggplot(trend_2024, aes(x = as.numeric(month), y = total_distance)) +
#   geom_point() +
#   geom_smooth(method = "lm", se = FALSE, color = "blue") +
#   labs(
#     title = "Trend der zurückgelegten Distanz über die Monate hinweg (2024)",
#     x = "Monat",
#     y = "Distanz (km)"
#   ) +
#   theme_minimal()
#
# # Gesamt-Korrelation zwischen Monat und zurückgelegter Distanz (2022-2024)
# cor_overall_month <- all_records %>%
#   group_by(month) %>%
#   summarize(total_distance = sum(dist, na.rm = TRUE), .groups = 'drop') %>%
#   summarize(correlation = cor(as.numeric(month), total_distance))
#
# p6 <- ggplot(cor_overall_month, aes(x = 1, y = correlation)) +
#   geom_point(size = 4) +
#   scale_x_continuous(breaks = NULL) +
#   labs(
#     title = "Gesamt-Korrelation zwischen Monat und zurückgelegter Distanz (2022-2024)",
#     x = "",
#     y = "Korrelation"
#   ) +
#   theme_minimal()
#
# # Korrelation zwischen Art der Aktivität und zurückgelegter Distanz für jedes Jahr
# cor_activity_by_year <- all_records %>%
#   group_by(year, activity1) %>%
#   summarize(total_distance = sum(dist, na.rm = TRUE), .groups = 'drop') %>%
#   group_by(year) %>%
#   summarize(correlation = cor(as.numeric(as.factor(activity1)), total_distance))
#
# p7 <- ggplot(cor_activity_by_year, aes(x = factor(year), y = correlation, color = factor(ye
#   geom_line(size = 1.5) +
#   geom_point(size = 4) +
#   scale_color_brewer(palette = "Set3") +
#   labs(
#     title = "Korrelation zwischen Art der Aktivität und zurückgelegter Distanz",
```

```
#     x = "Jahr",
#     y = "Korrelation",
#     color = "Jahr"
#   ) +
#   theme_minimal()
#
# # Gesamt-Korrelation zwischen Art der Aktivität und zurückgelegter Distanz (2022-2024)
# cor_overall_activity <- all_records %>%
#   group_by(activity1) %>%
#   summarize(total_distance = sum(dist, na.rm = TRUE), .groups = 'drop') %>%
#   summarize(correlation = cor(as.numeric(as.factor(activity1)), total_distance))
#
# p8 <- ggplot(cor_overall_activity, aes(x = 1, y = correlation)) +
#   geom_point(size = 4) +
#   scale_x_continuous(breaks = NULL) +
#   labs(
#     title = "Gesamt-Korrelation zwischen Art der Aktivität und zurückgelegter Distanz (202
#     x = "",
#     y = "Korrelation"
#   ) +
#   theme_minimal()
#
# # Arrange plots in grid
# grid.arrange(
#   p1, p2, p3, p4, p5, p6, p7, p8,
#   ncol = 2
# )
```

#Auswertungen erweitert

```
# # Load necessary libraries
# library(dplyr)
# library(tidyr)
# library(ggplot2)
# library(sf)
# library(scales)
# library(geosphere)
# library(lubridate)
# library(gridExtra)
# library(GGally)
# library(RColorBrewer)
#
```

```
# # Hilfsfunktion zur Klassifizierung der Tageszeit
# time_of_day <- function(hour) {
#   case_when(
#     hour >= 6 & hour < 10 ~ "Morgen (06-10 Uhr)",
#     hour >= 10 & hour < 14 ~ "Mittag (10-14 Uhr)",
#     hour >= 14 & hour < 18 ~ "Nachmittag (14-18 Uhr)",
#     hour >= 18 & hour < 22 ~ "Abend (18-22 Uhr)",
#     TRUE ~ "Nacht (22-06 Uhr)"
#   )
# }
#
# process_year_data <- function(records_filtered, year_label) {
#   # Filter records by timestamp for the specified year
#   records_filtered <- records_filtered %>%
#     filter(grepl(year_label, timestamp))
#
#   # Parse the activity column
#   records_filtered <- records_filtered %>%
#     mutate(activity = gsub("list\\(list\\(type = c\\(|\\), confidence = c\\(|\\)\\)\\)", "", a
#     separate(activity, into = c("activity1", "activity2"), sep = ", confidence = ") %>%
#     separate(activity1, into = c("activity1", "activity1_confidence"), sep = ", ") %>%
#     separate(activity2, into = c("activity2", "activity2_confidence"), sep = ", ")
#
#   # Convert lat/lon from E7 to decimal degrees
#   records_filtered <- records_filtered %>%
#     mutate(latitude = latitudeE7 / 1e7,
#            longitude = longitudeE7 / 1e7)
#
#   # Remove activities "Null" and "Still"
#   records_filtered <- records_filtered %>%
#     filter(!activity1 %in% c("Null", "Still"))
#
#   # Remove rows with missing latitude or longitude
#   records_filtered <- records_filtered %>%
#     filter(!is.na(latitude) & !is.na(longitude))
#
#   # Convert to spatial dataframe
#   sf_records_filtered <- st_as_sf(records_filtered, coords = c("longitude", "latitude"), cr
#
#   # Calculate distances between consecutive points
#   records_filtered <- records_filtered %>%
#     arrange(timestamp) %>%
```

```
#     mutate(
#       dist = distHaversine(cbind(longitude, latitude), cbind(lag(longitude), lag(latitude))
#       timestamp = as.POSIXct(timestamp, format = "%Y-%m-%dT%H:%M:%S", tz = "UTC"),
#       year = year(timestamp),
#       month = month(timestamp, label = TRUE, abbr = FALSE), # Monate als Namen
#       day = wday(timestamp, label = TRUE, abbr = FALSE), # Wochentage als Namen
#       hour = hour(timestamp),
#       period = time_of_day(hour)
#     ) %>%
#     replace_na(list(dist = 0))  # Replace NA distances with 0
#
#   return(records_filtered)
# }
#
# # Assuming records_N is loaded
# records_2022 <- process_year_data(records_N, "2022")
# records_2023 <- process_year_data(records_N, "2023")
# records_2024 <- process_year_data(records_N, "2024")
#
# # Combine all years
# all_records <- bind_rows(records_2022, records_2023, records_2024)
#
# # Define the order of periods and days
# all_records$period <- factor(all_records$period, levels = c("Morgen (06-10 Uhr)", "Mittag
# all_records$day <- factor(all_records$day, levels = c("Montag", "Dienstag", "Mittwoch", "Do
#
# # Analyse Verteilung, Korrelation und Trend
#
# # Verteilung der zurückgelegten Distanz nach Tageszeit und Jahr
# dist_by_period_year <- all_records %>%
#   group_by(year, period) %>%
#   summarize(total_distance = sum(dist, na.rm = TRUE), .groups = 'drop')
#
# p1 <- ggplot(dist_by_period_year, aes(x = period, y = total_distance, fill = factor(year)))
#   geom_bar(stat = "identity", position = "dodge") +
#   scale_fill_brewer(palette = "Set3") +
#   labs(
#     title = "Verteilung der zurückgelegten Distanz nach Tageszeit und Jahr",
#     x = "Tageszeit",
#     y = "Distanz (km)",
#     fill = "Jahr"
#   ) +
```

```
#   theme_minimal() +
#   theme(axis.text.x = element_text(angle = 45, hjust = 1))
#
# # Verteilung der zurückgelegten Distanz nach Monat und Jahr
# dist_by_month_year <- all_records %>%
#   group_by(year, month) %>%
#   summarize(total_distance = sum(dist, na.rm = TRUE), .groups = 'drop')
#
# p2 <- ggplot(dist_by_month_year, aes(x = month, y = total_distance, fill = factor(year))) +
#   geom_bar(stat = "identity", position = "dodge") +
#   scale_fill_brewer(palette = "Set3") +
#   labs(
#     title = "Verteilung der zurückgelegten Distanz nach Monat und Jahr",
#     x = "Monat",
#     y = "Distanz (km)",
#     fill = "Jahr"
#   ) +
#   theme_minimal() +
#   theme(axis.text.x = element_text(angle = 45, hjust = 1))
#
# # Korrelation zwischen Monat und zurückgelegter Distanz für jedes Jahr
# cor_by_year <- all_records %>%
#   group_by(year, month) %>%
#   summarize(total_distance = sum(dist, na.rm = TRUE), .groups = 'drop') %>%
#   group_by(year) %>%
#   summarize(correlation = cor(as.numeric(month), total_distance))
#
# p3 <- ggplot(cor_by_year, aes(x = factor(year), y = correlation, color = factor(year), grou
#   geom_line(size = 1.5) +
#   geom_point(size = 4) +
#   scale_color_brewer(palette = "Set3") +
#   labs(
#     title = "Korrelation zwischen Monat und zurückgelegter Distanz",
#     x = "Jahr",
#     y = "Korrelation",
#     color = "Jahr"
#   ) +
#   theme_minimal()
#
# # Korrelation zwischen Tageszeit und zurückgelegter Distanz für jedes Jahr
# cor_period_by_year <- all_records %>%
#   group_by(year, period) %>%
```

```
#   summarize(total_distance = sum(dist, na.rm = TRUE), .groups = 'drop') %>%
#   group_by(year) %>%
#   summarize(correlation = cor(as.numeric(as.factor(period)), total_distance))
#
# p4 <- ggplot(cor_period_by_year, aes(x = factor(year), y = correlation, color = factor(yea
#   geom_line(size = 1.5) +
#   geom_point(size = 4) +
#   scale_color_brewer(palette = "Set3") +
#   labs(
#     title = "Korrelation zwischen Tageszeit und zurückgelegter Distanz",
#     x = "Jahr",
#     y = "Korrelation",
#     color = "Jahr"
#   ) +
#   theme_minimal()
#
# # Trendanalyse der zurückgelegten Distanz über die Monate hinweg (2024) mit linearer Regres
# trend_2024 <- records_2024 %>%
#   group_by(month) %>%
#   summarize(total_distance = sum(dist, na.rm = TRUE), .groups = 'drop')
#
# p5 <- ggplot(trend_2024, aes(x = as.numeric(month), y = total_distance)) +
#   geom_point() +
#   geom_smooth(method = "lm", se = FALSE, color = "blue") +
#   labs(
#     title = "Trend der zurückgelegten Distanz über die Monate hinweg (2024)",
#     x = "Monat",
#     y = "Distanz (km)"
#   ) +
#   theme_minimal()
#
# # Gesamt-Korrelation zwischen Monat und zurückgelegter Distanz (2022-2024)
# cor_overall_month <- all_records %>%
#   group_by(month) %>%
#   summarize(total_distance = sum(dist, na.rm = TRUE), .groups = 'drop') %>%
#   summarize(correlation = cor(as.numeric(month), total_distance))
#
# p6 <- ggplot(cor_overall_month, aes(x = 1, y = correlation)) +
#   geom_point(size = 4) +
#   scale_x_continuous(breaks = NULL) +
#   labs(
#     title = "Gesamt-Korrelation zwischen Monat und zurückgelegter Distanz (2022-2024)",
```

```
#     x = "",
#     y = "Korrelation"
#   ) +
#   theme_minimal()
#
# # Gesamt-Korrelation zwischen Art der Aktivität und zurückgelegter Distanz (2022-2024)
# cor_overall_activity <- all_records %>%
#   group_by(activity1) %>%
#   summarize(total_distance = sum(dist, na.rm = TRUE), .groups = 'drop') %>%
#   summarize(correlation = cor(as.numeric(as.factor(activity1)), total_distance))
#
# p7 <- ggplot(cor_overall_activity, aes(x = 1, y = correlation)) +
#   geom_point(size = 4) +
#   scale_x_continuous(breaks = NULL) +
#   labs(
#     title = "Gesamt-Korrelation zwischen Art der Aktivität und zurückgelegter Distanz (202
#     x = "",
#     y = "Korrelation"
#   ) +
#   theme_minimal()
#
# # Heatmap der Aktivität pro Stunde und Tag
# activity_heatmap <- all_records %>%
#   group_by(year, day, hour) %>%
#   summarize(total_distance = sum(dist, na.rm = TRUE), .groups = 'drop')
#
# p8 <- ggplot(activity_heatmap, aes(x = hour, y = day, fill = total_distance)) +
#   geom_tile() +
#   facet_wrap(~ year) +
#   scale_fill_gradient(low = "white", high = "red") +
#   labs(
#     title = "Heatmap der Aktivität pro Stunde und Tag",
#     x = "Stunde",
#     y = "Tag",
#     fill = "Distanz (km)"
#   ) +
#   theme_minimal() +
#   theme(axis.text.y = element_text(angle = 0, hjust = 1))
#
# # Heatmap der Aktivität pro Monat
# activity_heatmap_month <- all_records %>%
#   group_by(year, month) %>%
```

```
#   summarize(total_distance = sum(dist, na.rm = TRUE), .groups = 'drop')
#
# p9 <- ggplot(activity_heatmap_month, aes(x = month, y = factor(year), fill = total_distance
#   geom_tile() +
#   scale_fill_gradient(low = "white", high = "red") +
#   labs(
#     title = "Heatmap der Aktivität pro Monat",
#     x = "Monat",
#     y = "Jahr",
#     fill = "Distanz (km)"
#   ) +
#   theme_minimal()
#
# # Vergleich der zurückgelegten Distanzen zwischen verschiedenen Aktivitäten
# activity_comparison <- all_records %>%
#   group_by(activity1) %>%
#   summarize(total_distance = sum(dist, na.rm = TRUE), .groups = 'drop')
#
# p10 <- ggplot(activity_comparison, aes(x = activity1, y = total_distance, fill = activity1)
#   geom_bar(stat = "identity") +
#   scale_fill_brewer(palette = "Set3") +
#   labs(
#     title = "Vergleich der zurückgelegten Distanzen zwischen verschiedenen Aktivitäten",
#     x = "Aktivität",
#     y = "Distanz (km)",
#     fill = "Aktivität"
#   ) +
#   theme_minimal() +
#   theme(axis.text.x = element_text(angle = 45, hjust = 1))
#
# # Arrange plots in grid
# grid.arrange(
#   p1, p2, p3, p4, p5, p6, p7, p8, p9, p10,
#   ncol = 2
# )
```

weitere Anaylsen

```
# # Lade notwendige Bibliotheken
# library(dplyr)
```

```
# library(tidyr)
# library(ggplot2)
# library(sf)
# library(scales)
# library(geosphere)
# library(lubridate)
# library(gridExtra)
# library(GGally)
# library(RColorBrewer)
# library(cluster)
# library(factoextra)
# library(viridis)
# library(spdep)
# library(dbscan)
# library(igraph)
# library(reshape2)
# library(gganimate)
# library(plotly)
#
# # Hilfsfunktion zur Klassifizierung der Tageszeit
# time_of_day <- function(hour) {
#   case_when(
#     hour >= 6 & hour < 10 ~ "Morgen (06-10 Uhr)",
#     hour >= 10 & hour < 14 ~ "Mittag (10-14 Uhr)",
#     hour >= 14 & hour < 18 ~ "Nachmittag (14-18 Uhr)",
#     hour >= 18 & hour < 22 ~ "Abend (18-22 Uhr)",
#     TRUE ~ "Nacht (22-06 Uhr)"
#   )
# }
#
# # Funktion zur Verarbeitung von Daten eines bestimmten Jahres
# process_year_data <- function(records_filtered, year_label) {
#   # Filtere Datensätze nach dem angegebenen Jahr
#   records_filtered <- records_filtered %>%
#     filter(grepl(year_label, timestamp))
#
#   # Analysiere die Aktivitätsspalte
#   records_filtered <- records_filtered %>%
#     mutate(activity = gsub("list\\(list\\(type = c\\(|\\), confidence = c\\(|\\)\\)\\)", "", a
#     separate(activity, into = c("activity1", "activity2"), sep = ", confidence = ", fill =
#     separate(activity1, into = c("activity1", "activity1_confidence"), sep = ", ", fill = "
#     separate(activity2, into = c("activity2", "activity2_confidence"), sep = ", ", fill = "
```

```
#
#   # Konvertiere Latitude/Longitude von E7 in Dezimalgrade
#   records_filtered <- records_filtered %>%
#     mutate(latitude = latitudeE7 / 1e7,
#            longitude = longitudeE7 / 1e7)
#
#   # Entferne Aktivitäten "Null" und "Still"
#   records_filtered <- records_filtered %>%
#     filter(!activity1 %in% c("Null", "Still"))
#
#   # Entferne Zeilen mit fehlender Latitude oder Longitude
#   records_filtered <- records_filtered %>%
#     filter(!is.na(latitude) & !is.na(longitude))
#
#   # Weitere Bereinigung des Feldes activity1
#   records_filtered <- records_filtered %>%
#     mutate(activity1 = gsub("list\\(activity = \"|\"\\)", "", activity1)) %>%
#     filter(activity1 != "NULL")
#
#   # Berechne Distanzen zwischen aufeinanderfolgenden Punkten
#   records_filtered <- records_filtered %>%
#     arrange(timestamp) %>%
#     mutate(
#       dist = distHaversine(cbind(longitude, latitude), cbind(lag(longitude), lag(latitude))
#       timestamp = as.POSIXct(timestamp, format = "%Y-%m-%dT%H:%M:%S", tz = "UTC"),
#       year = year(timestamp),
#       month = month(timestamp, label = TRUE, abbr = FALSE), # Monate als Namen
#       day = wday(timestamp, label = TRUE, abbr = FALSE), # Wochentage als Namen
#       hour = hour(timestamp),
#       period = time_of_day(hour)
#     ) %>%
#     replace_na(list(dist = 0))  # Ersetze NA-Distanzen durch 0
#
#   return(records_filtered)
# }
#
# # Angenommen, records_N ist geladen
# records_2022 <- process_year_data(records_N, "2022")
# records_2023 <- process_year_data(records_N, "2023")
# records_2024 <- process_year_data(records_N, "2024")
#
# # Kombiniere die Daten für alle Jahre
```

```
# all_records <- bind_rows(records_2022, records_2023, records_2024)
#
# # Definiere die Reihenfolge der Perioden und Tage
# all_records$period <- factor(all_records$period, levels = c("Morgen (06-10 Uhr)", "Mittag
# all_records$day <- factor(all_records$day, levels = c("Montag", "Dienstag", "Mittwoch", "Do
#
# # Bereite Daten für das Clustering vor
# clustering_data <- all_records %>%
#   select(longitude, latitude, activity1) %>%
#   mutate(activity1 = as.numeric(factor(activity1)))
#
# # Führe k-means Clustering durch
# set.seed(123)
# kmeans_result <- kmeans(clustering_data[, c("longitude", "latitude")], centers = 5)
#
# # Füge Clusterinformationen zu den Daten hinzu
# all_records$cluster <- factor(kmeans_result$cluster)
#
# # Clustering nach Jahr
# clustering_data_year <- all_records %>%
#   select(longitude, latitude, year)
#
# set.seed(123)
# kmeans_result_year <- kmeans(clustering_data_year[, c("longitude", "latitude")], centers =
#
# # Füge Jahres-Clusterinformationen zu den Daten hinzu
# all_records$cluster_year <- factor(kmeans_result_year$cluster)
#
# # Visualisiere tägliche Aktivitätsmuster
# hourly_activity <- all_records %>%
#   mutate(hour = hour(timestamp)) %>%
#   group_by(hour, activity1) %>%
#   summarize(count = n()) %>%
#   ungroup()
#
# p7 <- ggplot(hourly_activity, aes(x = hour, y = count, color = activity1)) +
#   geom_line() +
#   scale_x_continuous(breaks = 0:23) +
#   labs(
#     title = "Tägliche Aktivitätsmuster",
#     x = "Stunde des Tages",
#     y = "Aktivitätsanzahl",
```

```
#     color = "Aktivität"
#   ) +
#   theme_minimal()
#
# # Wöchentliche Muster
# weekly_activity <- all_records %>%
#   group_by(day, activity1) %>%
#   summarize(count = n()) %>%
#   ungroup()
#
# p8 <- ggplot(weekly_activity, aes(x = day, y = count, fill = activity1)) +
#   geom_bar(stat = "identity", position = "dodge") +
#   labs(
#     title = "Wöchentliche Aktivitätsmuster",
#     x = "Wochentag",
#     y = "Aktivitätsanzahl",
#     fill = "Aktivität"
#   ) +
#   theme_minimal()
#
# # Monatliche Muster
# monthly_activity <- all_records %>%
#   group_by(month, activity1) %>%
#   summarize(count = n()) %>%
#   ungroup()
#
# p9 <- ggplot(monthly_activity, aes(x = month, y = count, fill = activity1)) +
#   geom_bar(stat = "identity", position = "dodge") +
#   labs(
#     title = "Monatliche Aktivitätsmuster",
#     x = "Monat",
#     y = "Aktivitätsanzahl",
#     fill = "Aktivität"
#   ) +
#   theme_minimal()
#
# # Dichtekarten
# sf_all_records <- st_as_sf(all_records, coords = c("longitude", "latitude"), crs = 4326)
#
# # Korrelation Analyse
# activity_period_corr <- all_records %>%
#   select(activity1, period) %>%
```

```
#   group_by(activity1, period) %>%
#   summarize(count = n()) %>%
#   spread(key = period, value = count, fill = 0)
# cor_matrix <- cor(activity_period_corr[-1])
#
# p15 <- ggplot(melt(cor_matrix), aes(Var1, Var2, fill = value)) +
#   geom_tile() +
#   scale_fill_viridis_c() +
#   labs(
#     title = "Korrelation Heatmap der Aktivitäten und Perioden",
#     x = "Aktivität",
#     y = "Periode"
#   ) +
#   theme_minimal()
#
# # Jahr-für-Jahr Vergleich
# yearly_activity <- all_records %>%
#   group_by(year, activity1) %>%
#   summarize(count = n()) %>%
#   ungroup()
#
# p17 <- ggplot(yearly_activity, aes(x = year, y = count, fill = activity1)) +
#   geom_bar(stat = "identity", position = "dodge") +
#   labs(
#     title = "Jahr-für-Jahr Aktivitätsvergleich",
#     x = "Jahr",
#     y = "Aktivitätsanzahl",
#     fill = "Aktivität"
#   ) +
#   theme_minimal()
#
# # # Animierte Karten
# # animated_map <- ggplot(all_records, aes(x = longitude, y = latitude, color = activity1))
# # #   geom_point(size = 2, alpha = 0.6) +
# # #   transition_time(timestamp) +
# # #   labs(
# # #     title = "Animierte Karte der Aktivitäten",
# # #     x = "Längengrad",
# # #     y = "Breitengrad",
# # #     color = "Aktivität"
# # #   ) +
# # #   theme_minimal()
```

```
# #
# # animate(animated_map, renderer = gifski_renderer())
#
# # 3D Visualisierungen
# p19 <- plot_ly(all_records, x = ~longitude, y = ~latitude, z = ~elevation, color = ~activit
#   layout(title = "3D Visualisierung der Aktivitäten")
#
# # Drucke jedes Diagramm einzeln
# print(p7)
# print(p8)
# print(p9)
# print(p15)
# # plotly Ausgaben werden direkt in interaktiven Sitzungen gerendert
# # p16 ist ein igraph Plot, der direkt in interaktiven Sitzungen gerendert wird
# print(p17)
#
# # Anordnung zusätzlicher Diagramme im Raster
# grid.arrange(
#   p7, p8, p9, p15, p17,
#   ncol = 1
# )
#
```

#Saisonale Trends - Diagramme

```
# library(dplyr)
# library(ggplot2)
# library(lubridate)
#
# # Funktion, um Monate in Jahreszeiten umzuwandeln
# month_to_season <- function(month) {
#   case_when(
#     month %in% c(12, 1, 2) ~ "Winter",
#     month %in% c(3, 4, 5) ~ "Frühling",
#     month %in% c(6, 7, 8) ~ "Sommer",
#     TRUE ~ "Herbst"
#   )
# }
#
# # Hilfsfunktion zur Klassifizierung der Tageszeit
# time_of_day <- function(hour) {
#   case_when(
```

```
#     hour >= 6 & hour < 10 ~ "Morgen",
#     hour >= 10 & hour < 14 ~ "Mittag",
#     hour >= 14 & hour < 18 ~ "Nachmittag",
#     hour >= 18 & hour < 22 ~ "Abend",
#     TRUE ~ "Nacht"
#   )
# }
#
# # Daten für A von Januar 2022 bis heute vorbereiten
# As_data <- records_N %>%
#   mutate(
#     timestamp = as.POSIXct(timestamp, format = "%Y-%m-%dT%H:%M:%S", tz = "UTC"),
#     month = month(timestamp),
#     season = month_to_season(month),
#     hour = hour(timestamp),
#     period = time_of_day(hour),
#     next_timestamp = lead(timestamp),
#     duration = as.numeric(difftime(next_timestamp, timestamp, units = "hours")),
#     kilometers = duration * 0.08  # Annahme: 0.08km pro Stunde
#   ) %>%
#   filter(year(timestamp) >= 2022 & timestamp >= as.POSIXct("2022-01-01", tz = "UTC")) %>%
#   arrange(timestamp) %>%
#   filter(!is.na(duration))
#
# # Datenanalyse für die Anzahl der Aufzeichnungen nach Tageszeit
# daily_period_record_count <- As_data %>%
#   group_by(period) %>%
#   summarize(record_count = n(), .groups = 'drop')
#
# # Plot für die Anzahl der Aufzeichnungen nach Tageszeit
# ggplot(daily_period_record_count, aes(x = period, y = record_count, fill = period)) +
#   geom_bar(stat = "identity") +
#   scale_fill_brewer(palette = "Pastel1") +
#   labs(
#     title = "Anzahl der Aufzeichnungen nach Tageszeit",
#     subtitle = "Von Januar 2022 bis heute",
#     x = "Tageszeit",
#     y = "Anzahl der Aufzeichnungen"
#   ) +
#   theme_minimal()
#
# # Datenanalyse für die Anzahl der Aufzeichnungen nach Tageszeit und Jahreszeit
```

```
# season_period_record_count <- As_data %>%
#   group_by(season, period) %>%
#   summarize(record_count = n(), .groups = 'drop')
#
# # Visualisierung der Aufzeichnungen nach Tageszeit und Jahreszeit > Säulendiagramm
# ggplot(season_period_record_count, aes(x = period, y = record_count, fill = season)) +
#   geom_bar(stat = "identity", position = position_dodge()) +
#   scale_fill_brewer(palette = "Set3") +
#   labs(
#     title = "Anzahl der Aufzeichnungen pro Tageszeit und Jahreszeit",
#     subtitle = "Aufgeteilt nach Morgen, Mittag, Nachmittag, Abend und Nacht",
#     x = "Tageszeit",
#     y = "Anzahl der Aufzeichnungen",
#     fill = "Jahreszeit"
#   ) +
#   theme_minimal() +
#   theme(axis.text.x = element_text(angle = 45, hjust = 1))
#
# # Berechnung der Anzahl der Aufzeichnungen pro Jahreszeit
# season_record_count <- As_data %>%
#   group_by(season) %>%
#   summarize(record_count = n(), .groups = 'drop')
#
# # Visualisierung der Beziehung zwischen Jahreszeit und Anzahl der Aufzeichnungen
# ggplot(season_record_count, aes(x = season, y = record_count, group = 1)) +
#   geom_line(aes(group = 1), colour="blue", size=1) +
#   geom_point(size = 3) +
#   labs(title = "Beziehung zwischen Jahreszeit und Anzahl der Aufzeichnungen",
#        x = "Jahreszeit",
#        y = "Anzahl der Aufzeichnungen") +
#   theme_minimal()
#
# # Berechnung der Anzahl der Aufzeichnungen pro Tageszeit
# period_record_count <- As_data %>%
#   group_by(period) %>%
#   summarize(record_count = n(), .groups = 'drop')
#
# # Visualisierung der Beziehung zwischen Tageszeit und Anzahl der Aufzeichnungen
# ggplot(period_record_count, aes(x = period, y = record_count, group = 1)) +
#   geom_line(aes(group = 1), colour="green", size=1) +
#   geom_point(size = 3) +
#   labs(title = "Beziehung zwischen Tageszeit und Anzahl der Aufzeichnungen",
```

```
#        x = "Tageszeit",
#        y = "Anzahl der Aufzeichnungen") +
#   theme_minimal()
#
# library(dplyr)
# library(ggplot2)
# library(lubridate)
#
# # Funktion, um Monate in Jahreszeiten umzuwandeln
# month_to_season <- function(month) {
#   case_when(
#     month %in% c(12, 1, 2) ~ "Winter",
#     month %in% c(3, 4, 5) ~ "Frühling",
#     month %in% c(6, 7, 8) ~ "Sommer",
#     TRUE ~ "Herbst"
#   )
# }
#
# # Hilfsfunktion zur Klassifizierung der Tageszeit
# time_of_day <- function(hour) {
#   case_when(
#     hour >= 6 & hour < 10 ~ "Morgen (06-10 Uhr)",
#     hour >= 10 & hour < 14 ~ "Mittag (10-14 Uhr)",
#     hour >= 14 & hour < 18 ~ "Nachmittag (14-18 Uhr)",
#     hour >= 18 & hour < 22 ~ "Abend (18-22 Uhr)",
#     TRUE ~ "Nacht (22-06 Uhr)"
#   )
# }
#
# # Daten für A von Januar 2022 bis heute vorbereiten
# As_data <- records_N %>%
#   mutate(
#     timestamp = as.POSIXct(timestamp, format = "%Y-%m-%dT%H:%M:%S", tz = "UTC"),
#     month = month(timestamp),
#     season = month_to_season(month),
#     hour = hour(timestamp),
#     period = time_of_day(hour),
#     next_timestamp = lead(timestamp),
#     duration = as.numeric(difftime(next_timestamp, timestamp, units = "hours")),
#     kilometers = duration * 0.08  # Annahme: 0.08km pro Stunde
#   ) %>%
#   filter(year(timestamp) >= 2022 & timestamp >= as.POSIXct("2022-01-01", tz = "UTC")) %>%
```

```
#   arrange(timestamp) %>%
#   filter(!is.na(duration))
#
# # Reihenfolge der Tageszeiten definieren
# levels_order <- c("Morgen (06-10 Uhr)", "Mittag (10-14 Uhr)", "Nachmittag (14-18 Uhr)", "Al
# As_data$period <- factor(As_data$period, levels = levels_order)
#
# # Datenanalyse für die Anzahl der Aufzeichnungen nach Tageszeit und Jahreszeit
# season_period_record_count <- As_data %>%
#   group_by(season, period) %>%
#   summarize(record_count = n(), .groups = 'drop')
#
# # Daten überprüfen
# print(season_period_record_count)
#
# # Visualisierung der Aktivitätenanzahl pro Tageszeit und Jahreszeit als Flächendiagramm
# ggplot(season_period_record_count, aes(x = period, y = record_count, fill = season, group =
#   geom_area(position = 'stack', alpha = 0.6) +
#   scale_fill_brewer(palette = "Set3") +
#   labs(
#     title = "Anzahl der Aktivitäten pro Tageszeit und Jahreszeit",
#     subtitle = "Aufgeteilt nach Morgen, Mittag, Nachmittag, Abend und Nacht",
#     x = "Tageszeit",
#     y = "Anzahl der Aktivitäten",
#     fill = "Jahreszeit"
#   ) +
#   theme_minimal() +
#   theme(axis.text.x = element_text(angle = 45, hjust = 1))
#
# library(dplyr)
# library(ggplot2)
# library(lubridate)
#
# # Hilfsfunktion zur Klassifizierung der Tageszeit
# time_of_day <- function(hour) {
#   case_when(
#     hour >= 6 & hour < 10 ~ "Morgen (06-10 Uhr)",
#     hour >= 10 & hour < 14 ~ "Mittag (10-14 Uhr)",
#     hour >= 14 & hour < 18 ~ "Nachmittag (14-18 Uhr)",
#     hour >= 18 & hour < 22 ~ "Abend (18-22 Uhr)",
#     TRUE ~ "Nacht (22-06 Uhr)"
#   )
```

```
# }
#
# # Daten für A von Januar 2023 bis heute vorbereiten
# As_data <- records_N %>%
#   mutate(
#     timestamp = as.POSIXct(timestamp, format = "%Y-%m-%dT%H:%M:%S", tz = "UTC"),
#     year = year(timestamp),
#     month = month(timestamp, label = TRUE, abbr = FALSE), # Monate als Namen
#     hour = hour(timestamp),
#     period = time_of_day(hour),
#     next_timestamp = lead(timestamp),
#     duration = as.numeric(difftime(next_timestamp, timestamp, units = "hours")),
#     kilometers = duration * 0.08  # Annahme: 0.08km pro Stunde
#   ) %>%
#   filter(year == 2023 & timestamp >= as.POSIXct("2023-01-01", tz = "UTC")) %>%
#   arrange(timestamp) %>%
#   filter(!is.na(duration))
#
# # Reihenfolge der Tageszeiten definieren
# levels_order <- c("Morgen (06-10 Uhr)", "Mittag (10-14 Uhr)", "Nachmittag (14-18 Uhr)", "Al
# As_data$period <- factor(As_data$period, levels = levels_order)
#
# # Datenanalyse für die Anzahl der Aufzeichnungen nach Tageszeit und Monat
# month_period_record_count <- As_data %>%
#   group_by(month, period) %>%
#   summarize(record_count = n(), .groups = 'drop')
#
# # Daten überprüfen
# print(month_period_record_count)
#
# # Visualisierung der Aktivitätenanzahl pro Tageszeit und Monat als Flächendiagramm
# ggplot(month_period_record_count, aes(x = period, y = record_count, fill = month, group = m
#   geom_area(position = 'stack', alpha = 0.6) +
#   scale_fill_brewer(palette = "Set3") +
#   labs(
#     title = "Anzahl der Aktivitäten pro Tageszeit und Monat (2023)",
#     subtitle = "Aufgeteilt nach Morgen, Mittag, Nachmittag, Abend und Nacht",
#     x = "Tageszeit",
#     y = "Anzahl der Aktivitäten",
#     fill = "Monat"
#   ) +
#   theme_minimal() +
```

```
#   theme(axis.text.x = element_text(angle = 45, hjust = 1))
#
#
#
# library(dplyr)
# library(ggplot2)
# library(lubridate)
#
# # Hilfsfunktion zur Klassifizierung der Tageszeit
# time_of_day <- function(hour) {
#   case_when(
#     hour >= 6 & hour < 10 ~ "Morgen (06-10 Uhr)",
#     hour >= 10 & hour < 14 ~ "Mittag (10-14 Uhr)",
#     hour >= 14 & hour < 18 ~ "Nachmittag (14-18 Uhr)",
#     hour >= 18 & hour < 22 ~ "Abend (18-22 Uhr)",
#     TRUE ~ "Nacht (22-06 Uhr)"
#   )
# }
#
# # Daten für A im April 2024 vorbereiten
# As_data <- records_N %>%
#   mutate(
#     timestamp = as.POSIXct(timestamp, format = "%Y-%m-%dT%H:%M:%S", tz = "UTC"),
#     year = year(timestamp),
#     month = month(timestamp),
#     day = day(timestamp),
#     hour = hour(timestamp),
#     period = time_of_day(hour),
#     next_timestamp = lead(timestamp),
#     duration = as.numeric(difftime(next_timestamp, timestamp, units = "hours")),
#     kilometers = duration * 0.08  # Annahme: 0.08km pro Stunde
#   ) %>%
#   filter(year == 2024 & month == 4 & timestamp >= as.POSIXct("2024-04-01", tz = "UTC") & ti
#   arrange(timestamp) %>%
#   filter(!is.na(duration))
#
# # Reihenfolge der Tageszeiten definieren
# levels_order <- c("Morgen (06-10 Uhr)", "Mittag (10-14 Uhr)", "Nachmittag (14-18 Uhr)", "Al
# As_data$period <- factor(As_data$period, levels = levels_order)
#
# # Datenanalyse für die Anzahl der Aufzeichnungen nach Tageszeit und Tag
# day_period_record_count <- As_data %>%
```

```
#   group_by(day, period) %>%
#   summarize(record_count = n(), .groups = 'drop')
#
# # Daten überprüfen
# print(day_period_record_count)
#
# # Visualisierung der Aktivitätenanzahl pro Tageszeit und Tag im April 2024 als Flächendiagr
# ggplot(day_period_record_count, aes(x = day, y = record_count, fill = period, group = perio
#   geom_area(position = 'stack', alpha = 0.6) +
#   scale_fill_brewer(palette = "Set3") +
#   labs(
#     title = "Anzahl der Aktivitäten pro Tageszeit und Tag (April 2024)",
#     subtitle = "Aufgeteilt nach Morgen, Mittag, Nachmittag, Abend und Nacht",
#     x = "Tag",
#     y = "Anzahl der Aktivitäten",
#     fill = "Tageszeit"
#   ) +
#   theme_minimal() +
#   theme(axis.text.x = element_text(angle = 45, hjust = 1))
#
# #-------------------------------------------------------------------------------------------
# #-------------------------------------------------------------------------------------------
#
# # B:
#
# library(dplyr)
# library(ggplot2)
# library(lubridate)
#
# # Hilfsfunktion zur Klassifizierung der Tageszeit
# time_of_day <- function(hour) {
#   case_when(
#     hour >= 6 & hour < 10 ~ "Morgen (06-10 Uhr)",
#     hour >= 10 & hour < 14 ~ "Mittag (10-14 Uhr)",
#     hour >= 14 & hour < 18 ~ "Nachmittag (14-18 Uhr)",
#     hour >= 18 & hour < 22 ~ "Abend (18-22 Uhr)",
#     TRUE ~ "Nacht (22-06 Uhr)"
#   )
# }
#
# # Daten für B im April 2024 vorbereiten
# Bs_data <- records_S %>%
```

```
#   mutate(
#     timestamp = as.POSIXct(timestamp, format = "%Y-%m-%dT%H:%M:%S", tz = "UTC"),
#     year = year(timestamp),
#     month = month(timestamp),
#     day = day(timestamp),
#     hour = hour(timestamp),
#     period = time_of_day(hour),
#     next_timestamp = lead(timestamp),
#     duration = as.numeric(difftime(next_timestamp, timestamp, units = "hours")),
#     kilometers = duration * 0.08  # Annahme: 0.08km pro Stunde
#   ) %>%
#   filter(year == 2024 & month == 4 & timestamp >= as.POSIXct("2024-04-01", tz = "UTC") & ti
#   arrange(timestamp) %>%
#   filter(!is.na(duration))
#
# # Reihenfolge der Tageszeiten definieren
# levels_order <- c("Morgen (06-10 Uhr)", "Mittag (10-14 Uhr)", "Nachmittag (14-18 Uhr)", "Al
# Bs_data$period <- factor(Bs_data$period, levels = levels_order)
#
# # Datenanalyse für die Anzahl der Aufzeichnungen nach Tageszeit und Tag
# day_period_record_count <- Bs_data %>%
#   group_by(day, period) %>%
#   summarize(record_count = n(), .groups = 'drop')
#
# # Daten überprüfen
# print(day_period_record_count)
#
# # Visualisierung der Aktivitätenanzahl pro Tageszeit und Tag im April 2024 als Flächendiagr
# ggplot(day_period_record_count, aes(x = day, y = record_count, fill = period, group = perio
#   geom_area(position = 'stack', alpha = 0.6) +
#   scale_fill_brewer(palette = "Set3") +
#   labs(
#     title = "Anzahl der Aktivitäten pro Tageszeit und Tag B (April 2024)",
#     subtitle = "Aufgeteilt nach Morgen, Mittag, Nachmittag, Abend und Nacht",
#     x = "Tag",
#     y = "Anzahl der Aktivitäten",
#     fill = "Tageszeit"
#   ) +
#   theme_minimal() +
#   theme(axis.text.x = element_text(angle = 45, hjust = 1))
#
# #_____________________________________________________________________________________________
```

```
# #_____________________________________________________________________________________
#
# # Reihenfolge der Tageszeiten definieren
# levels_order <- c("Morgen (06-10 Uhr)", "Mittag (10-14 Uhr)", "Nachmittag (14-18 Uhr)", "Al
# As_data$period <- factor(As_data$period, levels = levels_order)
# Bs_data$period <- factor(Bs_data$period, levels = levels_order)
#
# # Datenanalyse für die Anzahl der Aufzeichnungen nach Tageszeit und Tag für beide Datensätz
# A_day_period_record_count <- As_data %>%
#   group_by(day, period) %>%
#   summarize(record_count = n(), .groups = 'drop') %>%
#   mutate(person = "A")
#
# B_day_period_record_count <- Bs_data %>%
#   group_by(day, period) %>%
#   summarize(record_count = n(), .groups = 'drop') %>%
#   mutate(person = "B")
#
# # Kombinieren der Datensätze
# combined_data <- bind_rows(A_day_period_record_count, B_day_period_record_count)
#
# # Visualisierung der Aktivitätenanzahl pro Tageszeit und Tag im April 2024 für beide Perso
# ggplot(combined_data, aes(x = day, y = record_count, fill = period, group = period)) +
#   geom_area(position = 'stack', alpha = 0.6) +
#   facet_wrap(~ person) +
#   scale_fill_brewer(palette = "Set3") +
#   labs(
#     title = "Anzahl der Aktivitäten pro Tageszeit und Tag (April 2024)",
#     subtitle = "Vergleich zwischen A und B",
#     x = "Tag",
#     y = "Anzahl der Aktivitäten",
#     fill = "Tageszeit"
#   ) +
#   theme_minimal() +
#   theme(axis.text.x = element_text(angle = 45, hjust = 1))
#
#
# ____________________________----------------------------
#
#
# library(dplyr)
# library(ggplot2)
```

```
#
# # A
#
# # Funktion, um Monate in Jahreszeiten umzuwandeln
# month_to_season <- function(month) {
#   ifelse(month %in% c(12, 1, 2), "Winter",
#          ifelse(month %in% c(3, 4, 5), "Frühling",
#                 ifelse(month %in% c(6, 7, 8), "Sommer", "Herbst")))
# }
#
# # Daten für A von Januar 2022 bis heute vorbereiten
# As_data <- records_N %>%
#   mutate(
#     timestamp = as.POSIXct(timestamp, format = "%Y-%m-%dT%H:%M:%S", tz = "UTC"),
#     month = month(timestamp),
#     season = month_to_season(month)
#   ) %>%
#   filter(year(timestamp) >= 2022 & timestamp >= as.POSIXct("2022-01-01", tz = "UTC")) %>%
#   arrange(timestamp) %>%
#   mutate(
#     next_timestamp = lead(timestamp),
#     duration = as.numeric(difftime(next_timestamp, timestamp, units = "hours"))
#   ) %>%
#   # Letzten Eintrag entfernen, da keine nachfolgende Zeit
#   filter(!is.na(duration))
#
# # Daten nach Jahreszeit gruppieren und Anzahl der Datensätze pro Saison zählen
# seasonal_data_count <- As_data %>%
#   group_by(season) %>%
#   summarize(count = n(), .groups = 'drop')
#
# # Median der Gesamtzählungen für jede Jahreszeit berechnen
# seasonal_median_count <- seasonal_data_count %>%
#   summarize(median_count = median(count), .groups = 'drop')
#
# # Säulendiagramm erstellen zur Anzeige der Anzahl der Datensätze pro Jahreszeit
# ggplot(seasonal_data_count, aes(x = season, y = count, fill = season)) +
#   geom_col() +
#   geom_errorbar(aes(ymin = count, ymax = count), width = .2, position = position_dodge(0.9)
#   geom_hline(data = seasonal_median_count, aes(yintercept = median_count),
#              color = "red", linetype = "dashed", linewidth = 1) +
#   scale_fill_brewer(palette = "Set1", name = "Jahreszeit") +
```

```
#   labs(
#     title = "Anzahl der Datensätze von A nach Jahreszeit",
#     subtitle = "Datenanalyse ab Januar 2022",
#     y = "Anzahl der Datensätze",
#     x = "Jahreszeit",
#     caption = "Jahreszeiten: Frühling (März-Mai), Sommer (Juni-Aug), Herbst (Sept-Nov), Wi
#   ) +
#   theme_minimal() +
#   theme(axis.text.x = element_text(angle = 45, hjust = 1))
# #_____________________________________________________________________________________
# # nach Zeitdauer
#
# # Daten für A von Januar 2022 bis heute vorbereiten
# As_data <- records_N %>%
#   mutate(
#     timestamp = as.POSIXct(timestamp, format = "%Y-%m-%dT%H:%M:%S", tz = "UTC")
#   ) %>%
#   filter(year(timestamp) >= 2022 & timestamp >= as.POSIXct("2022-01-01", tz = "UTC")) %>%
#   mutate(
#     month = month(timestamp),
#     season = month_to_season(month)
#   ) %>%
#   arrange(timestamp) %>%
#   mutate(
#     next_timestamp = lead(timestamp),
#     duration = as.numeric(difftime(next_timestamp, timestamp, units = "hours"))
#   ) %>%
#   # Letzten Eintrag entfernen, da keine nachfolgende Zeit
#   filter(!is.na(duration))
#
# # Gesamtdauer pro Jahreszeit berechnen
# total_duration_per_season <- As_data %>%
#   group_by(season) %>%
#   summarize(total_duration = sum(duration), .groups = 'drop')
#
# # Plot der Gesamtdauer der Bewegung pro Jahreszeit in Stunden
# ggplot(total_duration_per_season, aes(x = season, y = total_duration, fill = season)) +
#   geom_col() +
#   labs(
#     title = "Gesamtdauer der Bewegung von A nach Jahreszeit in Stunden",
#     subtitle = "Datenanalyse ab Januar 2022",
#     y = "Gesamtdauer (Stunden)",
```

```
#     x = "Jahreszeit",
#     caption = "Jahreszeiten: Frühling (März-Mai), Sommer (Juni-Aug), Herbst (Sept-Nov), Wi
#   ) +
#   theme_minimal() +
#   theme(axis.text.x = element_text(angle = 45, hjust = 1))
#
# #_____________________________________________________________________________________
# # Diagramme kombinieren
#
# # Funktion, um Monate in Jahreszeiten umzuwandeln
# month_to_season <- function(month) {
#   ifelse(month %in% c(12, 1, 2), "Winter",
#          ifelse(month %in% c(3, 4, 5), "Frühling",
#                 ifelse(month %in% c(6, 7, 8), "Sommer", "Herbst")))
# }
#
# # Daten für A von Januar 2022 bis heute vorbereiten
# As_data <- records_N %>%
#   mutate(
#     timestamp = as.POSIXct(timestamp, format = "%Y-%m-%dT%H:%M:%S", tz = "UTC"),
#     month = month(timestamp),
#     season = month_to_season(month)
#   ) %>%
#   filter(year(timestamp) >= 2022 & timestamp >= as.POSIXct("2022-01-01", tz = "UTC")) %>%
#   arrange(timestamp) %>%
#   mutate(
#     next_timestamp = lead(timestamp),
#     duration = as.numeric(difftime(next_timestamp, timestamp, units = "hours"))
#   ) %>%
#   # Letzten Eintrag entfernen, da keine nachfolgende Zeit
#   filter(!is.na(duration))
#
# # Daten nach Jahreszeit gruppieren und Anzahl der Datensätze pro Saison zählen
# seasonal_data_count <- As_data %>%
#   group_by(season) %>%
#   summarize(count = n(), .groups = 'drop')
#
# # Gesamtdauer pro Jahreszeit berechnen
# total_duration_per_season <- As_data %>%
#   group_by(season) %>%
#   summarize(total_duration = sum(duration), .groups = 'drop')
#
```

```
# # Normalisierung der Werte
# max_count <- max(seasonal_data_count$count)
# max_duration <- max(total_duration_per_season$total_duration)
# seasonal_data_count$normalized_count <- seasonal_data_count$count / max_count
# total_duration_per_season$normalized_duration <- total_duration_per_season$total_duration /
#
# # Überlagerte Visualisierung
# ggplot() +
#   geom_col(data = seasonal_data_count, aes(x = season, y = normalized_count, fill = "Anzahl
#   geom_line(data = total_duration_per_season, aes(x = season, y = normalized_duration, grou
#   scale_fill_manual(name = "Legende", values = c("Anzahl der Datensätze" = "blue"), labels
#   scale_color_manual(name = "", values = c("Gesamtdauer" = "red"), labels = c("Gesamtdauer
#   labs(title = "Vergleich von Datensätzen und Bewegungsdauer",
#        subtitle = "Jahreszeitenanalyse ab Januar 2022",
#        x = "Jahreszeit", y = "Normalisierte Werte") +
#   theme_minimal() +
#   theme(legend.position = "bottom")
#
#
# library(ggplot2)
# library(dplyr)
#
# # Berechne die Korrelation
# correlation_result <- cor(seasonal_data_count$normalized_count, total_duration_per_season$n
#
# # Erstelle eine kombinierte Visualisierung
# combined_plot <- ggplot() +
#   geom_col(data = seasonal_data_count, aes(x = season, y = normalized_count, fill = "Anzahl
#   geom_line(data = total_duration_per_season, aes(x = season, y = normalized_duration, grou
#   geom_text(aes(x = 3, y = 1, label = sprintf("Korrelation: %.2f", correlation_result)), pa
#   scale_fill_manual(name = "Legende", values = c("Anzahl der Datensätze" = "blue"), labels
#   scale_color_manual(name = "", values = c("Gesamtdauer" = "red"), labels = c("Gesamtdauer
#   labs(title = "Vergleich von Datensätzen und Bewegungsdauer",
#        subtitle = "Jahreszeitenanalyse ab Januar 2022",
#        x = "Jahreszeit", y = "Normalisierte Werte",
#        caption = paste("Korrelation zwischen Datensätzen und Gesamtdauer: ", sprintf("%.2f"
#                        "\nEine Korrelation nahe 1 zeigt eine starke positive Beziehung."))
#   theme_minimal() +
#   theme(legend.position = "bottom", plot.caption = element_text(hjust = 0))
#
# print(combined_plot)
#
```

```
# #_____________________________________________________________________________________________
# # Analyse per Montate einzeln:
#
# library(dplyr)
# library(ggplot2)
#
# # Funktion, um Monate in Jahreszeiten umzuwandeln
# month_to_season <- function(month) {
#   ifelse(month %in% c(12, 1, 2), "Winter",
#          ifelse(month %in% c(3, 4, 5), "Frühling",
#                 ifelse(month %in% c(6, 7, 8), "Sommer", "Herbst")))
# }
#
# # Daten für A von Januar 2022 bis heute vorbereiten
# As_data <- records_N %>%
#   mutate(
#     timestamp = as.POSIXct(timestamp, format = "%Y-%m-%dT%H:%M:%S", tz = "UTC"),
#     month = format(timestamp, "%Y-%m"), # Erstelle eine neue Spalte für Monat im Format Ja
#     season = month_to_season(month(timestamp))
#   ) %>%
#   filter(timestamp >= as.POSIXct("2022-01-01", tz = "UTC")) %>%
#   arrange(timestamp) %>%
#   mutate(
#     next_timestamp = lead(timestamp),
#     duration = as.numeric(difftime(next_timestamp, timestamp, units = "hours"))
#   ) %>%
#   filter(!is.na(duration))
#
# # Daten nach Monat gruppieren und Anzahl der Datensätze pro Monat sowie Gesamtdauer zählen
# monthly_data_summary <- As_data %>%
#   group_by(month) %>%
#   summarize(
#     count = n(),
#     total_duration = sum(duration),
#     .groups = 'drop'
#   )
#
# # Berechne das Maximum für Skalierungszwecke
# max_count <- max(monthly_data_summary$count)
# max_duration <- max(monthly_data_summary$total_duration)
#
# # Visualisierung der Daten nach Monat
```

```
# ggplot(monthly_data_summary, aes(x = month)) +
#   geom_col(aes(y = count), fill = "blue", alpha = 0.7) +
#   geom_line(aes(y = total_duration / max_duration * max_count), color = "red", group = 1) +
#   scale_y_continuous(
#     name = "Anzahl der Datensätze",
#     sec.axis = sec_axis(~ . / max_count * max_duration, name = "Gesamtdauer (Stunden)")
#   ) +
#   labs(
#     title = "Monatliche Analyse der Bewegungsdaten von A",
#     subtitle = "Von Januar 2022 bis heute",
#     x = "Monat",
#     caption = "Blaue Balken zeigen die Anzahl der Datensätze, rote Linie zeigt die skaliert
#   ) +
#   theme_minimal() +
#   theme(axis.text.x = element_text(angle = 45, hjust = 1))
#
# # Korrelation
# library(dplyr)
# library(ggplot2)
#
# # Daten für A von Januar 2022 bis heute vorbereiten
# As_data <- records_N %>%
#   mutate(
#     timestamp = as.POSIXct(timestamp, format = "%Y-%m-%dT%H:%M:%S", tz = "UTC"),
#     month = format(timestamp, "%Y-%m"), # Erstelle eine neue Spalte für Monat im Format Jah
#     season = month_to_season(month(timestamp))
#   ) %>%
#   filter(timestamp >= as.POSIXct("2022-01-01", tz = "UTC")) %>%
#   arrange(timestamp) %>%
#   mutate(
#     next_timestamp = lead(timestamp),
#     duration = as.numeric(difftime(next_timestamp, timestamp, units = "hours"))
#   ) %>%
#   filter(!is.na(duration))
#
# # Daten nach Monat gruppieren und Anzahl der Datensätze pro Monat sowie Gesamtdauer zählen
# monthly_data_summary <- As_data %>%
#   group_by(month) %>%
#   summarize(
#     count = n(),
#     total_duration = sum(duration),
#     .groups = 'drop'
```

```
#   )
#
# # Berechne die Korrelation zwischen der Anzahl der Datensätze und der Gesamtdauer
# correlation_result <- cor(monthly_data_summary$count, monthly_data_summary$total_duration)
#
# # Berechne das Maximum für Skalierungszwecke
# max_count <- max(monthly_data_summary$count)
# max_duration <- max(monthly_data_summary$total_duration)
#
# # Visualisierung der Daten nach Monat
# ggplot(monthly_data_summary, aes(x = month)) +
#   geom_col(aes(y = count), fill = "blue", alpha = 0.7) +
#   geom_line(aes(y = total_duration / max_duration * max_count), color = "red", group = 1) +
#   scale_y_continuous(
#     name = "Anzahl der Datensätze",
#     sec.axis = sec_axis(~ . / max_count * max_duration, name = "Gesamtdauer (Stunden)")
#   ) +
#   geom_text(aes(x = 1, y = max_count, label = sprintf("Korrelation: %.2f", correlation_resu
#   labs(
#     title = "Monatliche Analyse der Bewegungsdaten von A",
#     subtitle = "Von Januar 2022 bis heute",
#     x = "Monat",
#     caption = "Blaue Balken zeigen die Anzahl der Datensätze, rote Linie zeigt die skaliert
#   ) +
#   theme_minimal() +
#   theme(axis.text.x = element_text(angle = 45, hjust = 1))
#
#
#
# #-------------------------------------------------------------------------------------------
# #-------------------------------------------------------------------------------------------
#
# # B
#
# library(dplyr)
# library(ggplot2)
#
# # Funktion, um Monate in Jahreszeiten umzuwandeln
# month_to_season <- function(month) {
#   ifelse(month %in% c(12, 1, 2), "Winter",
#          ifelse(month %in% c(3, 4, 5), "Frühling",
#                 ifelse(month %in% c(6, 7, 8), "Sommer", "Herbst")))
```

```
# }
#
# # Daten für B von Januar 2022 bis heute vorbereiten
# Bs_data <- records_S %>%
#   mutate(
#     timestamp = as.POSIXct(timestamp, format = "%Y-%m-%dT%H:%M:%S", tz = "UTC"),
#     month = month(timestamp),
#     season = month_to_season(month)
#   ) %>%
#   filter(year(timestamp) >= 2022 & timestamp >= as.POSIXct("2022-01-01", tz = "UTC")) %>%
#   arrange(timestamp) %>%
#   mutate(
#     next_timestamp = lead(timestamp),
#     duration = as.numeric(difftime(next_timestamp, timestamp, units = "hours"))
#   ) %>%
#   # Letzten Eintrag entfernen, da keine nachfolgende Zeit
#   filter(!is.na(duration))
#
# # Daten nach Jahreszeit gruppieren und Anzahl der Datensätze pro Saison zählen
# seasonal_data_count <- Bs_data %>%
#   group_by(season) %>%
#   summarize(count = n(), .groups = 'drop')
#
# # Median der Gesamtzählungen für jede Jahreszeit berechnen
# seasonal_median_count <- seasonal_data_count %>%
#   summarize(median_count = median(count), .groups = 'drop')
#
# # Säulendiagramm erstellen zur Anzeige der Anzahl der Datensätze pro Jahreszeit
# ggplot(seasonal_data_count, aes(x = season, y = count, fill = season)) +
#   geom_col() +
#   geom_errorbar(aes(ymin = count, ymax = count), width = .2, position = position_dodge(0.9)
#   geom_hline(data = seasonal_median_count, aes(yintercept = median_count),
#              color = "red", linetype = "dashed", linewidth = 1) +
#   scale_fill_brewer(palette = "Set1", name = "Jahreszeit") +
#   labs(
#     title = "Anzahl der Datensätze von B nach Jahreszeit",
#     subtitle = "Datenanalyse ab Januar 2022",
#     y = "Anzahl der Datensätze",
#     x = "Jahreszeit",
#     caption = "Jahreszeiten: Frühling (März-Mai), Sommer (Juni-Aug), Herbst (Sept-Nov), Wi
#   ) +
#   theme_minimal() +
```

```
#   theme(axis.text.x = element_text(angle = 45, hjust = 1))
#
# # Gesamtdauer pro Jahreszeit berechnen
# total_duration_per_season <- Bs_data %>%
#   group_by(season) %>%
#   summarize(total_duration = sum(duration), .groups = 'drop')
#
# # Plot der Gesamtdauer der Bewegung pro Jahreszeit in Stunden
# ggplot(total_duration_per_season, aes(x = season, y = total_duration, fill = season)) +
#   geom_col() +
#   labs(
#     title = "Gesamtdauer der Bewegung von B nach Jahreszeit in Stunden",
#     subtitle = "Datenanalyse ab Januar 2022",
#     y = "Gesamtdauer (Stunden)",
#     x = "Jahreszeit",
#     caption = "Jahreszeiten: Frühling (März-Mai), Sommer (Juni-Aug), Herbst (Sept-Nov), Wi
#   ) +
#   theme_minimal() +
#   theme(axis.text.x = element_text(angle = 45, hjust = 1))
#
# # Analyse per Monate einzeln:
# # Daten nach Monat gruppieren und Anzahl der Datensätze pro Monat sowie Gesamtdauer zählen
# monthly_data_summary <- Bs_data %>%
#   group_by(month) %>%
#   summarize(
#     count = n(),
#     total_duration = sum(duration),
#     .groups = 'drop'
#   )
#
# # Berechne die Korrelation zwischen der Anzahl der Datensätze und der Gesamtdauer
# correlation_result <- cor(monthly_data_summary$count, monthly_data_summary$total_duration)
#
# # Berechne das Maximum für Skalierungszwecke
# max_count <- max(monthly_data_summary$count)
# max_duration <- max(monthly_data_summary$total_duration)
#
# # Visualisierung der Daten nach Monat
# ggplot(monthly_data_summary, aes(x = month)) +
#   geom_col(aes(y = count), fill = "blue", alpha = 0.7) +
#   geom_line(aes(y = total_duration / max_duration * max_count), color = "red", group = 1) +
#   scale_y_continuous(
```

```
#     name = "Anzahl der Datensätze",
#     sec.axis = sec_axis(~ . / max_count * max_duration, name = "Gesamtdauer (Stunden)")
#   ) +
#   geom_text(aes(x = 1, y = max_count, label = sprintf("Korrelation: %.2f", correlation_resu
#   labs(
#     title = "Monatliche Analyse der Bewegungsdaten von B",
#     subtitle = "Von Januar 2022 bis heute",
#     x = "Monat",
#     caption = "Blaue Balken zeigen die Anzahl der Datensätze, rote Linie zeigt die skaliert
#   ) +
#   theme_minimal() +
#   theme(axis.text.x = element_text(angle = 45, hjust = 1))
#
# #-------------------------------------------------------------------------------------------
# #-------------------------------------------------------------------------------------------
#
# # Ueberlagerung
#
# library(dplyr)
# library(ggplot2)
#
# # Funktion, um Monate in Jahreszeiten umzuwandeln
# month_to_season <- function(month) {
#   ifelse(month %in% c(12, 1, 2), "Winter",
#          ifelse(month %in% c(3, 4, 5), "Frühling",
#                 ifelse(month %in% c(6, 7, 8), "Sommer", "Herbst")))
# }
#
# # Spezifiziere den Zeitraum für die Analyse
# start_date <- as.POSIXct("2024-04-07T17:58:12.052Z", tz = "UTC")
# end_date <- as.POSIXct("2024-05-15T21:46:26.529Z", tz = "UTC")
#
# # Daten für A und B vorbereiten
# As_data <- records_N %>%
#   mutate(
#     Person = "A",
#     timestamp = as.POSIXct(timestamp, format = "%Y-%m-%dT%H:%M:%S", tz = "UTC"),
#     season = month_to_season(month(timestamp)),
#     next_timestamp = lead(timestamp)
#   ) %>%
#   filter(timestamp >= start_date & timestamp <= end_date) %>%
#   mutate(
```

```
#     duration = as.numeric(difftime(next_timestamp, timestamp, units = "hours"))
#   ) %>%
#   filter(!is.na(duration))
#
# Bs_data <- records_S %>%
#   mutate(
#     Person = "B",
#     timestamp = as.POSIXct(timestamp, format = "%Y-%m-%dT%H:%M:%S", tz = "UTC"),
#     season = month_to_season(month(timestamp)),
#     next_timestamp = lead(timestamp)
#   ) %>%
#   filter(timestamp >= start_date & timestamp <= end_date) %>%
#   mutate(
#     duration = as.numeric(difftime(next_timestamp, timestamp, units = "hours"))
#   ) %>%
#   filter(!is.na(duration))
#
# # Daten kombinieren
# combined_data <- bind_rows(As_data, Bs_data)
#
# # Aggregierte Daten vorbereiten
# aggregated_data <- combined_data %>%
#   group_by(Person, season) %>%
#   summarise(
#     count = n(),
#     total_duration = sum(duration, na.rm = TRUE),
#     .groups = 'drop'
#   )
#
# # Visualisierung der Daten
# ggplot(aggregated_data, aes(x = season, y = total_duration, fill = Person)) +
#   geom_col(position = "dodge") +
#   scale_fill_brewer(palette = "Set1", name = "Person") +
#   labs(
#     title = "Vergleich der Aktivitätsdauer nach Jahreszeit und Person im spezifizierten Ze
#     y = "Gesamtdauer (Stunden)",
#     x = "Jahreszeit",
#     caption = "Jahreszeiten: Frühling (März-Mai), Sommer (Juni-Aug), Herbst (Sept-Nov), Wi
#   ) +
#   theme_minimal() +
#   theme(axis.text.x = element_text(angle = 45, hjust = 1))
#
```

```
# library(dplyr)
# library(ggplot2)
# library(tidyr)
#
# # Funktion, um Monate in Jahreszeiten umzuwandeln
# month_to_season <- function(month) {
#   ifelse(month %in% c(12, 1, 2), "Winter",
#          ifelse(month %in% c(3, 4, 5), "Frühling",
#                 ifelse(month %in% c(6, 7, 8), "Sommer", "Herbst")))
# }
#
# # Spezifiziere den Zeitraum für die Analyse
# start_date <- as.POSIXct("2024-04-07T17:58:12.052Z", tz = "UTC")
# end_date <- as.POSIXct("2024-05-15T21:46:26.529Z", tz = "UTC")
#
# # Daten für A und B vorbereiten
# As_data <- records_N %>%
#   mutate(
#     Person = "A",
#     timestamp = as.POSIXct(timestamp, format = "%Y-%m-%dT%H:%M:%S", tz = "UTC"),
#     season = month_to_season(month(timestamp)),
#     next_timestamp = lead(timestamp)
#   ) %>%
#   filter(timestamp >= start_date & timestamp <= end_date) %>%
#   mutate(
#     duration = as.numeric(difftime(next_timestamp, timestamp, units = "hours"))
#   ) %>%
#   filter(!is.na(duration))
#
# Bs_data <- records_S %>%
#   mutate(
#     Person = "B",
#     timestamp = as.POSIXct(timestamp, format = "%Y-%m-%dT%H:%M:%S", tz = "UTC"),
#     season = month_to_season(month(timestamp)),
#     next_timestamp = lead(timestamp)
#   ) %>%
#   filter(timestamp >= start_date & timestamp <= end_date) %>%
#   mutate(
#     duration = as.numeric(difftime(next_timestamp, timestamp, units = "hours"))
#   ) %>%
#   filter(!is.na(duration))
#
```

```
# # Daten kombinieren
# combined_data <- bind_rows(As_data, Bs_data)
#
# # Aggregierte Daten vorbereiten
# aggregated_data <- combined_data %>%
#   group_by(Person, season) %>%
#   summarise(
#     count = n(),
#     total_duration = sum(duration, na.rm = TRUE),
#     .groups = 'drop'
#   )
#
# # Daten für Korrelation vorbereiten
# comparison_data <- aggregated_data %>%
#   pivot_wider(
#     id_cols = season,
#     names_from = Person,
#     values_from = c(total_duration, count),
#     names_sep = "_"
#   )
#
# # Korrelation berechnen, wenn es möglich ist
# if ("A_total_duration" %in% names(comparison_data) && "B_total_duration" %in% names(compari
#   correlation_duration <- cor(comparison_data$A_total_duration, comparison_data$B_total_du
#   correlation_count <- cor(comparison_data$A_count, comparison_data$B_count, use = "complet
#
#   # Ergebnisse ausgeben
#   print(sprintf("Korrelation der Gesamtdauer zwischen A und B: %.2f", correlation_duration)
#   print(sprintf("Korrelation der Anzahl der Datensätze zwischen A und B: %.2f", correlatio
# } else {
#   print("Erforderliche Daten für die Korrelation sind nicht verfügbar.")
#
# # Korrelation berechnen
# correlation_duration <- cor(comparison_data$A_total_duration, comparison_data$B_total_durat
# correlation_count <- cor(comparison_data$A_count, comparison_data$B_count, use = "complete
#
# # Visualisierung der Daten
# ggplot(aggregated_data, aes(x = season, y = total_duration, fill = Person)) +
#   geom_col(position = "dodge") +
#   scale_fill_brewer(palette = "Set1", name = "Person") +
#   labs(
#     title = "Vergleich der Aktivitätsdaten von A und B nach Jahreszeit",
```

```
#     y = "Gesamtdauer (Stunden)",
#     x = "Jahreszeit",
#     caption = sprintf("Korrelation der Gesamtdauer: %.2f, Korrelation der Datensatzanzahl:
#                       correlation_duration, correlation_count)
#   ) +
#   theme_minimal() +
#   theme(axis.text.x = element_text(angle = 45, hjust = 1))
#
# #_____________________________________________________________________________________
#
#
# # Visualisierung der saisonalen Bewegungsspielräume:
#
# # if (!require("tmap")) install.packages("tmap")
# # library(tmap)
# #
# # library(dplyr)
# # library(tmap)
# #
# # # Funktion, um Monate in Jahreszeiten umzuwandeln
# # month_to_season <- function(month) {
# #   ifelse(month %in% c(12, 1, 2), "Winter",
# #          ifelse(month %in% c(3, 4, 5), "Frühling",
# #                 ifelse(month %in% c(6, 7, 8), "Sommer", "Herbst")))
# # }
#
# # Daten für A von Januar 2022 bis heute vorbereiten
# As_data <- records_N %>%
#   mutate(
#     timestamp = as.POSIXct(timestamp, format = "%Y-%m-%dT%H:%M:%S", tz = "UTC"),
#     lat = latitudeE7 / 1e7,  # Umrechnung von E7 Format in reguläre Koordinaten
#     lon = longitudeE7 / 1e7,
#     month = month(timestamp),
#     season = month_to_season(month)
#   ) %>%
#   filter(year(timestamp) >= 2022 & timestamp >= as.POSIXct("2022-01-01", tz = "UTC"))
#
# # Konvertiere die Daten in ein sf-Objekt
# As_sf <- As_data %>%
#   st_as_sf(coords = c("lon", "lat"), crs = 4326, agr = "constant")
#
# # Setze tmap-Modus auf "view" für interaktive Karten
```

```
# tmap_mode("view")
#
# tm <- tm_shape(As_sf) +
#   tm_dots(col = "season", size = 0.1, palette = c("Winter" = "blue", "Frühling" = "green",
#           title = "Saison") +
#   tm_layout(title = "Saisonale Bewegung von A")
#
# print(tm)
```

#Saisonale Trends - Bewegungsraum

```
# # Install necessary libraries if not already installed
# # install.packages("dplyr")
# # install.packages("tidyr")
# # install.packages("ggplot2")
# # install.packages("sf")
# # install.packages("ggspatial")
# # install.packages("scales")
# # install.packages("osmdata")
#
# # Load necessary libraries
# library(dplyr)
# library(tidyr)
# library(ggplot2)
# library(sf)
# library(ggspatial)
# library(scales)
# library(osmdata)
#
# # Assuming you have loaded the data into records_N dataframe
# # records_N <- read.csv("path_to_your_data.csv") # Adjust the path accordingly
#
# # Filter records by timestamp for the specific date
# records_filtered <- records_N %>%
#   filter(grepl("2024-05-24", timestamp))
#
# # Parse the activity column
# records_filtered <- records_filtered %>%
#   mutate(activity = gsub("list\\(list\\(type = c\\(|\\), confidence = c\\(|\\)\\)\\)", "",
#   separate(activity, into = c("activity1", "activity2"), sep = ", confidence = ") %>%
#   separate(activity1, into = c("activity1", "activity1_confidence"), sep = ", ") %>%
#   separate(activity2, into = c("activity2", "activity2_confidence"), sep = ", ")
```

```
#
# # Convert lat/lon from E7 to decimal degrees
# records_filtered <- records_filtered %>%
#   mutate(latitude = latitudeE7 / 1e7,
#          longitude = longitudeE7 / 1e7)
#
# # Convert to spatial dataframe
# sf_records_filtered <- st_as_sf(records_filtered, coords = c("longitude", "latitude"), crs
#
# # Create convex hulls for each activity
# convex_hulls <- sf_records_filtered %>%
#   group_by(activity1) %>%
#   summarize(geometry = st_union(geometry)) %>%
#   st_convex_hull() %>%
#   mutate(area = st_area(.)) # Calculate the area
#
# # Create labels for the legend
# convex_hulls <- convex_hulls %>%
#   mutate(legend_label = paste(activity1, round(as.numeric(area), 2), "m²"))
#
# # Get bounding box for the plot area
# bbox <- st_bbox(sf_records_filtered)
#
# # Get OpenStreetMap data
# osm_map <- opq(bbox = c(bbox["xmin"], bbox["ymin"], bbox["xmax"], bbox["ymax"])) %>%
#   add_osm_feature(key = "highway") %>%
#   osmdata_sf()
#
# # Plot using ggplot2
# ggplot() +
#   geom_sf(data = osm_map$osm_polygons, fill = "grey80", color = NA) +
#   geom_sf(data = convex_hulls, aes(fill = legend_label), color = NA, alpha = 0.5) +
#   geom_point(data = records_filtered, aes(x = longitude, y = latitude, color = activity1),
#   theme_minimal() +
#   labs(title = "Smallest Polygon Areas (Convex Hulls) for Each Activity on 24.05.2024",
#        fill = "Activity and Area",
#        color = "Activity")
```