# Berlin: Generating dot density coordinates for place of origin data

Nicole Dwenger and Orla Mallon

01/06/2021

This script generates dots and their associated coordinates for Berlin's dot density map of place of origin. The data is taken from the national statics of Germany, 2019, Amt für Statistik Berlin-Brandenburg, Statistisches Informationssystem Berlin-Brandenburg (2019), Einwohnerregisterstatistik Berlin.

The countries of origin have been grouped into the following 7 regions:

*Europe, Africa, North America, South America, Asia, Oceania and Unknown.*

**Density ratio: 1 dot = 100 people**

## 1. Preparation

### 1.1. Packages

```
# Loading up the packages
library(maptools)
library(rgeos)
library(tidyverse)
library(rgdal)
library(ggthemes)
library(sf)
library(dplyr)
library(readxl)
```

### 1.2 Load in the Shapefiles

Data downloaded from: https://daten.odis-berlin.de/de/dataset/lor__planungsgraeume/

```
# --- Load the data with the Bezirke, Planungsraeume, and geometries ---
berlin_geo = st_read("berlin_planungsraeme/lor_planungsraeume.shp") %>%
  # Selecting the relevant dataframes
  dplyr::select("B_name" = BEZIRKSNAM,
                "P_name" = PLANUNGSRA,
                "geometry" = geometry)
```

```
## Reading layer 'lor_planungsraeume' from data source '/Users/Orla/Desktop/Shiny/berlin_planungsraeme/l
## Simple feature collection with 448 features and 6 fields
## geometry type:  MULTIPOLYGON
## dimension:      XY
## bbox:           xmin: 370000.7 ymin: 5799520 xmax: 415785.3 ymax: 5837259
## CRS:            25833
```

```
# --- Fix name columns to character variables ---
berlin_geo$P_name <- as.character(berlin_geo$P_name)
berlin_geo$B_name <- as.character(berlin_geo$B_name)
```

### 1.3 Load and clean the data

The data used can be downloaded from https://www.statistik-berlin-brandenburg.de/webapi/jsf/tableView/tableView.xhtml

**P_name** = 'Planungsraum': the smaller geographical regions within the Bezirke (count = 448)

**B_name** = 'Bezirke': the larger districts of Berlin (count = 12)

```
# --- Load the place of origin data ---
berlin_POO_raw = read_excel("berlin/raw_data/country_of_origin/berlin_place_of_origin.xlsx", range = "A
```

```
## New names:
## * `` -> ...2
```

```
# --- Remove redundant rows (first 2) and columns (first 1) ---
berlin_POO = berlin_POO_raw[-c(1,2),-1]

# --- Rename the first column (Planungsraum = smaller geographical area name) ---
colnames(berlin_POO)[1] = "Planungsraum"

# --- Split the Planungsraum data into the number and name for later matching ---
berlin_POO = separate(berlin_POO, Planungsraum, into = c("P_number", "P_name"), sep = "\\s", extra = "me

# --- Make the numbers numeric ---
berlin_POO[,c(3:26)] <- sapply(berlin_POO[,c(3:26)],as.numeric)

# --- Fix Planungsraum names to match with the berlin_geo dataset ---
berlin_POO$P_name[berlin_POO$P_name == "Siedlung Kämmereiheide (bis 2018 Allende II)"] = "Siedlung Kämme
berlin_POO$P_name[berlin_POO$P_name == "Allende II (ab 2019)"] = "Allende II"

# --- Two Planungsraum have the same name but are from different "Bezirke". Here, they'll be renamed --
berlin_POO$P_name[berlin_POO$P_number == "04030417"] = "Schloßstraße (C-W)"
berlin_POO$P_name[berlin_POO$P_number == "06010102"] = "Schloßstraße (S-Z)"

# --- Rename these Planungsraum names in the berlin_geo dataset also to match ---
berlin_geo$P_name[berlin_geo$B_name == "Charlottenburg-Wilmersdorf" & berlin_geo$P_name == "Schloßstraße
berlin_geo$P_name[berlin_geo$B_name == "Steglitz-Zehlendorf" & berlin_geo$P_name == "Schloßstraße"] = "S
```

### 1.4 Group the countries of origin into larger groups

Regions of origin used: Europe, Africa, North America, South America, Asia, Oceania, and Unknown.

```
# --- Join the columns into larger categories in a new dataframe ---
berlin_all = left_join(berlin_POO, berlin_geo, by = "P_name")
berlin_all$europe = rowSums(berlin_all[, c(3:4)], na.rm=T)
berlin_all$africa = rowSums(berlin_all[, c(5:10)], na.rm=T)
berlin_all$north_america = rowSums(berlin_all[,11], na.rm=T)
```

```r
berlin_all$south_america = rowSums(berlin_all[, c(12:14)], na.rm=T)
berlin_all$asia = rowSums(berlin_all[, c(15:18)], na.rm=T)
berlin_all$oceania = rowSums(berlin_all[, c(19:21)], na.rm=T)
berlin_all$unknown = rowSums(berlin_all[, c(22:26)], na.rm=T)

# --- Create a new dataframe with only the categorised regions and relevant information ---
berlin_origin_data = berlin_all %>%
  dplyr::select(P_name, P_number, B_name, europe, africa, north_america, south_america, asia, oceania, 
```

### 1.5 Calculate the country of origin percentages per Bezirke

```r
# --- First calculate the total population per Planungsraum ---
berlin_origin_data <- berlin_origin_data %>% mutate(P_population = rowSums(berlin_origin_data[4:10]))

# --- Use the Planungsraum population to calculate total population per Bezirk ---
berlin_origin_data <- berlin_origin_data %>% group_by(B_name) %>% mutate(B_population = sum(P_populatio

# --- Then calculate the percentages per origin, per region ---
berlin_percentages <- berlin_origin_data %>%
  group_by(B_name) %>%
  summarise(origin_perc_africa = sum(africa/B_population), origin_perc_asia = sum(asia/B_population),
            origin_perc_europe = sum(europe/B_population), origin_perc_north_america =
            sum(north_america/B_population), origin_perc_oceania = sum(oceania/ B_population),
            origin_perc_south_america = sum(south_america/B_population),
            origin_perc_unknown = sum(unknown/ B_population)) %>% mutate_if(is.numeric, ~round(., 2)*10
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

## 2. Generating the Dots and their Coordinates

### 2.1 Defining functions

**randomround** = A function which helps to soften the threshold of 100 people = 1 dot. This function randomly rounds the data so that if a group has over 50 people, but is not quite 100, sometimes it will be generated as a dot as the number will be randomly rounded up instead of down. This is preferred over a simple rounding or hard threshold, which can often lead to systematic bias against minority groups.

credit: Jens von Bergmann https://github.com/mountainMath/dotdensity/blob/master/R/dot-density.R

**count_dots_B** = A function which generates how many dots should be created for each region of origin, in each Planungsraum. The output is dataframe with a column for each region of origin and a row for each Planungsraum, holding the number of dots which should be plotted for each.

**generate_dots_B** = A function which generates random coordinates for the dot, within the specified geographical area.Here, we use the smallest area shapefiles available, meaning that even though the coordinates are randomly generated, they should reflect the ethnicity of the area to an accurate degree.

```r
# --- randomround ---
#Function for rounding data to soften density threshold
random_round <- function(x) {
    v=as.integer(x)
    r=x-v
    test=runif(length(r), 0.0, 1.0)
    add=rep(as.integer(0),length(r))
    add[r>test] <- as.integer(1)
    value=v+add
    ifelse(is.na(value) | value<0,0,value)
    return(value)
}

# --- count_dots ---
# A function which takes the borough sf object and counts how many dots should be created
# A safety function has been added for when no dots are created for oceania, due to low populations (li
count_dots_B <- function(x) {
  district_dots = as.data.frame(x) %>%
    select(europe:unknown) %>%
    mutate_all(funs(. / 100)) %>%
    mutate_all(random_round)
  number = sample(1:nrow(district_dots), 1)
  district_dots$oceania[number] = 1
  return(district_dots)
}


# --- generate_dots_B ---
# A function which determines the spatial coordinates of the dots
generate_dots_B <- function(district_dots, sf_object) {
  district_coords <- map_df(names(district_dots),
                            ~ st_sample(sf_object, size = district_dots[,.x], type = "random") %>%
                              st_cast("POINT") %>%
                              st_coordinates() %>%
                              as_tibble() %>%
                              setNames(c("lon","lat")) %>%
                              mutate(country_of_orig = .x)) %>%
                       slice(sample(1:n()))
                       return(district_coords)
}
```

### 2.2 Generating the dot coordinates

A loop to go through each Bezirk, count the number of dots and generate the coordinates accordingly

```r
# --- Create empty df to append into ---
berlin_df = data.frame()

# --- Make berlin_origin_data into an sf object ---
berlin_origin_data <- berlin_origin_data %>% st_as_sf()

# --- to run through each Bezirke and genrate the dot coordinates ---
for(Bezirk in unique(berlin_origin_data$B_name)){
  Bezirk_sf = filter(berlin_origin_data, B_name == Bezirk)
  Bezirk_dots = count_dots_B(Bezirk_sf)
  Bezirk_coords = generate_dots_B(Bezirk_dots, Bezirk_sf)
  Bezirk_coords$name = as.character(Bezirk)
  berlin_df = rbind(berlin_df, Bezirk_coords)
}
```

### 2.3 Transform the coordinates into the 4362 CRS

```
# --- Make dataframe into an sf object ---
berlin_df <- st_as_sf(berlin_df, coords = c("lon", "lat"), crs= 25833)

# --- Transform into the 4326 crs for leaflet ---
berlin_df <- st_transform(berlin_df, crs = 4326)
```

## 3. Save the data

```
# --- Save the spatial object as an RDS ---
saveRDS(berlin_df, file = "berlin/preprocessed_data/berlin_dot_coordinates.RDS")

# --- Save the dataframe with percentages of each region of origin per Bezirke ---
write.csv(berlin_percentages, "berlin/preprocessed_data/berlin_percentages.csv", row.names = F)

# --- Save a csv file of the pre-transformed coordinates ---
origins_csv <- st_drop_geometry(berlin_origin_data)
write.csv(origins_csv, "berlin/preprocessed_data/berlin_origins.csv", row.names = F)
```