London Preprocessing

Nicole Dwenger and Orla Mallon

2021/06/01

Information

This document pre-processing spatial and attribute data of LONDON for the final project of Spatial Analytics. Data for the following variables will be processed in the following:

1. Borough Shapes (spatial, vector)
2. Population and Age data (attribute)
3. Ethnicity (extracted and pre processed in `london_dot_density.rmd`)
4. Places of Worship (extracted and pre processed in `london_osm_culture_religion.Rmd`)
5. Museums, Theatres and Nightlife (extracted and pre processed in `london_osm_culture_religion.Rmd`)
6. Travel Time (spatial, vector and attribute)
7. Crime (attribute)
8. Rent (attribute)
9. Tree cover (spatial, raster)
10. Imperviousness (spatial, raster)

In a last step all pre-processed data will be joined in one data frame.

Loading Libraries

```
# Libraries
library(tidyverse)
library(sf)
library(leaflet)
library(leaflet.extras)
library(htmlwidgets)
library(pacman)
library(ggplot2)
library(readxl)
library(httr)
library(raster)
library(here)

# Setting here for path management
here::here()
```

```
## [1] "/Users/nicoledwenger/Documents/University/6SEMESTER/Spatial Analytics/FINALPROJECT/cdsspatial-pr
```

1. Borough Shapes

Loading shapefiles for the borough boundaries of London, as vector data of polygons. The raw data is in a projected CRS, so we extract centroids of each of the polygons (boroughs). Lastly, we extract the relevant columns, and transform both the coordinates of the polygons and the coordiantes of the centroids in EPSG 4326, as this is the required CRS for Leaflet (which will be used in the shiny app) and calculate the area.

- Data source: https://data.london.gov.uk/dataset/statistical-gis-boundary-files-london

```
# Loading shapefile of London boroughs
london_raw <- st_read(here("london", "raw_data", "boroughs", "London_Borough_Excluding_MHW.shp"))
```

```
## Reading layer ‘London_Borough_Excluding_MHW’ from data source ‘/Users/nicoledwenger/Documents/Univers
## Simple feature collection with 33 features and 7 fields
## Geometry type: MULTIPOLYGON
## Dimension:     XY
## Bounding box:  xmin: 503568.2 ymin: 155850.8 xmax: 561957.5 ymax: 200933.9
## Projected CRS: OSGB 1936 / British National Grid
```

```
# Checking CRS: OSGB 1936 / British National Grid > projected
st_crs(london_raw)$input
```

```
## [1] "OSGB 1936 / British National Grid"
```

```
# Save the crs to use later
crs_london = st_crs(london_raw)

# Getting center point of each polygon
london_raw$center_coords = st_centroid(london_raw$geometry)
# Getting area for each polygon / borough
london_raw$area_m2 = st_area(london_raw$geometry)
# Transforming into ha
london_raw$area_ha = as.numeric(london_raw$area_m2)/10000
# Transforming into km2
london_raw$area_km2 = as.numeric(london_raw$area_m2)/1000000

# Turing the borough geometry into 4326 for Leaflet
london_geo = london_raw %>%
  st_transform(crs = 4326) %>%
  # Selecting relevant columns
  dplyr::select("name" = NAME,
                "area_ha" = area_ha,
                "area_km2" = area_km2,
                "center_coords" = center_coords, # coordinates of centroids
                "geometry" = geometry) # coordinates in 4326 for boroughs, "geometry" of sf object

# Also transforming the center coordinates into 4326, the above step did only transformed the "geometry
london_geo$center_coords = st_transform(london_geo$center_coords, crs = 4326)

# Checking CRS of center coordinates again
st_crs(london_geo$center_coords)$input
```

```
## [1] "EPSG:4326"
```

```
# Checking CRS of boroughs geometry again
st_crs(london_geo$geometry)$input
```

```
## [1] "EPSG:4326"
```

```
# Saving borough names
borough_names = london_geo$name
```

2. Population and Age Data

The aim was to get a mean age, and percentages for different age ranges for each borough. Thus, here we load a data file containing data that was extracted from an interactive dataset (see link below). We calculate the total population by getting the sum of all age-ranges and get percentages by dividing the population of each age range by the total population in the borough. Lastly, we calculate the mean age, by taking the center value of each of the age ranges. This is simplified and comes with some assumptions.

- Data source: https://data.london.gov.uk/dataset/gla-population-projections-custom-age-tables?q=age%20demographics
- Info: Extracted populations from upper projections for 2020 (were same as lower projections)

```
# Loading raw age data
age_raw <- read_excel(here("london", "raw_data", "age", "age_2020.xlsx"))

# Change the column name of borough to be "name"
colnames(age_raw)[1] = "name"

# Change remaining column names and turn values into numeric
for (i in 2:11){
  colnames(age_raw)[i] = paste0("age_", colnames(age_raw)[i])
  age_raw[,i] <- sapply(age_raw[,i], as.numeric)
}

# Calculate total population by getting sum of the row
age_raw$total_population = rowSums(age_raw[,2:11])

# Calculating the percentages of each age group
age_perc = age_raw
# Getting the age groups
ranges = colnames(age_perc[2:11])
# Looping through the age groups and getting the percentage
for (range in ranges){
    percentage = round(age_perc[range]/age_perc["total_population"], 2)
    col_name = paste0("perc_", range)
    age_perc[col_name] = percentage
}

# Calculating the mean age for each borough
age_sumaries = age_perc
# Initializing column
```

```r
age_sumaries$age_mean = NA
# Looping through each row (borough) to get the mean, using the middle value
for(i in 1:nrow(age_sumaries)) {
  mean_age = (age_sumaries[i,2]*5.5 + age_sumaries[i,3]*15.5 + age_sumaries[i,4]*25.5 + age_sumaries[i,5
              age_sumaries[i,6]*45.5 + age_sumaries[i,7]*55.5 + age_sumaries[i,8]*65.5 + age_sumaries[i
              age_sumaries[i,10]*85.5 + age_sumaries[i,11]*95.5)/age_sumaries$total_population[i]
  age_sumaries$age_mean[i] = round(mean_age,2)
}

# Only getting the percentages and mean columns
age = age_sumaries %>%
  dplyr::select(-c("age_0_9":"age_90+"))
# Transforming to numeric
age$age_mean = as.numeric(age$age_mean)
```

3. Ethnicity Data

This data was processed in london_dot_density.rmd. Here only the preprocessed data is loaded and added.

```r
ethnicity = read.csv(here("london", "preprocessed_data", "london_percentages.csv"))
```

4. Places of Worship

This data was extracted from OSM and preprocessd in the script london_osm_culture_religion.Rmd.

```r
# Loading the preprocessed religion count data
religion = read.csv(here("london", "preprocessed_data", "london_religion_counts.csv"))
```

5. Museums, Theatres and Nightlife

This data was extracted from OSM and preprocessd in the script london_osm_culture_religion.Rmd.

```r
# Loading the preprocessed culture count data
culture = read.csv(here("london", "preprocessed_data", "london_culture_counts.csv"))
```

6. Travel Time

Here, we extract transit travel times from Google Maps, using the gmapsdistance package, and the Google Maps Distance Matrix API. We decided to define a set of points of interests of train stations and airports, and calcualte the time it takes to get from the centroids of districts to these POI. To get the data, it requires coordiantes for "origin" and coordinates for "destination" in the format lat+lng, and in the CRS that Google Maps required the coordinates to be in (4326). The data is extracted for using transit, for UCT 7.00 (which is 8.00) in London. The extracted data in seconds is converted to minutes and stored in a dataframe. As it is not allowed to create content with data extracted from Google Maps, the data should not be further used, read more here: https://cloud.google.com/maps-platform/terms/.

- For more info on the gmapsdistance package: https://github.com/rodazuero/gmapsdistance

```r
# Installing gmaps distance
# devtools::install_github("rodazuero/gmapsdistance")
library(gmapsdistance)
# Setting API key (removed for security)
set.api.key("")

# Getting centroids data
centers = london_geo %>%
  dplyr::select(name, center_coords) %>%
  # Dropping borough shape geometry
  st_drop_geometry() %>%
  # Turning into sf, so that centroids become the geometry
  st_as_sf() %>%
  # Mutate empty column
  mutate(origin = NA)

# For each row (centroid), bring the coordinates in the right format
# They should be in the same format as Google Maps data (4326, WGS84)
for (row in 1:nrow(centers)){
  lat = st_coordinates(centers)[row, "Y"]
  lng = st_coordinates(centers)[row, "X"]
  centers$origin[row] = paste0(lat, "+", lng)
}

# Define locations of points of interest, coordinates were extracted manually from GoogleMaps
POI = rbind(c("train_waterloo", 51.503325560796156, -0.11230510142714892),
            c("train_paddington", 51.5168012313166, -0.17693280142677986),
            c("train_kingscross", 51.531991337385136, -0.12346287259050913),
            c("train_liverpoolstreet", 51.51889845379851, -0.08148031676972306),
            c("train_victoria", 51.49537726830846, -0.14388717259162917),
            c("train_londonbridge", 51.504416056960665, -0.08478633026290418),
            c("air_heathrow", 51.47004901261934, -0.45423112847256897),
            c("air_gatwick", 51.1537159154631, -0.1820950880013889),
            c("air_stansted", 51.88591866726579, 0.23890872742011673),
            c("air_londonca", 51.50464333476912, 0.04963601576625392),
            c("air_southend", 51.57031595153878, 0.6924333715894126),
            c("air_luton", 51.879401911519295, -0.3767099014158356))


# Saving the POI as a data frame to use in shiny app
POI = as.data.frame(POI)
# Fixing column names
colnames(POI) = c("id", "lat", "lng")
# Saving coordinates of the points of interest
write.csv(POI, "london/preprocessed_data/london_poi.csv")

# Here we are getting the gmaps travel time data, for a monday morning in May
# For each Point of Interest
for (i in 1:nrow(POI)){
  # Turn the coordniates into the correct format
  dest = paste0(POI[i,2], "+", POI[i,3])
  # Extract values from gmapsdistance (7.00 UCT is 8.00 in London)
  gmaps_values = gmapsdistance(origin = centers$origin, destination = dest,
```

```
                                dep_date = "2021-06-07", dep_time = "07:00:00", mode = "transit")
  # Append to dataframe with column name of time_POI, and divide by 60 to get in minutes
  centers[,paste0("time_", POI[i,1])] = gmaps_values$Time[,2]/60
}

# Remove the ceter coordinnates from the dataframe, to only keep the travel time
travel_time = centers %>% st_drop_geometry() %>% dplyr::select(-origin) %>% as.data.frame()

# Save for knitting
saveRDS(travel_time, "london/preprocessed_data/london_travel_time.rds")
```

```
# Loading the travel time rds
travel_time = readRDS(here("london", "preprocessed_data", "london_travel_time.rds"))
```

7. Crime Data

Crime data was downloaded, and here we simply select the columns with crime data from 2020, and calcualte a sum of all crime types across the months of 2020.

- Data source: https://data.london.gov.uk/dataset/recorded_crime_summary

```
# Downloading data
crime_raw <- read.csv("https://data.london.gov.uk/download/recorded_crime_summary/866c05de-c5cd-454b-8f

# Getting the index of the start columnn (2020,01) and end column (2020, 12)
start = match("X202001", names(crime_raw))
end = match("X202012", names(crime_raw))

# For each row (borough), calculating the total number of crimes in 2020
crime <- crime_raw %>%
  # Getting total number of crimes for the year
  mutate(crime_total = rowSums(crime_raw[start:end])) %>%
  # Grouping by borough
  group_by(name) %>%
  # Getting total crime by borough
  summarise(crime_total = sum(crime_total))
```

8. Rent

Here, we load some rent data of the private rent market in London from 2020, and get the mean values for a 1 bedrooom apartment and a 2 bedroom apartment.

- Source: https://www.ons.gov.uk/peoplepopulationandcommunity/housing/adhocs/12871privaterentalmarketinlondonja

```
# Read raw rent file
rent_raw <- read_excel(here("london", "raw_data", "rent", "raw_rent.xls"), sheet = "Table 1.2", skip =

# Get mean and median rent values for 1 Bedroom
rent_1b = rent_raw %>% filter(`Bedroom Category` == "One Bedroom") %>%
  dplyr::select("name" = Borough,
```

```
                "mean_rent1b" = Mean,
                "median_rent1b" = Median)

# Get mean and median rent values for 2 Bedroom
rent_2b = rent_raw %>% filter(`Bedroom Category` == "Two Bedrooms") %>%
  dplyr::select("name" = Borough,
                "mean_rent2b" = Mean,
                "median_rent2b" = Median)
```

9. Green Cover (Raster)

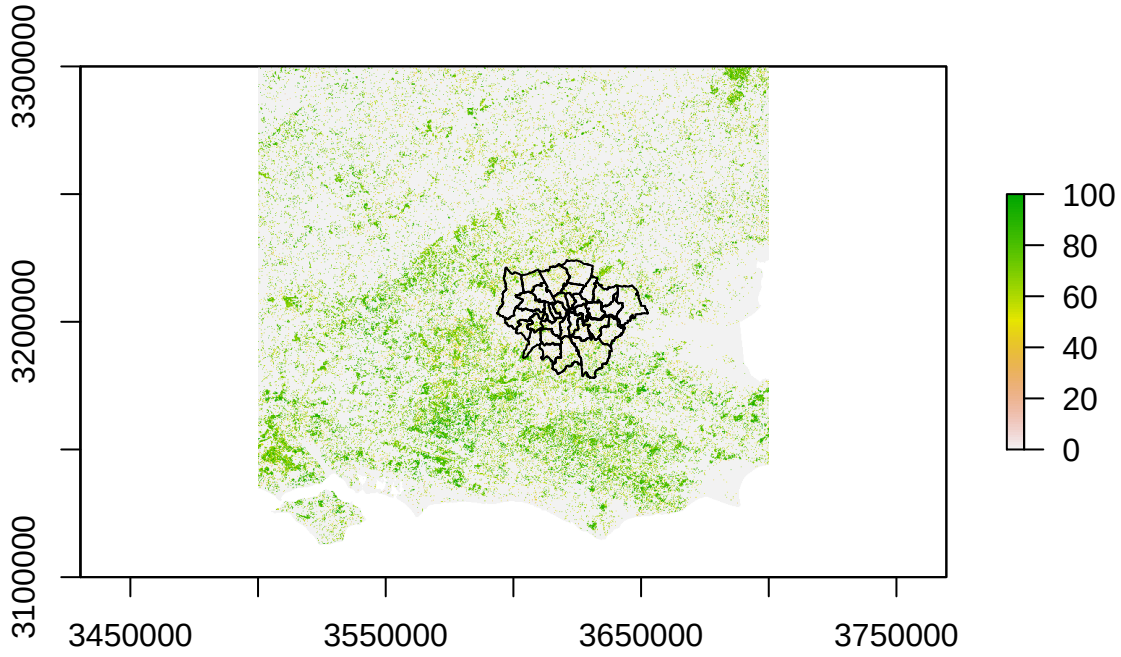
The aim was a value of mean percentage of green cover for each of the boroughs. Thus, we extracted raster data from the Corpernicus Land Monitoring Service, which contains percentages of green cover on a high resolution. The data is split into tiles. Thus, the relevant tiles which covered London were found, loaded and merged into one raster file. Then, we transform the vector data of the boroughs to the crs of the raster data, we mask and crop the raster data using the transformed vector data, and lastly extract mean values of the raster data (green cover percentage) for each of the boroughs. Note that here some of the code is commented out, to be able to knit the document. If you are running the code, a warning message might occur about the datum and projection. This was not deemed to have an affect on results, and thus ignored. You can read more about the warning here: https://stackoverflow.com/questions/63374488/since-update-of-sp-package-i-get-a-warning-by-calling-a-spcrs-definition.

- Data Source: https://land.copernicus.eu/pan-european/high-resolution-layers/forests/tree-cover-density/status-maps/tree-cover-density-2018

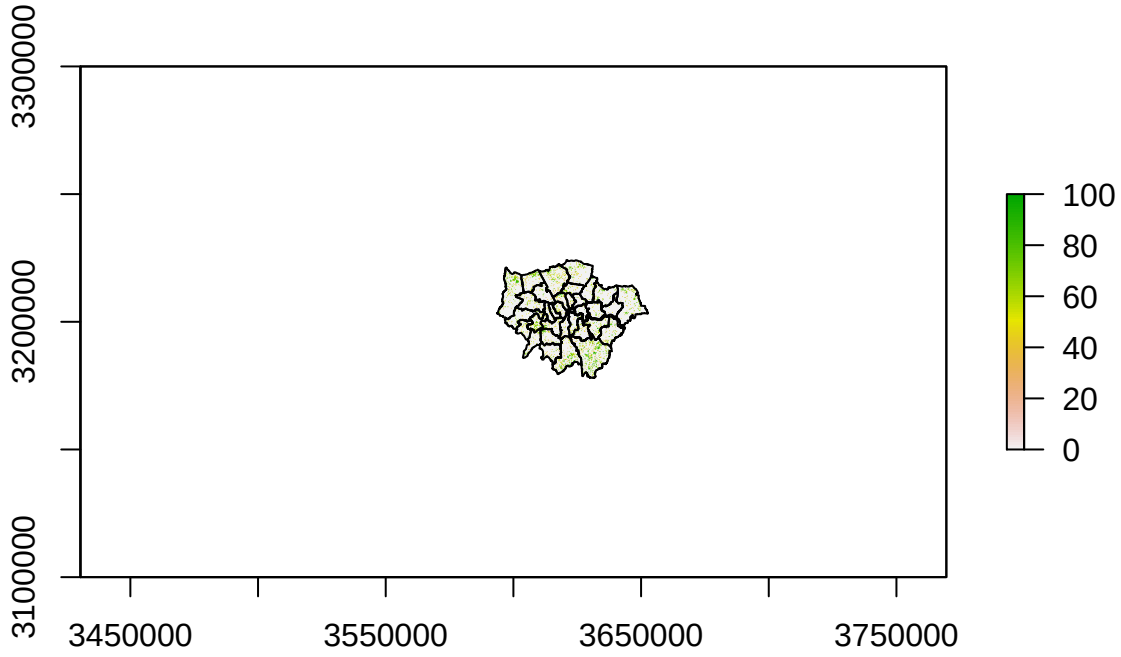
```
# Loading all rasters of the london area
#raster1 = raster(here("london", "raw_data", "treecover", "TCD_2018_010m_E36N32_03035_v020.tif"))
# raster2 = raster(here("london", "raw_data", "treecover", "TCD_2018_010m_E36N31_03035_v020.tif"))
# raster3 = raster(here("london", "raw_data", "treecover", "TCD_2018_010m_E35N32_03035_v020.tif"))
# raster4 = raster(here("london", "raw_data", "treecover", "TCD_2018_010m_E35N31_03035_v020.tif"))

# Merge all raster files and save in new file
# raster::merge(raster1, raster2, raster3, raster4, filename = "london/raw_data/treecover/merged_treeco

# Loading merged raster
treecover_raw = raster(here("london", "raw_data", "treecover", "merged_treecover.tif"))
crs(treecover_raw) #GRS80, LAEA = projected
```

```
## CRS arguments:
##  +proj=laea +lat_0=52 +lon_0=10 +x_0=4321000 +y_0=3210000 +ellps=GRS80
## +units=m +no_defs
```

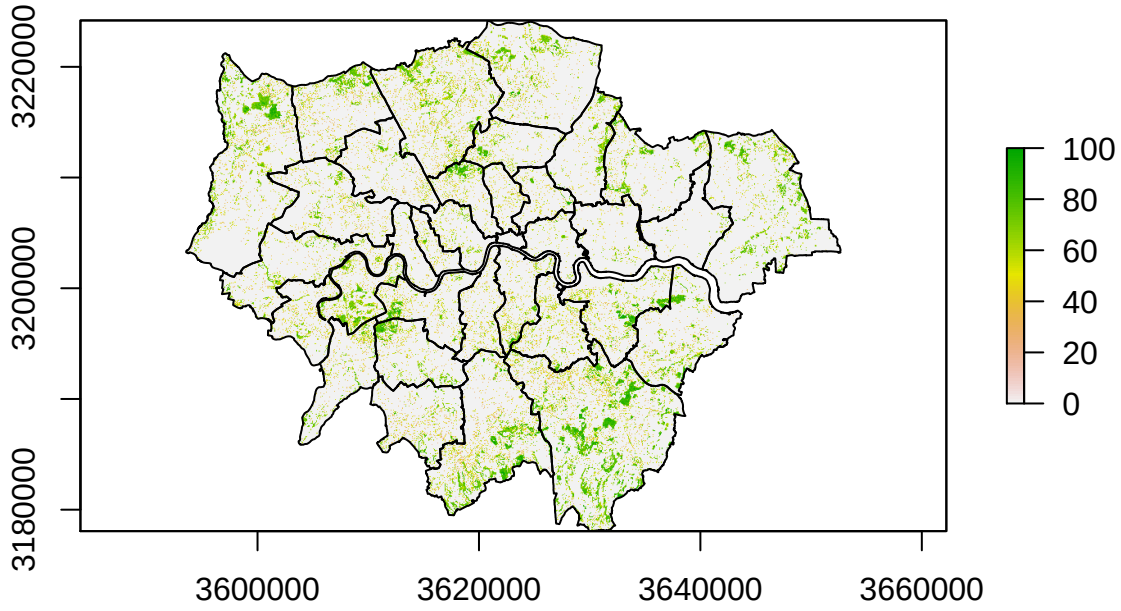
```
# Aiming to get percentage of green area for each borough, so the CRS of the raster data and boroughs
# need to be aligned. We choose the CRS of the tree-cover, as this is more precise of the london-area.
# Get the CRS of the raster tree cover data
crs_3035 = crs(treecover_raw, asText=TRUE)
# Transform the London boroughs into the tree cover CRS
london_3035 = st_transform(london_geo, crs = crs_3035)

# Quick plot to see what it looks like
plot(treecover_raw, col=rev(terrain.colors(100)))
plot(london_3035$geometry, add=TRUE)
```




```
# Masking raster data to only keep the London area
#tree_mask = raster::mask(treecover_raw, london_3035)
# Save masked raster file
#raster::writeRaster(tree_mask, "london/raw_data/treecover/masked_treecover.tif")
# Read raster for plotting and knitting
tree_mask = raster(here("london", "raw_data", "treecover", "masked_treecover.tif"))
# Plot
plot(tree_mask, col=rev(terrain.colors(100)))
plot(london_3035$geometry, add=TRUE)
```




```
# Cropping to only keep the London boroughs
#tree_cropped = raster::crop(tree_mask, london_3035)
# Save as raster file
#raster::writeRaster(tree_cropped, "london/raw_data/treecover/cropped_treecover.tif")
# Load raster file for plotting and knitting
tree_cropped = raster(here("london", "raw_data", "treecover", "cropped_treecover.tif"))
# Plot
plot(tree_cropped, col=rev(terrain.colors(100)));
plot(london_3035$geometry, add=TRUE)
```




```
# Print info of data from which values will be extracted
tree_cropped
```

```
## class      : RasterLayer
## dimensions : 4613, 5912, 27272056  (nrow, ncol, ncell)
## resolution : 10, 10  (x, y)
## extent     : 3593550, 3652670, 3178060, 3224190  (xmin, xmax, ymin, ymax)
## crs        : +proj=laea +lat_0=52 +lon_0=10 +x_0=4321000 +y_0=3210000 +ellps=GRS80 +units=m +no_defs
## source     : /Users/nicoledwenger/Documents/University/6SEMESTER/Spatial Analytics/FINALPROJECT/cdss
## names      : cropped_treecover
## values     : 0, 100  (min, max)
```

```
# Extract mean tree cover values for each of the boroughs, the followin is only commented out for knitt
# boroughs_treecover = raster::extract(x=tree_cropped$merged_treecover, y=london_3035, fun=mean, na.rm=

# Save percentages to join to the data frame later
# treecover = as.data.frame(borough_names) %>% mutate("treecover"=boroughs_treecover)
# colnames(treecover)[1] = "name"

# Save extracted values for knitting
# saveRDS(treecover, "london/preprocessed_data/london_tree_extracted.rds")

# Read data for knitting
treecover = readRDS(here("london", "preprocessed_data", "london_tree_extracted.rds"))
```

10. Imperviousness (Raster)

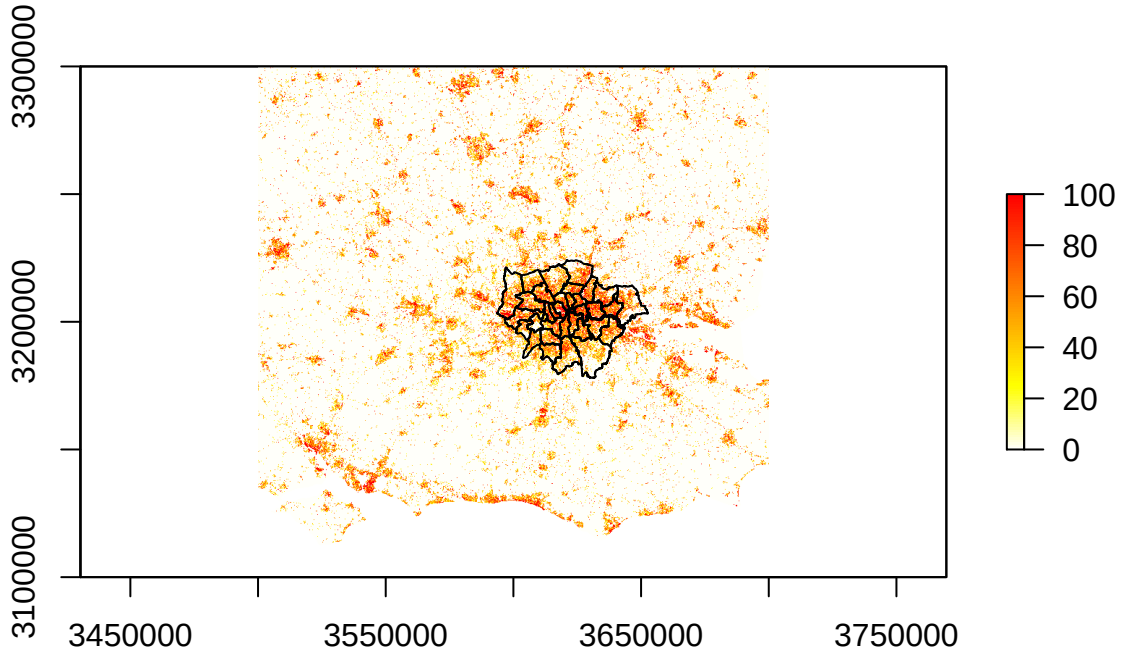
The aim was a value of mean percentage of imperviousness for each of the boroughs. Thus, we extracted raster data from the Corpernicus Land Monitoring Service, which contains percentages of imperviousness on a high resolution. The data is split into tiles. Thus, the relevant tiles which covered London were found, loaded and merged into one raster file. Then, we transform the vector data of the boroughs to the crs of the raster data, we mask and crop the raster data using the transformed vector data, and lastly extract mean values of the raster data (imperviousness percentage) for each of the boroughs. Note that here some of the code is commented out, to be able to knit the document. This was not deemed to have an affect on results, and thus ignored. You can read more about the warning here: https://stackoverflow.com/questions/63374488/since-update-of-sp-package-i-get-a-warning-by-calling-a-spcrs-definition.

- Source: https://land.copernicus.eu/pan-european/high-resolution-layers/imperviousness/status-maps/imperviousness-density-2018

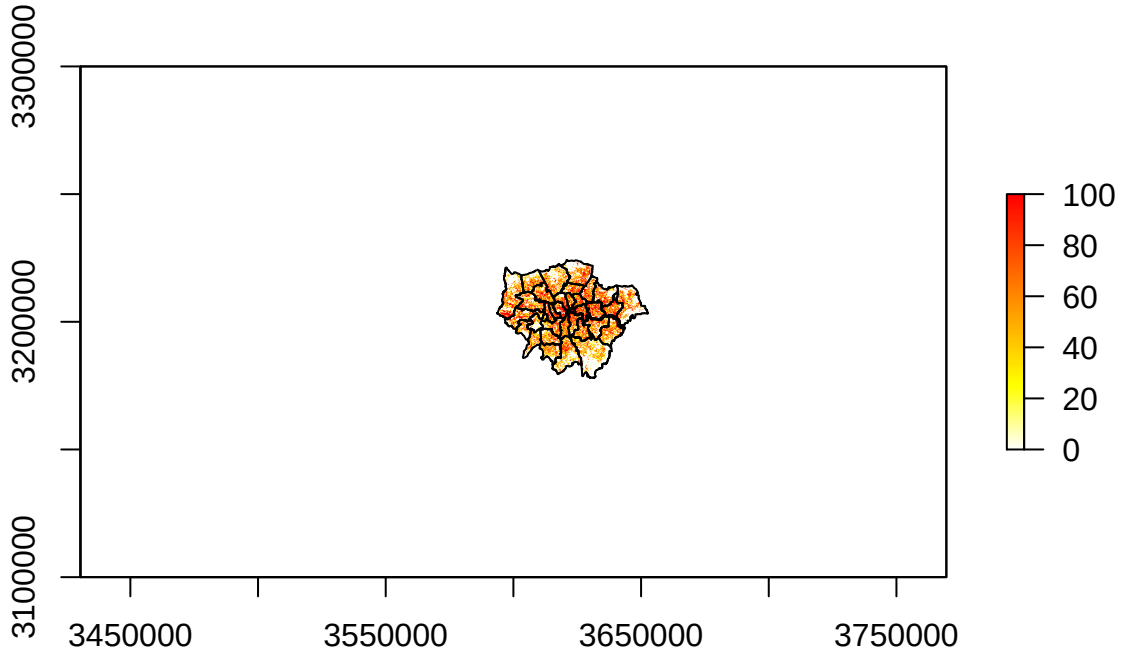
```
# Loading all rasters in london area
# raster1 = raster::raster(here("london", "raw_data", "imperviousness", "IMD_2018_010m_E35N31_03035_v02
# raster2 = raster(here("london", "raw_data", "imperviousness", "IMD_2018_010m_E35N32_03035_v020.tif"))
# raster3 = raster(here("london", "raw_data", "imperviousness", "IMD_2018_010m_E36N31_03035_v020.tif"))
# raster4 = raster(here("london", "raw_data", "imperviousness", "IMD_2018_010m_E36N32_03035_v020.tif"))

# Merge all raster files and save in new file
# raster::merge(raster1, raster2, raster3, raster4, filename = "london/raw_data/imperviousness/merged_i

# Loading merged raster
london_imperv = raster(here("london", "raw_data", "imperviousness", "merged_imperv.tif"))
crs(london_imperv) #GRS80, LAEA
```

```
## CRS arguments:
##  +proj=laea +lat_0=52 +lon_0=10 +x_0=4321000 +y_0=3210000 +ellps=GRS80
## +units=m +no_defs
```

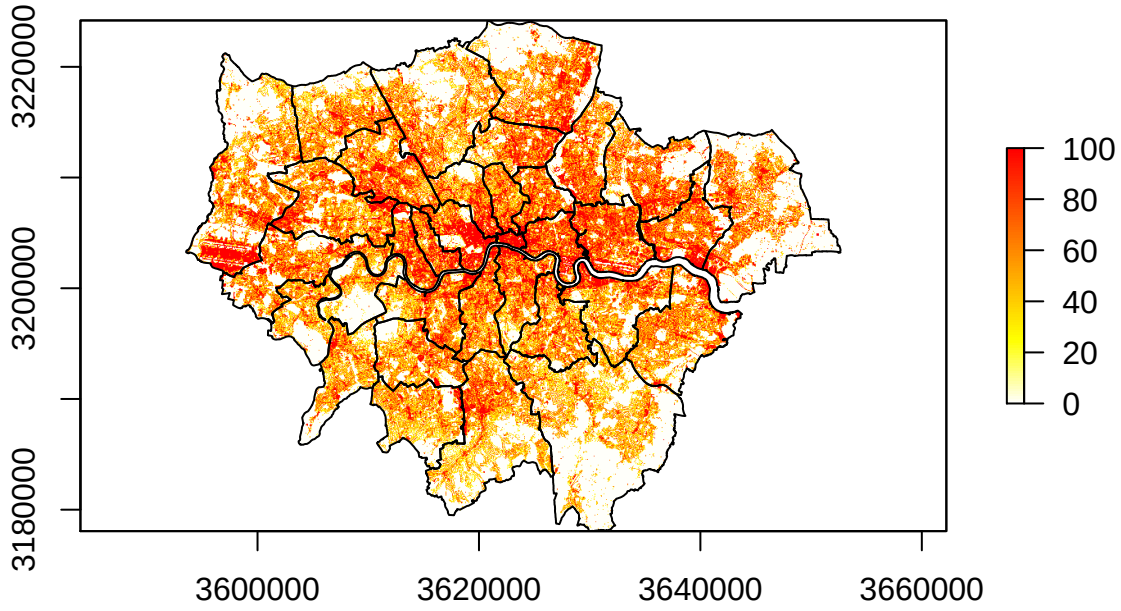
```
# Transform london geometry to the raster crs
crs_3035 = crs(london_imperv, asText=TRUE)
london_3035 = st_transform(london_geo, crs = crs_3035)

# Quick plot to see how it looks
plot(london_imperv, col=rev(heat.colors(100)));
plot(london_3035$geometry, add=TRUE)
```




```
# Mask to only London area (commented out for knitting)
# imperv_mask = mask(london_imperv, london_3035)
# Save as raster
# writeRaster(imperv_mask, "london/raw_data/imperviousness/masked_imperv.tif")
# Read raster for plotting and knitting
imperv_mask = raster(here("london", "raw_data", "imperviousness", "masked_imperv.tif"))
# Plot
plot(imperv_mask, col=rev(heat.colors(100)));
plot(london_3035$geometry, add=TRUE)
```




```
# Crop to only have London area (commented out for knitting)
# imperv_cropped = crop(imperv_mask, london_3035)
# Save as raster
# writeRaster(imperv_cropped, "london/raw_data/imperviousness/cropped_imperv.tif")
# Read raster for plotting and knitting
imperv_cropped = raster(here("london", "raw_data", "imperviousness", "cropped_imperv.tif"))
# Plot
plot(imperv_cropped, col=rev(heat.colors(100)));
plot(london_3035$geometry, add=TRUE)
```




```
# Print info of data from which values will be extracted
imperv_cropped
```

```
## class      : RasterLayer
## dimensions : 4613, 5912, 27272056  (nrow, ncol, ncell)
## resolution : 10, 10  (x, y)
## extent     : 3593550, 3652670, 3178060, 3224190  (xmin, xmax, ymin, ymax)
## crs        : +proj=laea +lat_0=52 +lon_0=10 +x_0=4321000 +y_0=3210000 +ellps=GRS80 +units=m +no_defs
## source     : /Users/nicoledwenger/Documents/University/6SEMESTER/Spatial Analytics/FINALPROJECT/cdss
## names      : cropped_imperv
## values     : 0, 100  (min, max)
```

```
# Extract values for each of the boroughs, mean value, to add them to the final data later (commented o
# boroughs_imperv = raster::extract(imperv_cropped, london_3035, fun = mean)

# Save to join
# imperviousness = as.data.frame(borough_names) %>% mutate("imperviousness" = boroughs_imperv)
# colnames(imperviousness)[1] = "name"

# Save rds
# saveRDS(imperviousness, "london/preprocessed_data/london_imperv_extracted.rds")

# Read rds for knitting
imperviousness = readRDS(here("london", "preprocessed_data", "london_imperv_extracted.rds"))
```

FINAL: Joining all data

Here, we take all of the data which we have processed above, and join it together to a large dataframe, which will be used as input for the shiny app.

```
# Joining the all data together
london = left_join(london_geo, age, by = "name") %>%
         mutate(population_dens_ha = (total_population/area_ha)) %>%
         mutate(population_dens_km2 = (total_population/area_km2)) %>%
         left_join(., ethnicity, by = "name") %>%
         left_join(., religion, by = "name") %>%
         left_join(., culture, by = "name") %>%
         left_join(., travel_time, by = "name") %>%
         left_join(., crime, by = "name") %>%
         mutate(crime_rate = (crime_total/total_population)*1000) %>%
         left_join(., rent_1b, by = "name") %>%
         left_join(., rent_2b, by = "name") %>%
         left_join(., treecover, by = "name") %>%
         left_join(., imperviousness, by = "name")

# Printing colnames
colnames(london)
```

```
##  [1] "name"                       "area_ha"
##  [3] "area_km2"                   "total_population"
##  [5] "perc_age_0_9"               "perc_age_10_19"
##  [7] "perc_age_20_29"             "perc_age_30_39"
##  [9] "perc_age_40_49"             "perc_age_50_59"
## [11] "perc_age_60_69"             "perc_age_70_79"
## [13] "perc_age_80_89"             "perc_age_90+"
## [15] "age_mean"                   "population_dens_ha"
## [17] "population_dens_km2"        "ethnicity_perc_arab"
## [19] "ethnicity_perc_asian"       "ethnicity_perc_black"
## [21] "ethnicity_perc_mixed"       "ethnicity_perc_white"
## [23] "ethnicity_perc_other"       "n_buddhist"
## [25] "n_hindu"                    "n_jewish"
## [27] "n_muslim"                   "n_sikh"
## [29] "n_protestant"               "n_catholic"
## [31] "n_museum"                   "n_theatre"
## [33] "n_nightlife"                "time_train_waterloo"
## [35] "time_train_paddington"      "time_train_kingscross"
## [37] "time_train_liverpoolstreet" "time_train_victoria"
## [39] "time_train_londonbridge"    "time_air_heathrow"
## [41] "time_air_gatwick"           "time_air_stansted"
## [43] "time_air_londonca"          "time_air_southend"
## [45] "time_air_luton"             "crime_total"
## [47] "crime_rate"                 "mean_rent1b"
## [49] "median_rent1b"              "mean_rent2b"
## [51] "median_rent2b"              "treecover"
## [53] "imperviousness"             "center_coords"
## [55] "geometry"
```

```
# Save rds
#saveRDS(london, "london/preprocessed_data/london.rds")

# Save CSV
#write.csv(london, "london/preprocessed_data/london.csv")
```