**MRes mini project: Accessibility stages document**

*Create a document that lists each of the stages involved in generating and summarising the accessibility surface.*

**Need to fill in ‘packages’ columns**

**Need to add info as to what is being done**

**Re-number all steps**

**Highlight which I did and which already done**

**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, including those potentially missing from client computers, from both CRAN and GitHub servers (where not available from CRAN)
* 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**. | Packages to install:  sf,  mapview,  googledrive,  osmdata,  ggplot2,  raster,  gdistance,  fasterize,  remotes,  rgdal,  stars,  geojsonio  devtools  rgee | # 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 API,  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. | Packages to install:  wpgpDownloadR,  wpgpCovariates,  afrimapr/afrihealthsites | # Install devtools package, if needed  if(!(“devtools” %in% installed.packages())){  install.packages(“devtools”) }  # 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) | Executed within R API,  Does **not** require GEE |

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 API,  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  **OR**  Read in shapefile to define area of interest |  | # 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)  **OR**  # Read in shapefile  aoi <- readOGR(  dsn = “user\_directory”,  layer = “user\_shapefile”) | Executed within R API,  Does **not** require GEE |

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 API,  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 API,  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 according to expected changes in travel speed, e.g., during the wet season). |  | # NDVI value = < 0.35 (impassable)  walk\_speed\_1 <- 0.1  # NDVI value = 0.35 – 0.6  walk\_speed\_2 <- 3.5  # NDVI value = 0.6 – 0.7  walk\_speed\_3 <- 2.48  # NDVI value = > 0.7  walk\_speed\_4 <- 1.49 | Executed within R API,  Does **not** require GEE |

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 API,  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 covariates downloaded from the WorldPop FTP (for country of interest) based on coi (defined during setup stage 2) |  | population\_data <-  function(coi){  # 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)  } | Executed within R API,  Does **not** require GEE |
| 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 API,  Does **not** require GEE |
| Store chosen covariate as object |  | covariate <- “ppp\_2020” | Executed within R API,  Does **not** require GEE |

1h. Setup: Connecting to and initialising rgee package

|  |  |  |  |
| --- | --- | --- | --- |
| **Task** | **Package(s)** | **Function(s)** | **R/GEE** |
| Connect to and initialize rgee package.  More info on this process can be found here: | rgee  googledrive | # Connect to Google Earth Engine (GEE) using rgee (only need to run once)  ee\_install(rgee)  # Initialize rgee package to check whether everything has been set up correctly in order to begin using Google Earth Engine (GEE) via R (only need to setup once)  ee\_initialize(drive = TRUE) | Executed within R,  **Performed using GEE** |
|  |  |  |  |

**2. Create area of interest (aoi) polygon**

Area of interest defined using 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 API,  **Performed using GEE (rgee)** |

**3. Read in 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$ImageCollection(user\_landsat\_data) | Executed within R API,  **Performed using GEE** |
| Filter ls8 data by area of interest (aoi) |  | spatialFiltered <-  ls8$filterBounds(aoi) | Executed within R API,  **Performed using GEE** |
| Filter ls8 data by collection date |  | temporalFiltered <-  spatialFiltered$filterDate(  start\_date, end\_date) | Executed within R API,  **Performed using GEE** |

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

|  |  |  |  |
| --- | --- | --- | --- |
| **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)$updateMask(mask)  } | Executed within R API,  **Performed using GEE** |
| Apply cloud mask |  | cloudlessNDVI =  temporalFiltered$map(ndvilowcloud) | Executed within R API,  Does **not** require GEE |

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

|  |  |  |  |
| --- | --- | --- | --- |
| **Task** | **Package(s)** | **Function(s)** | **R/GEE** |
| Calculate median NDVI per pixel |  | medianimage <-  cloudlessNDVI$median()$select(‘NDVI’) | Executed within R API,  **Performed using GEE** |
| Clip to aoi |  | medNDVIaoi <-  medianimage$clip(aoi) | Executed within R API,  **Performed using GEE** |
| View output |  | Map$centerObject(aoi)  Map$addLayer(  eeObject = medNDVIaoi,  viaParam = list(min = -1,  max = 1,  palette = c(‘blue’,  ‘white’,  ‘green’)),  name = “Median NDVI”) | Executed within R API,  **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 |  | med\_ndvi <-  ee\_as\_raster(  image = medNDVIaoi,  region = aoi,  scale = 30  via = ‘drive’) | Executed within R API,  **Performed using GEE** |
| Write raster (.tif) to local folder |  | writeRaster(  med\_ndvi,  “local\_filepath”,  Format = ‘GTiff”,  Overwrite = TRUE) | Executed within R API,  **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, an open source road network which is publicly compiled and hosted by OpenStreetMaps (OSM) is used.

Hosted here: [www.openstreetmap.org](http://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 API,  Does **not** require GEE |
| Obtain road data | osmdata | q <-  opq(bbox = aoi\_bbox) %>%  add\_osm\_feature(key = ‘highway’) %>%  osmdata\_sf() | Executed within R API,  Does **not** require GEE |
| Plot road data to check | ggplot  osmdata | ggplot(q$osm\_lines) + geom\_sf() |  |

**Should produce map of road data**

**8. Assign travel speeds**

Need to document how these speeds are determined and how they vary by season and location.

8a: Assign off-road (on-foot) travel speeds, dependent on NDVI values

|  |  |  |  |
| --- | --- | --- | --- |
| **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. |  | ndvipath <- “NDVIexample.tif”  ndvi <- raster(ndvipath) | Executed within R API,  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)  **NDVI 0.6 – 0.7** = walk\_speed\_3  (2.48 km p/hour)  **NDVI > 0.7** = walk\_speed\_4  (1.49 km p/hour) |  | # Generate ndvi\_walk\_kph vector object  ndviwalk\_kph <- c(walk\_speed\_1,  walk\_speed\_2,  walk\_speed\_3,  walk\_speed\_4)  # 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)  # Reclassify ndvi raster  ndvi\_assigned <- ndvi  ndvi\_assigned <-  reclassify(ndvi\_assigned,  ndviwalk\_mat) | Executed within R API,  Does **not** require GEE |

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

For primary and major roads, national speed limits are used as a maximum.

|  |  |  |  |
| --- | --- | --- | --- |
| **Task** | **Package(s)** | **Function(s)** | **R/GEE** |
| Create dataframe (to be converted to raster) |  | # 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  q$osm\_lines$motorspeedkph <-  ifelse(q$osm\_lines$highway %in%  road\_vector,  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 | Executed within R API,  Does **not** require GEE |
| Convert to raster, matching the NDVI raster resolution and extent | fasterise | # 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 API,  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 areas where off-road and on-road cells overlap, road values will be retained as these will be associated with the lowest cost (i.e., quickest speed).

|  |  |  |  |
| --- | --- | --- | --- |
| **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. |  | friction\_surface\_motor <-  mosaic(osm\_road\_raster,  ndvi\_assigned,  fun = min,  tolerance = 1) | Executed within R API,  Does **not** require GEE |
| Save friction\_surface\_motor raster as .tif file |  | writeraster(friction\_surface\_motor,  “local\_filepath”,  format = “GTiff”,  overwrite = TRUE) | Executed within R API,  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.

WHO health facility location data is obtained using the afrihealthsites function, as part of the afrimapr/afrihealthsites package.

|  |  |  |  |
| --- | --- | --- | --- |
| **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 API,  Does **not** require GEE |
| Convert mwi\_healthfac\_who to SpatialPolygonDataFrame class |  | mwi\_healthfac\_who\_spdf <-  as(mwi\_healthfac\_who, “Spatial”) | Executed within R API,  Does **not** require GEE |
| Crop mwi\_healthfac\_who\_spdf to extent of friction\_surface\_motor raster |  | mwi\_healthfac\_who\_spdf\_cropped <-  raster::crop(mwi\_healthfac\_who\_spdf,  extent  y = friction\_surface\_motor) | Executed within R API,  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 API,  Does **not** require GEE |

**11. Calculate shortest paths**

|  |  |  |  |
| --- | --- | --- | --- |
| **Task** | **Package(s)** | **Function(s)** | **R/GEE** |
| Calculate transition matrix |  | trans\_motor <-  transition(friction\_surface\_motor,  transitionFunction =  function(x){1/mean(x)},  directions = 8) | Executed within R API,  Does **not** require GEE |
| Calculate cumulative cost |  | leastcost\_motor <-  accost(trans\_motor,  as\_Spatial(healthfac)) | Executed within R API,  Does **not** require GEE |
| Save leastcost\_motor raster as .tif file |  | writeRaster(leastcost\_motor,  “local\_filepath”,  format = “GTiff”,  overwrite = TRUE) | Executed within R API,  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 API,  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 API,  Does **not** require GEE |
| Create plot to visualise raster data.  Time-boundary thresholds from closest-proximity health facility set to: < 30 minutes  < 1 hour < 3 hours < 6 hours  < 12 hours  < 24 hours  > 24 hours | ggplot | ggplot()+  geom\_raster(  data = lcm\_df,  aes(x = x,  y = y,  fill = cut(mins,  c(0,30,  60, 120  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)”)) | Executed within R API,  Does **not** require GEE |

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

Using the wpgpGetCountryDataset function, as part of the wpgpDownloadR package,

|  |  |  |  |
| --- | --- | --- | --- |
| **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.  (downloaded .tif stored locally) | wpgpDownloadR | pop\_data <-  wpgpGetCountryDataset(  ISO3 = ISO3,  covariate = covariate  destDIR = (“local\_filepath”) | Executed within R API,  Does **not** require GEE |
| Create raster from .tif file |  | pop\_data <- raster(pop\_data) | Executed within R API,  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 API,  Does **not** require GEE |
| Determine leastcost\_motor resolution |  | res(leastcost\_motor)  # 0.0002694946 0.0002694946 | Executed within R API,  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 |  | pop\_data <-  resample(pop\_data,  leastcost\_motor,  method = “bilinear”) | Executed within R API,  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 API,  Does **not** require GEE |

**15. Reclassify leastcost\_motor raster to chosen time-boundary categorical zones**

|  |  |  |  |
| --- | --- | --- | --- |
| **Task** | **Package(s)** | **Function(s)** | **R/GEE** |
| Create raster from lcm\_df |  | # rasterFromXYZ function works only with three columns, so remove ‘layer’ column from lcm\_df  lcm\_df$layer <- NULL  # Create raster  lcm\_raster <- rasterfromXYZ(lcm\_df) | Executed within R API,  Does **not** require GEE |
| Create matrix of time-boundary categories of interest (This will be used to resample lcm\_raster) |  | ## Categories:  # < 30 minutes  # 30 minutes – 1 hour  # 1 hour – 3 hours  # 3 hours – 6 hours  # 6 hours – 12 hours  # > 12 hours  rcl\_matrix <- c( 0, 30, 1,  30, 60, 2,  60, 180, 3,  180, 360, 4,  360, 720, 5,  720, max(lcm\_df$mins), 6)  # Reorder rcl\_matrix  rcl\_matrix <- matrix(rcl\_matrix,  ncol = 3,  byrow = TRUE) | Executed within R API,  Does **not** require GEE |
| Reclassify lcm\_raster using lcm\_matrix according to time-boundary categorical zones |  | lcm\_pop\_data\_rcl <-  reclassify(lcm\_raster,  rcl\_matrix,  include.lowest = TRUE) | Executed within R API,  Does **not** require GEE |
| Assign lcm\_pop\_data\_rcl coordinate reference system (CRS) to that of leastcost\_motor (CRS: WGS84) |  | # Assign CRS  projection(lcm\_pop\_data\_rcl) <-  crs(leastcost\_motor)  # Check CRS  crs(lcm\_pop\_data\_rcl)  # WGS84 | Executed within R API,  Does **not** require GEE |
| Plot lcm\_pop\_data\_rcl to visualise |  | plot(lcm\_pop\_data\_rcl) | Executed within R API,  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. |  | lcm\_rcl\_zone <-  zonal(pop\_data,  lcm\_pop\_data\_rcl,  fun = sum) | Executed within R API,  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  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$”Zone Population”[1:6])  \* 100) | Executed within R API,  Does **not** require GEE |
| View lcm\_rcl\_zone\_df dataframe |  | View(lcm\_rcl\_zone\_df) |  |
| Create new dataframe detailing number and percent (%) of population residing within pre-defined time-boundaries of the closest proximity health centre |  | # Create ‘time\_boundary’ vector  time\_boundaries <-  c("< 30 minutes", "< 1 hour", "< 3  hours", "< 6 hours", "< 12 hours", "<  24 hours")  # Create dataframe  FS\_output <-  data.frame(time\_boundaries,  lcm\_rcl\_zone\_df$`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-  boundaries from the closest  proximity health centre")  # View  FS\_output\_kable |  |

**17. Create bar plot to visualise % 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 |  | 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)) |  |