CHIC599 Mini Project: Accessibility stages document

John Archer

Student ID: 35813062

Supervisors: Dr Michelle Stanton and Mr Joshua Longbottom

This document details each of the stages involved in generating and summarising the accessibility surface.

Note: Tables with blue header detail code that I contributed. Tables with red header detail code that had already been written prior to my joining the project.

1. Setup

As part of the MRes mini-project, the .Rmd code was restructured to contain an initial 'setup' section, allowing users to input specific parameters relating to:

- Downloading, installing and loading all packages needed to generate and summarise the accessibility surface from both CRAN and GitHub servers
- Defining the country of interest (coi), used to obtain World Health Organisation (WHO) health facility data and WorldPop population data
- Defining the area of interest (through inputting specific co-ordinate data or shapefiles)
- Reading in Landsat-8 satellite data (obtained using Google Earth Engine (GEE)) and defining start and end dates to filter Landsat-8 satellite data by
- Walking travel speeds (non-vehicle, km/ hour) expected when traversing non-road pixels (dependant on NDVI value and customisable according to expected changes in travel speed, e.g., during the wet season).
- Road travel speeds (km/ hour by motor vehicle) for major and minor road types expected when traversing road pixels (road data obtained using OpenStreetMaps and customisable according to expected changes in travel speed, e.g., during the wet season).
- Defining parameters used to download population data, obtained using the `wpgpDownloadR` package
- Population data obtained using the `wpgpDownloadR` package, an interface for downloading raster population data from the WorldPop FTP.
- Connecting to and initialising `rgee` package

1a. Setup: Downloading, installing and loading all required packages (from both CRAN and GitHub servers)

Task	Package(s)	Function(s)	R/GEE

Download, install and load all required packages available from **CRAN**.	<u>Packages to install:</u> `sf`, `mapview`, `googledrive`, `osmdata`, `ggplot2`, `raster`, `gdistance`, `fasterize`, `remotes`, `rgdal`, `stars`, `geojsonio` `devtools` `rgee` `tidyr` `knitr`	`# Create vector list of packages needed` `list.of.packages <-` `c(list_packages)` `# Identify which packages aren't currently installed on client computers and store in object` `new.packages <-` `list.of.packages[!(list.of.packages %in% installed.packages()[,"Package"])]` `# Install any missing packages` `if(length(new.packages))` `install.packages(new.packages)` `# Load all required packages` `lapply(list.of.packages, library,` `character.only = TRUE)`	Executed within R, Does **not** require GEE
Download, install and load all required packages **not** available from CRAN. These are downloaded from GitHub and installed using the `devtools` package.	<u>Packages to install:</u> `wpgpDownloadR`, `wpgpCovariates`, `afrimapr/afrihealthsites`	`# Install devtools package, if needed` `if(!("devtools" %in% installed.packages())){` `install.packages("devtools")` `}`	Executed within R, Does **not** require GEE

		# Install wpgpDownloadR package, if needed if(!("wpgpDownloadR" %in% installed.packages())){ devtools::install_github("wpgp/wpgpDownloadR") } # Install wpgpCovariates package, if needed if(!("wpgpCovariates" %in% installed.packages())){ devtools::install_github("wpgp/wpgpCovariates") } # Install afrimapr/afrihealthsites package, if needed if(!("afrimapr/afrihealthsites" %in% installed.packages())){ devtools::install_github("afrimapr/afrihealthsites") } # Load packages library(devtools)	

		`library(wpgpDownloadR)` `library(wpgpCovariates)` `library(afrihealthsites)`	

All required packages should now be loaded.

1b. Setup: Define country of interest (coi)

Task	Package(s)	Function(s)	R/GEE
Define the country of interest (coi) using British English spelling and capitalised first letter.		`# Create object that defines country of interest` `coi <- "Malawi"`	Executed within R, Does **not** require GEE

1c. Setup: Define area of interest (aoi)

Task	Package(s)	Function(s)	R/GEE
Define the area of interest (aoi) by detailing coordinate limits of a bounding box (latitude/longitude maximum and minimum values using WGS84 projection) to be used as area of interest	`rgdal` (if reading in shapefile)	`# Detail limits of bounding box` `bbxmin <- co-ordinate_1_xmin (WGS84)` `bbxmax <- co-ordinate_2_xmax (WGS84)` `bbymin <- co-ordinate_3_ymin (WGS84)` `bbymax <- co-ordinate_4_ymax (WGS84)`	Executed within R, Does **not** require GEE

OR Read in shapefile to define area of interest		**OR** `# Read in shapefile` `aoi <- readOGR(` `dsn = "user_directory",` `layer = "user_shapefile")`	

1d. Setup: Read in Landsat-8 satellite data and define start and end dates to filter Landsat-8 satellite data by

Task	Package(s)	Function(s)	R/GEE
Read in Landsat-8 satellite data, obtained using Google Earth Engine (GEE)		`# Read in Landsat-8 satellite data` `ls8_data <- "user_landsat_8_data"`	Executed within R, Does **not** require GEE
Define start and end dates to filter Landsat-8 data by		`# Read in Landsat-8 satellite data` `start_date <- "2018-06-01"` `end_date <- "2018-09-30"`	Executed within R, Does **not** require GEE

1e. Setup: Define off-road (on-foot) travel speeds (non-vehicle, km/ hour and dependant on NDVI values)

Task	Package(s)	Function(s)	R/GEE
Define walking travel speeds (non-vehicle, km/ hour) expected when traversing non-road pixels (dependant on NDVI value and customisable		`# NDVI value = < 0.35 (impassable)` `walk_speed_1 <- 0.1` `# NDVI value = 0.35 – 0.6` `walk_speed_2 <- 3.5`	Executed within R, Does **not** require GEE

according to expected changes in travel speed, e.g., during the wet season).		`# NDVI value = 0.6 - 0.7` `walk_speed_3 <- 2.48` `# NDVI value = > 0.7` `walk_speed_4 <- 1.49`	

1f. Setup: Define road travel speeds (km/ hour by motorcycle) for major and minor road types

Task	Package(s)	Function(s)	R/GEE
Define road travel speeds (km/ hour by motor vehicle) for major and minor road types expected when traversing road pixels (road data obtained using OpenStreetMaps and customisable according to expected changes in travel speed, e.g., during the wet season).		`# Major road speed, on which national speed limits can typically be reached (e.g., motorway and trunk roads)` `major_road_speed <- 80` `# Minor road speed, on which slower speeds would be expected (e.g., urban roads, dirt roads)` `minor_road_speed <- 20`	Executed within R, Does **not** require GEE

1g: Setup: Population data parameters

Task	Package(s)	Function(s)	R/GEE
Create the `population_data` function, constructed to return a dataframe of available population	`wpgpDownloadR`, `wpgpCovariates`	`population_data <-` `function(coi){`	Executed within R, Does **not** require GEE

covariates downloaded from the WorldPop FTP (for country of interest) based on `coi` (defined during setup stage 2)		`# Obtain ISO3 country codes` `ISO3_df <- wpgpListCountries()` `# Identify ISO3 country code for country of interest (coi) and store as object` `ISO3 <- ISO3_df[ISO3_df$Country == Coi, "ISO3"]` `# Download dataset of available covariates for country of interest and store as object` `covariates_df <- wpgpListCountryDatasets(ISO3=ISO3)` `# Return dataframe` `return(covariates_df)` `}`	
Select and set covariate of interest by running `population_data` function and viewing dataframe of available covariates downloaded from WorldPop FTP		`View(population_data(coi))` `# Using our Malawi example, the "ppp_2020" dataset provides estimated total number of people per grid-cell for year 2020`	Executed within R, Does **not** require GEE

Store chosen covariate as object		`covariate <- "ppp_2020"`	Executed within R, Does **not** require GEE

1h. Setup: Connecting to and initialising `rgee` package

More info on this process can be found here.

Task	Package(s)	Function(s)	R/GEE
Connect to `rgee` package. NOTE: `ee_install` function only needs to be run once.	rgee googledrive	`# Connect to Google Earth Engine (GEE) using rgee` `ee_install(rgee)`	Executed within R, **Performed using GEE**
Initialize `rgee` package to check whether everything has been set up correctly in order to begin using Google Earth Engine (GEE) via R	rgee googledrive	`# Initialize rgee package` `ee_initialize(drive = TRUE)`	Executed within R, **Performed using GEE**

All setup stages are now complete.

2. Create area of interest (aoi) polygon

Create an area of interest polygon using coordinates defied during setup stage 1c (WGS84 projection).

Task	Package(s)	Function(s)	R/GEE

Create polygon using xy coordinate combinations for each corner of the area of interest (aoi)	`rgee`	`aoi <- ee$Geometry$Polygon( cords = list (c(bbxmin, bbymax), c(bbxmax, bbymax), c(bbxmax, bbymin), c(bbxmin, bbymin)))`	Executed within R, **Performed using GEE**

3. Read in and filter Landsat-8 (ls8) Tier 1 satellite data and filter by area of interest and collection date

Landsat-8 satellite obtained using Google Earth Engine (GEE)

Task	Package(s)	Function(s)	R/GEE
Read in Landsat 8 (ls8) Tier 1 dataset	`rgee`	`ls8 <- ee$ImageCollection(user_landsat_data)`	Executed within R, **Performed using GEE**
Filter ls8 data by area of interest (`aoi`)	`rgee`	`spatialFiltered <- ls8$filterBounds(aoi)`	Executed within R, **Performed using GEE**
Filter ls8 data by collection date	`rgee`	`temporalFiltered <- spatialFiltered$filterDate( start_date, end_date)`	Executed within R, **Performed using GEE**

4. Create and apply a cloud mask to filtered LS8 data

Excludes any pixel deemed to be 'cloud' data (clouds and cloud shadow) from any further analysis.

Task	Package(s)	Function(s)	R/GEE

Create `ndvilowcloud` function, constructed to generate a cloud score, create mask of cloudy pixels, compute NDVI (using in-built functions) and return a masked image with an NDVI band	`rgee`	`ndvilowcloud <-` `function(image){` `# Generate a cloud score in [0, 100]` `cloud <- ee$Algorithims$landset$simpleCloudScore(image)$select('cloud')` `# Create a mask of cloudy pixels from an arbitrary threshold (20%)` `mask <- cloud$lte(20)` `# Compute NDVI using inbuilt functions` `nvdi <- image$normalizedDifference(c('B5', 'B4'))$rename('NDVI')` `# Return the masked image with an NDVI band` `image$addBands(ndvi)$updateMask(mask)` `}`	Executed within R, **Performed using GEE**
Apply cloud mask	`rgee`	`cloudlessNDVI = temporalFiltered$map(ndvilowcloud)`	Executed within R, **Performed using GEE**

5. Calculate median normalised difference vegetation index (NDVI) per pixel and clip to area of interest (and view output)

Calculates median NDVI values for each image pixel across all satellite imagery generated from within specified time period.

Task	Package(s)	Function(s)	R/GEE
Calculate median NDVI per pixel	`rgee`	`medianimage <-` `cloudlessNDVI$median()$select('NDVI')`	Executed within R, **Performed using GEE**
Clip to aoi	`rgee`	`medNDVIaoi <-` `medianimage$clip(aoi)`	Executed within R, **Performed using GEE**
View output	`rgee`	`Map$centerObject(aoi)` `Map$addLayer(` `eeObject = medNDVIaoi,` `viaParam = list(min = -1,` `max = 1,` `palette = c('blue',` `'white',` `'green')),` `name = "Median NDVI")`	Executed within R, **Performed using GEE**

6. Convert image to raster and download it using Google Drive (drive) or Google Cloud Storage (GCS)

These data are saved as an image within google earth engine (GEE).

Convert data to raster and download using drive or GCS.

Raster is stored as .tif file in a temporary local folder, which can then be written to our data folder.

More information on this process can be found here: https://r-spatial.github.io/rgee/reference/ee_as_raster.html

Task	Package(s)	Function(s)	R/GEE

Convert data to raster (within GEE) and download/store in temporary folder	`rgee,` `googledrive,`	`med_ndvi <-` `  ee_as_raster(` `    image = medNDVIaoi,` `    region = aoi,` `    scale = 30` `    via = 'drive')`	Executed within R, **Performed using GEE**
Write raster (.tif) to local folder	`raster`	`writeRaster(` `  med_ndvi,` `  "local_filepath",` `  Format = 'GTiff",` `  Overwrite = TRUE)`	Executed within R, **Performed using GEE**

7. Download OpenStreetMap (OSM) road network data for our area of interest (aoi) within R.

To detail travel speeds within the area of interest, open source road network data, publicly compiled and hosted by OpenStreetMaps (OSM), is used.

Hosted here: www.openstreetmap.org

OSM road data from the area of interest (aoi) can be directly downloaded within R.

Task	Package(s)	Function(s)	R/GEE
Define bounding box		`aoi_bbox <-` `  c(bbxmin,` `    bbymin,` `    bbxmax,` `    bbymax)`	Executed within R, Does **not** require GEE

Obtain road data	`osmdata`	`q <-` `opq(bbox = aoi_bbox) %>%` `add_osm_feature(key = 'highway') %>%` `osmdata_sf()`	Executed within R, Does **not** require GEE
Plot road data to check	`ggplot` `osmdata`	`ggplot(q$osm_lines) + geom_sf()`	

8. Assign travel speeds

8a: Assign off-road (on-foot) travel speeds

Off-road (on-foot) travel speeds are dependent on NDVI values and specified during setup stage 1e.

Task	Package(s)	Function(s)	R/GEE
Temporarily read in NDVI example data from folder (downloaded from GEE) and save as raster object (This may be replaced with `med_ndvi` if `rgee` continues to be reliable).	`raster`	`ndvipath <- "NDVIexample.tif"` `ndvi <- raster(ndvipath)`	Executed within R, Does **not** require GEE
Reclassify raster so that: **NDVI < 0.35** = `walk_speed_1` (0.1 km p/hour; impassable) **NDVI 0.35 – 0.6** = `walk_speed_2` (3.5 km p/hour)	`raster`	`# Generate ndvi_walk_kph vector object` `ndviwalk_kph <- c(walk_speed_1,` `walk_speed_2,` `walk_speed_3,` `walk_speed_4)`	Executed within R, Does **not** require GEE

<table>
<tr>
<td>NDVI 0.6 – 0.7 = walk_speed_3
(2.48 km p/hour)

NDVI > 0.7 = walk_speed_4
(1.49 km p/hour)</td>
<td></td>
<td><pre># Convert ndvi_walk_kph vector to metres
p/second
ndviwalk_mps <- ndviwalk_kph/3.6

Convert to crossing time in seconds
(assumes travel along hypotenuse and
pixel size is 30 m2)
nvdiwalk_secs <- 42.43 / ndviwalk_mps

Convert km p/hour to metres p/second
using matrix

Create matrix
ndviwalk_vec <-
   c(-1, 0.35, nvdiwalk_secs[1],
     0.35, 0.6, nvdiwalk_secs[2],
     0.6, 0.7, nvdiwalk_secs[3],
     0.7, 1,   nvdiwalk_secs[4])

ndviwalk_mat <-
   matrix(ndviwalk_vec,
          ncol = 3,
          byrow = TRUE)</pre></td>
<td></td>
</tr>
</table>

		`` ndvi_assigned <- ndvi # Reclassify ndvi raster ndvi_assigned <- reclassify(ndvi_assigned, ndviwalk_mat) ```	

8b: Assign on-road (by motor vehicle) travel speeds to OpenStreetMaps (OSM) road data

Road travel speeds are dependent on road type and specified during setup stage 1f.

Task	Package(s)	Function(s)	R/GEE
Create dataframe (to be converted to raster)	`osmdata`	``` # Create road-type vector road_vector <- c("primary", "secondary", "motorway", "trunk") # Set road speeds to those within road_vector to 80 km/h, and all other road types to 20 km/h qosm_linesmotorspeedkph <- ifelse(qosm_lineshighway %in% road_vector, ```	Executed within R, Does **not** require GEE

		`major_road_speed,` `minor_road_speed)` `# Convert to metres p/second` `q$osm_lines$motorspeedmps <-` `q$osm_lines$motorspeedkph / 3.6` `# Assume a 30 m resolution cell` `q$osm_lines$time_secs <-` `42.43 / q$osm_lines$motorspeedmps`	
Convert to raster, matching the NDVI raster resolution and extent	fasterise, sf	`# fasterise function only works with polygons, so a road buffer of ~30 m is added` `roads.poly <-` `st_buffer(q$osm_line, 0.00015)` `# Convert to raster` `osm_road_raster <-` `fasterise(roads_poly,` `ndvi_assigned,` `“time_secs”,` `fun = ‘min’)`	Executed within R, Does **not** require GEE

9. Merge ‘NVDI’ raster (`ndvi_assigned`) and road-data raster (`osm_road_raster`)

This step will merge data for off-road and on-road travel to create one cohesive friction surface.

In cells containing both road data and non-road data, road values will be retained as these will be associated with the lowest cost (i.e., quickest speed). Similarly, in cells where both road types (major and minor) are found, major road speeds will take precedence.

Task	Package(s)	Function(s)	R/GEE
Merge `ndvi_assigned` and `osm_road_raster`. Maintain the minimum value, i.e., the quickest cell crossing time.	`raster`	`friction_surface_motor <- mosaic(osm_road_raster, ndvi_assigned, fun = min, tolerance = 1)`	Executed within R, Does **not** require GEE
Save `friction_surface_motor` raster as .tif file	`raster`	`writeRaster(friction_surface_motor, “local_filepath”, format = “GTiff”, overwrite = TRUE)`	Executed within R, Does **not** require GEE

10. Download and prepare health facility location data

Download, prepare and view health facility location data from a World Health Organisation (WHO) database.

Task	Package(s)	Function(s)	R/GEE
Download country-wide WHO health facility data using country of interest	`afrimapr/afrihealthsites`	`mwi_healthfac_who <- afrihealthsites(coi, datasource = “who”)`	Executed within R, Does **not** require GEE
Convert `mwi_healthfac_who` to SpatialPolygonDataFrame class	`sf`	`mwi_healthfac_who_spdf <- as(mwi_healthfac_who, “Spatial”)`	Executed within R, Does **not** require GEE

Crop mwi_healthfac_who_spdf to extent of friction_surface_motor raster	raster	`mwi_healthfac_who_spdf_cropped <- raster::crop(mwi_healthfac_who_spdf, extent y = friction_surface_motor)`	Executed within R, Does **not** require GEE
View health facility data specific to the area of interest (aoi)		`# Convert to dataframe` `mwi_healthfac_who_data <- as.data.frame( mwi_healthfac_who_spdf_cropped)` `# View` `View(mwi_healthfac_who_data)`	Executed within R, Does **not** require GEE

11. Calculate shortest paths

To carry out cost-distance analyses, a transition matrix that estimates travel times (in seconds) required to transition between all friction surface cells and their 8 adjacent cells (queens case contiguity) is created. Health facility location data is then overlaid onto the transition matrix the cumulative 'least-cost' (shortest time) distance to reach each friction surface cell from all available health facility location points is calculated. These are then plotted for visualisation.

Task	Package(s)	Function(s)	R/GEE
Calculate transition matrix	gdistance	`trans_motor <- transition(friction_surface_motor, transitionFunction = function(x){1/mean(x)}, directions = 8)`	Executed within R, Does **not** require GEE

Calculate cumulative cost	`gdistance`	`leastcost_motor <-` `accost(trans_motor,` `as_Spatial(healthfac))`	Executed within R, Does **not** require GEE
Save `leastcost_motor` raster as .tif file	`raster`	`writeRaster(leastcost_motor,` `“local_filepath”,` `format = “GTiff”,` `overwrite = TRUE)`	Executed within R, Does **not** require GEE

12. Create plot to visualise `leastcost_motor` raster data

Task	Package(s)	Function(s)	R/GEE
Store `leastcost_motor` as dataframe		`lcm_df <-` `as.data.frame(leastocost_motor,` `xy = TRUE)`	Executed within R, Does **not** require GEE
Add ‘mins’ column by dividing ‘Layer’ column (shortest path time in seconds) by 60		`Lcm_df$mins <-` `lcm_df$Layer / 60`	Executed within R, Does **not** require GEE
Create plot to visualise raster data. Time-boundary thresholds from closest-proximity health facility set to: < 30 minutes 30 minutes - 1 hour 1 hour - 3 hours 3 hours - 6 hours	`ggplot`	`ggplot()+` `geom_raster(` `data = lcm_df,` `aes(x = x,` `y = y,` `fill = cut(mins,` `c(0,30,` `60, 120`	Executed within R, Does **not** require GEE

6 hours - 12 hours 12 hours - 24 hours > 24 hours		`180, 360,` `720,` `max(mins)))))+` `scale_fill_brewer(` `palette = "YlGnBu")+` `geom_sf(` `data = q$osm_lines,` `colour = "darkgrey",` `alpha = 0.3)+` `geom_sf(` `data = healthfac,` `size = 2,` `colour = "red")+` `guides(fill= guide_legend(` `title= "Time (mins)"))`	

13. Download population data (.tif) for country of interest and covariate of interest, and create raster

Population density data is downloaded from the WorldPop FTP.

Task	Package(s)	Function(s)	R/GEE
Download dataset (.tif) for country and covariate of interest, using `ISO3` and `covariate`, both defined during setup stage 1g.	`wpgpDownloadR`	`pop_data <-` `wpgpGetCountryDataset(` `ISO3 = ISO3,` `covariate = covariate` `destDIR = ("local_filepath")`	Executed within R, Does **not** require GEE

(downloaded .tif stored locally)			
Create raster from .tif file	`raster`	`pop_data <- raster(pop_data)`	Executed within R, Does **not** require GEE

14. Resample and clip `pop_data` raster to match resolution and extent of `leastcost_motor` raster

Task	Package(s)	Function(s)	R/GEE
Determine `pop_data` resolution		`res(pop_data)` `# 0.0008333333 0.0008333333`	Executed within R, Does **not** require GEE
Determine `leastcost_motor` resolution		`res(leastcost_motor)` `# 0.0002694946 0.0002694946`	Executed within R, Does **not** require GEE
Use `resample` function to: * Resample `pop_data` to match resolution of `leastcost_motor` * Clip `pop_data` to match extent of `leastcost_motor`	`raster`	`pop_data <-` `resample(pop_data,` `leastcost_motor,` `method = "bilinear")`	Executed within R, Does **not** require GEE

Check resolution and extent of `pop_data` match that of `leastcost_motor`		`# Check resolution` `res(pop_data) ==` `res(leastcost_motor)` `# TRUE` `# Check extent` `extent(pop_data) ==` `extent(leastcost_motor)` `# TRUE`	Executed within R, Does **not** require GEE

15. Reclassify `leastcost_motor` raster to specified time-boundary categorical zones

All cell values (estimated cumulative travel times to closest-proximity health facility) are then reclassified according to specified time-boundary categorical zones.

Task	Package(s)	Function(s)	R/GEE
Create raster from `lcm_df`	`raster`	`# rasterFromXYZ function works only with three columns, so remove 'layer' column from lcm_df` `lcm_df$layer <- NULL` `# Create raster` `lcm_raster <- rasterfromXYZ(lcm_df)`	Executed within R, Does **not** require GEE
Create matrix of time-boundary categories of interest (This will be used to resample `lcm_raster`)		`## Categories:` `# < 30 minutes` `# 30 minutes – 1 hour` `# 1 hour – 3 hours`	Executed within R, Does **not** require GEE

		`# 3 hours - 6 hours` `# 6 hours - 12 hours` `# > 12 hours` `rcl_matrix <- c( 0,   30,  1,` `                 30,  60,  2,` `                 60,  180, 3,` `                 180, 360, 4,` `                 360, 720, 5,` `                 720, max(lcm_df$mins), 6)` `# Reorder rcl_matrix` `rcl_matrix <- matrix(rcl_matrix,` `                     ncol = 3,` `                     byrow = TRUE)`	
Reclassify `lcm_raster` using `lcm_matrix` according to time-boundary categorical zones	`raster`	`lcm_pop_data_rcl <-` `   reclassify(lcm_raster,` `              rcl_matrix,` `              include.lowest = TRUE)`	Executed within R, Does **not** require GEE
Assign `lcm_pop_data_rcl` coordinate reference system (CRS) to that of `leastcost_motor` (CRS: WGS84)	`raster,` `sf`	`# Assign CRS` `projection(lcm_pop_data_rcl) <-` `   crs(leastcost_motor)` `# Check CRS` `crs(lcm_pop_data_rcl)` `# WGS84`	Executed within R, Does **not** require GEE

Plot lcm_pop_data_rcl to visualise		`plot(lcm_pop_data_rcl)`	Executed within R, Does **not** require GEE

16. Determine population within each time-boundary zone and summate data

Summating population data within each time-boundary zone allows for number of people within certain travel times from health facilities to be determined and also allows for the percent of the population within certain travel times from health facilities to be calculated.

Task	Package(s)	Function(s)	R/GEE
Determine population (population data within pop_data raster) within time-boundary zones using lcm_pop_data_rcl raster and zonal function.	raster	`lcm_rcl_zone <-` `zonal(pop_data,` `lcm_pop_data_rcl,` `fun = sum)`	Executed within R, Does **not** require GEE
Create dataframe from lcm_rcl_zone		`# Create dataframe` `lcm_rcl_zone_df <-` `as.data.frame(lcm_rcl_zone, xy = TRUE)` `# Rename columns` `names(lcm_rcl_zone_df)[1] <-` `"Zone"` `names(lcm_rcl_zone_df)[2] <-` `"Zone Population"` `# Replace time-boundary zone codes with chosen time-boundary categories`	Executed within R, Does **not** require GEE

```
lcm_rcl_zone_df$Zone <-
   c("< 30 minutes,
     30 minutes - 1 hour,
     1 hour - 3 hours,
     3 hours - 6 hours,
     6 hours - 12 hours,
     > 12 hours)

# Add 'Total Population' column
lcm_rcl_zone_df$"Total Population" <-

c(sum(
 lcm_rcl_zone_df$"Zone Population"[1]),
 lcm_rcl_zone_df$"Zone Population"[1:2]),
 lcm_rcl_zone_df$"Zone Population"[1:3]),
 lcm_rcl_zone_df$"Zone Population"[1:4]),
 lcm_rcl_zone_df$"Zone Population"[1:5]),
 lcm_rcl_zone_df$"Zone Population"[1:6]))

# Add '% of Total Population' column
lcm_rcl_zone_df$"% Population" <-

c(sum(

lcm_rcl_zone_df$"Zone Population"[1]/
lcm_rcl_zone_df$"Zone Population"[1:6])
```

		* 100, lcm_rcl_zone_df$"Zone Population"[1:2]/ lcm_rcl_zone_df$"Zone Population"[1:6]) * 100, lcm_rcl_zone_df$"Zone Population"[1:3]/ lcm_rcl_zone_df$"Zone Population"[1:6]) * 100, lcm_rcl_zone_df$"Zone Population"[1:4]/ lcm_rcl_zone_df$"Zone Population"[1:6]) * 100, lcm_rcl_zone_df$"Zone Population"[1:5]/ lcm_rcl_zone_df$"Zone Population"[1:6]) * 100, lcm_rcl_zone_df$"Zone Population"[1:6]/ lcm_rcl_zone_df$"Zone Population"[1:6]) * 100)	
View lcm_rcl_zone_df dataframe		View(lcm_rcl_zone_df)	
Create new dataframe detailing number and percent (%) of population residing	knitr	# Create 'time_boundary' vector time_boundaries <-	

<table>
<tr>
<td>within pre-defined time-boundaries of the closest proximity health centre</td>
<td></td>
<td>
<pre>
   c("< 30 minutes", "< 1 hour", "< 3
   hours", "< 6 hours", "< 12 hours", "<
   24 hours")

Create dataframe
FS_output <-
   data.frame(time_boundaries,
              lcm_rcl_zone_df$`Total
              Population`,
              lcm_rcl_zone_df$`%
              Population`)

Rename columns
names(FS_output)[1] <-
   "Time boundaries"
names(FS_output)[2] <-
   "Number population"
names(FS_output)[3] <-
   "Percent (%)population"

Check output using Kable formatting
FS_output_kable <-
   kable(FS_output,
         caption = "Number and percent
         (%) of population residing
         within pre-defined time-
</pre>
</td>
<td></td>
</tr>
</table>

		`boundaries from the closest` `proximity health centre")` `# View` `FS_output_kable`	

17. Create bar plot to visualise percent (%) of total population within chosen time-boundary categorical zones

Task	Package(s)	Function(s)	R/GEE
Create new dataframe, containing only time-boundary and population percent (%) data		`# Create dataframe` `bar_plot_df <-` `data.frame(FS_output$`Time` `boundaries`,` `FS_output$`Percent (%)` `population`)` `# Rename columns` `names(bar_plot_df)[1] <- "Time-boundary"` `names(bar_plot_df)[2] <- "% Population"`	
Create bar plot using `bar_plot_df` dataframe	`ggplot2`	`FS_barplot <-` `ggplot(data = bar_plot_df,` `aes(x = bar_plot_df$Zone,` `y = bar_plot_df$`%` `Population`)) +` `geom_bar(aes(fill = "% Population"),`	

		`` width = 0.4, position = position_dodge(width=0.5), stat="identity") + scale_x_discrete(limits = bar_plot_df$Zone) + theme_light() + ylab("Percent (%) of population") + xlab("Time-boundary category") + ggtitle("Percent (%) of population residing within pre-defined time-boundaries from the closest proximity health centre") + theme(plot.title = element_text(size = 9), legend.title = element_blank(), axis.title = element_text(size = 9), axis.text.x = element_text(angle = 90, vjust = 0.5, hjust=1)) ```	