LTDSMethod

Max Dolton Jones

2023-03-14

Introduction to LTDSMethod

The *LTDSMethod* package was designed for several main tasks:

- Designing line transect distance sampling pilot surveys
- Using pilot survey methodology to inform full LTDS surveys
- Estimate the density of a species within a study area
- Creating a summary of survey effort and results

More specifically, the package was initially intended as a tool to aid monitoring efforts for translocated gopher tortoises throughout Florida. As a result, some of the functions are highly specific to gopher tortoise biology and conservation. However, *LTDSMethod* may be applicable for other studies/projects trying to implement LTDS methods.

The *LTDSMethod* package has a complimentary Standard Operating Procedure (SOP), which can be referenced to help understand background, terminology and access some results behind survey design decisions.

Working example

The *LTDSMethod* package is currently open-access on a GitHub repository. To download and install the package, we first need the devtools package.

```
#Install devtools: Issues? - https://github.com/r-lib/devtools/issues/2131
install.packages("devtools") #Ignore if previously installed
#Load devtools into the session:
library(devtools)
```

We can now use the devtools package to install *LTDSMethod* from GitHub.

```
install_github("maxdoltonjones/LTDSMethod",  build_vignettes = TRUE, force = TRUE)

#The installation will ask if any package updates are needed, we can skip these
#by running the number three (3):
3
```

The installation of the package should also prompt the installation of all package dependencies (other packages which are needed for *LTDSMethod* to work). However, if this is not the case, run the following lines (this may take a few minutes or so):

```
install.packages("rgeos")
install("rgdal")
install("dplyr")
install.packages("readr")
install.packages("htmltools")
install.packages("gt")
install.packages("tinytex")
install.packages("nimble")
install.packages("terra")
install.packages("ggplot2")
install.packages("tinytex")
install.packages("tidyterra")
install.packages("knitr")
install.packages("imager")
install.packages("gstat")
```

Again, if these packages are already installed, the dependencies will be loaded at the same time as *LTDSMethod* is loaded. However, we can also load all of the dependencies separately.

```
library(rgeos)
#> Warning: package 'rgeos' was built under R version 4.1.3
#> Loading required package: sp
#> Warning: package 'sp' was built under R version 4.1.3
#> rgeos version: 0.5-9, (SVN revision 684)
#>  GEOS runtime version: 3.9.1-CAPI-1.14.2
#>  Please note that rgeos will be retired by the end of 2023,
#> plan transition to sf functions using GEOS at your earliest convenience.
#>  GEOS using OverlayNG
#>  Linking to sp version: 1.4-7
#>  Polygon checking: TRUE
library(rgdal)
#> Warning: package 'rgdal' was built under R version 4.1.3
#> Please note that rgdal will be retired by the end of 2023,
#> plan transition to sf/stars/terra functions using GDAL and PROJ
#> at your earliest convenience.
#>
#> rgdal: version: 1.5-32, (SVN revision 1176)
#> Geospatial Data Abstraction Library extensions to R successfully loaded
#> Loaded GDAL runtime: GDAL 3.4.1, released 2021/12/27
#> Path to GDAL shared files: C:/Users/maxdoltonjones/Documents/R/R-4.1.2/library/rgdal/gdal
#> GDAL binary built with GEOS: TRUE
#> Loaded PROJ runtime: Rel. 7.2.1, January 1st, 2021, [PJ_VERSION: 721]
#> Path to PROJ shared files: C:/Users/maxdoltonjones/Documents/R/R-4.1.2/library/rgdal/proj
#> PROJ CDN enabled: FALSE
#> Linking to sp version:1.5-0
#> To mute warnings of possible GDAL/OSR exportToProj4() degradation,
#> use options("rgdal_show_exportToProj4_warnings"="none") before loading sp or rgdal.
library(dplyr)
#> Warning: package 'dplyr' was built under R version 4.1.3
#>
#> Attaching package: 'dplyr'
#> The following objects are masked from 'package:rgeos':
```

```
#>
#>     intersect, setdiff, union
#> The following objects are masked from 'package:stats':
#>
#>     filter, lag
#> The following objects are masked from 'package:base':
#>
#>     intersect, setdiff, setequal, union
library(readr)
#> Warning: package 'readr' was built under R version 4.1.3
library(htmltools)
#> Warning: package 'htmltools' was built under R version 4.1.3
library(nimble)
#> nimble version 0.12.2 is loaded.
#> For more information on NIMBLE and a User Manual,
#> please visit https://R-nimble.org.
#>
#> Attaching package: 'nimble'
#> The following object is masked from 'package:stats':
#>
#>     simulate
library(terra)
#> Warning: package 'terra' was built under R version 4.1.3
#> terra 1.6.17
#>
#> Attaching package: 'terra'
#> The following objects are masked from 'package:nimble':
#>
#>     values, values<-
#> The following object is masked from 'package:rgdal':
#>
#>     project
library(ggplot2)
library(gt)
#> Warning: package 'gt' was built under R version 4.1.3
library(tinytex)
#> Warning: package 'tinytex' was built under R version 4.1.3
library(tidyterra)
#> Warning: package 'tidyterra' was built under R version 4.1.3
#>
#> Attaching package: 'tidyterra'
#> The following object is masked from 'package:stats':
#>
#>     filter
library(knitr)
#> Warning: package 'knitr' was built under R version 4.1.3
#>
#> Attaching package: 'knitr'
#> The following object is masked from 'package:terra':
#>
#>     spin
library(imager)
#> Warning: package 'imager' was built under R version 4.1.3
```

```
#> Loading required package: magrittr
#> Warning: package 'magrittr' was built under R version 4.1.3
#>
#> Attaching package: 'magrittr'
#> The following objects are masked from 'package:terra':
#>
#>     extract, inset
#>
#> Attaching package: 'imager'
#> The following object is masked from 'package:magrittr':
#>
#>     add
#> The following objects are masked from 'package:terra':
#>
#>     depth, width
#> The following object is masked from 'package:nimble':
#>
#>     resize
#> The following object is masked from 'package:sp':
#>
#>     bbox
#> The following objects are masked from 'package:stats':
#>
#>     convolve, spectrum
#> The following object is masked from 'package:graphics':
#>
#>     frame
#> The following object is masked from 'package:base':
#>
#>     save.image
library(gstat)
#> Warning: package 'gstat' was built under R version 4.1.3
```

The *LTDSMethod* package works by saving and loading files from the working directory. It is important to set the working directory, and keep this consistent for the entirety of use of the package and functions.

```
setwd("FILEPATH HERE")
```

Now load in the *LTDSMethod* package and set the session seed to follow the exact workflow of this vignette. If for some reason you need to run a function more than once, please re-run the set.seed function before re-running any functions.

```
library(LTDSMethod)

#Set a seed for reproducible results
set.seed(1235)
```

Before sampling

Let's remind ourselves what is needed to work through the LTDSMethod package. This information can be found in the Standard Operating Procedure (SOP):

1. Shapefile of recipient site (only permitted, habitat area)
2. Shapefile of additional acreage (only permitted, habitat area that is not already included in the original shapefile)
3. The acreage of the recipient site (only permitted, habitat area which is to be surveyed)
4. UTM zone of recipient site location (either 16 or 17, see Figure 3 of SOP)
5. Location of soft-release enclosures at recipient site (either embedded into shapefiles, or on a separate map)

Let's start with number 1. Read in our shapefile showing the recipient site to be surveyed.

```
#We can read in a shapefile with:
site.shp <- readOGR(dsn = "SHAPEFILE FILE PATH HERE")
site.shp
```

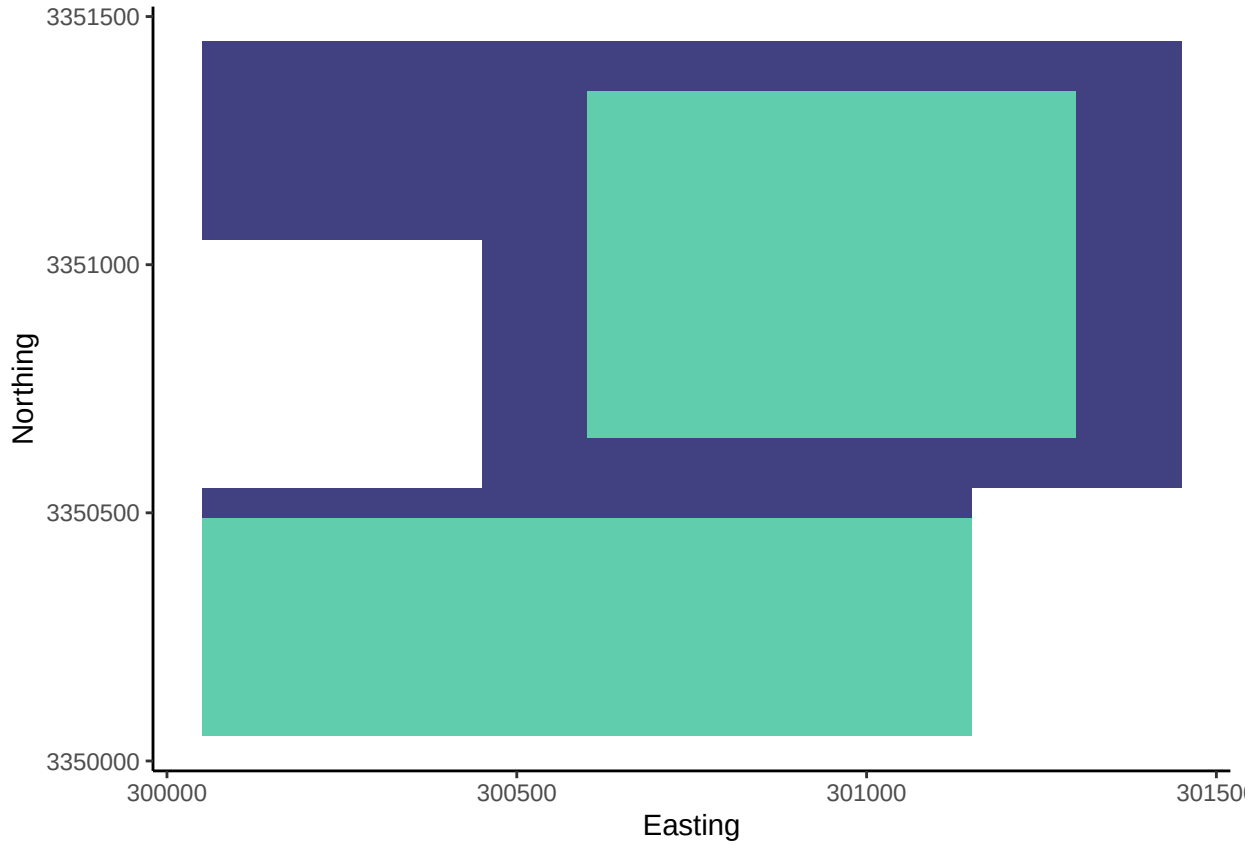
If the data has been collected in FL Albers, or another projection system, we can transform the data to work with the package via the next lines. When setting the new.crs, make sure to allocate the appropriate zone using the SOP as a guide.

```
new.crs <- "+proj=utm +zone=17 +datum=NAD83 +units=m +no_defs +ellps=GRS80 +towgs84=0,0,0"
site.shp <- spTransform(site.shp, new.crs)
site.shp

#Convert the shapefile to a SpatVector, to work with the terra package
site.shp <- vect(site.shp)
#We can visually check the site by:
plot(site.shp)
```

We could move forward with the functions using just the site.shp shapefile just read in. However, here we are going to create our own shapefile and raster. We do this mostly for ease of reproducibility, however, it also allows us to easily visualize pen areas (5 on above list) which is useful for demonstrating the utility of the *LTDSMethod* functions. Specifically, we will create an irregular-shaped site (not square), with 100% available permitted habitat area for tortoises - set at 400 acres, that also has two 120-acre soft-release pens.

```
#400 acre irregular site, 100% habitat, 2x120 acre pens

#Site size is length in meters of one square edge
site.size <- 1500
#Pixel size will be set to 5m per pixel
pixel.size <- 5
#We can create a raster based on those set sizes. Projection is for Florida.
new.crs <- "+proj=utm +zone=17 +datum=NAD83 +units=m +no_defs +ellps=GRS80 +towgs84=0,0,0"
site <- rast(nrows=site.size/pixel.size, ncol=site.size/pixel.size, crs=new.crs)
#Set smallest Easting and Northing for site
xmin <- 300000
ymin <- 3350000
#Set the extent based on coordinates and size
ext(site) <- c(xmin, xmin+site.size, ymin, ymin+site.size)
#For now, set all pixel values to 1
values(site) <- 1
site
#> class       : SpatRaster
```

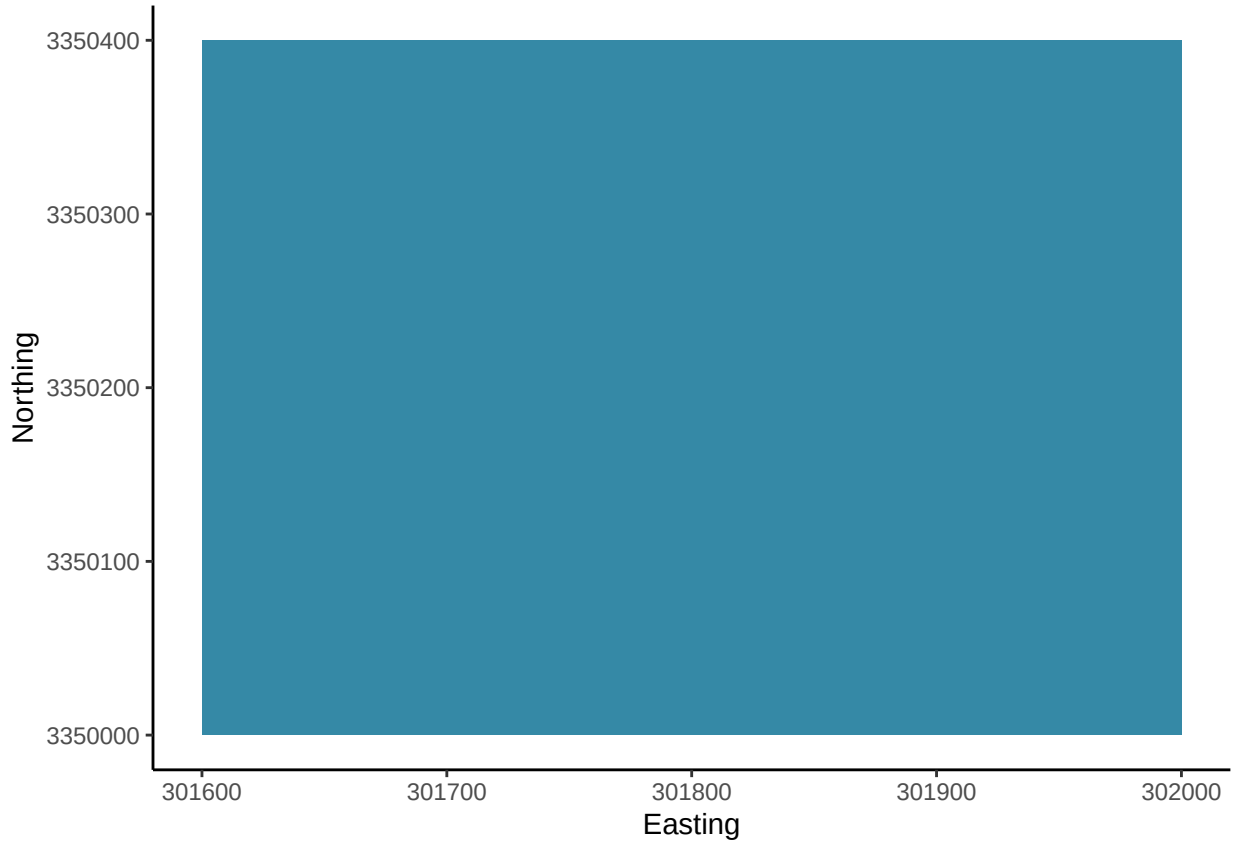
```
#> dimensions  : 300, 300, 1  (nrow, ncol, nlyr)
#> resolution  : 5, 5  (x, y)
#> extent      : 3e+05, 301500, 3350000, 3351500  (xmin, xmax, ymin, ymax)
#> coord. ref. : +proj=utm +zone=17 +ellps=GRS80 +towgs84=0,0,0,0,0,0,0 +units=m +no_defs
#> source      : memory
#> name        : lyr.1
#> min value   :     1
#> max value   :     1

#The below code cuts out sections of the site to give an irregular shape.
area.1 <- crop(site, ext(300050, 301150, 3350050, 3350550))
area.2 <- crop(site, ext(300450, 301450, 3350550, 3351450))
area.3 <- crop(site, ext(300050, 300450, 3351050, 3351450))
site.new <- merge(area.1, area.2, area.3)

#We can use the below code to check the acreage of the site
area <- sum(values(site.new), na.rm = T)
(sqrt(area)*5)^2/4047
#> [1] 397.8255

#We need to add our soft-release pens - just to help with the vignette example
pen.area <- site
values(pen.area) <- 3
pen.1 <- crop(pen.area, ext(300050, 301150, 3350050, 3350490))
pen.2 <- crop(pen.area, ext(300600, 301300, 3350650, 3351350))
pen.new <- merge(pen.1, pen.2)

#Merge site and pen area together
site.rast <- merge(pen.new, site.new)

#Convert and store shapefiles to crop rasters later
site.shp <- as.polygons(site.new)

#Plot raster to visualize site with pen areas
temp<-as.data.frame(site.rast, xy = T)
temp <- temp %>% na.omit()
ggplot(temp) +
  geom_tile(aes(x = x, y = y, fill = lyr.1)) +
  scale_fill_viridis_c(option = "mako", begin = 0.3, end = 0.8) +
  xlab("Easting") +
  ylab("Northing") +
  theme_classic() +
  theme(legend.position = "none")
```



We have now created a recipient site to work with. The second item on our list is any additional acreage to be added to the site. We will only work with the site we have made above moving forward, but we can create additional acreage to test package functionality later on. Additional acreage to the recipient site would usually be provided via shapefile, which we could load in as we did above. We are producing our own here for reproducibility purposes.

```
#Create another site, to demosntrate merging shapefiles
site.size <- 400  #Site size is length in meters of one square edge
pixel.size <- 5
new.crs <- "+proj=utm +zone=17 +datum=NAD83 +units=m +no_defs +ellps=GRS80 +towgs84=0,0,0"
site.2 <- rast(nrows=site.size/pixel.size, ncol=site.size/pixel.size, crs=new.crs)
xmin <- 301600
ymin <- 3350000
ext(site.2) <- c(xmin, xmin+site.size, ymin, ymin+site.size)
values(site.2) <- 1
site.2
#> class       : SpatRaster
#> dimensions  : 80, 80, 1  (nrow, ncol, nlyr)
#> resolution  : 5, 5  (x, y)
#> extent      : 301600, 302000, 3350000, 3350400  (xmin, xmax, ymin, ymax)
#> coord. ref. : +proj=utm +zone=17 +ellps=GRS80 +towgs84=0,0,0,0,0,0,0 +units=m +no_defs
#> source      : memory
#> name        : lyr.1
#> min value   :     1
#> max value   :     1
#Convert and store shapefiles to crop rasters later
```

```
site.shp.2 <- as.polygons(site.2)

temp.2<-as.data.frame(site.2, xy = T)
temp.2 <- temp.2 %>% na.omit()
ggplot(temp.2) +
  geom_tile(aes(x = x, y = y, fill = lyr.1)) +
  scale_fill_viridis_c(option = "mako", begin = 0.3, end = 0.8) +
  xlab("Easting") +
  ylab("Northing") +
  theme_classic() +
  theme(legend.position = "none")
```



The last three requirements can be conveniently parsed from the first site we created. We have already checked the site acreage, but we do this again below.

```
#Calculate site acreage
area <- sum(values(site.new), na.rm = T)
(sqrt(area)*5)^2/4047
#> [1] 397.8255
```

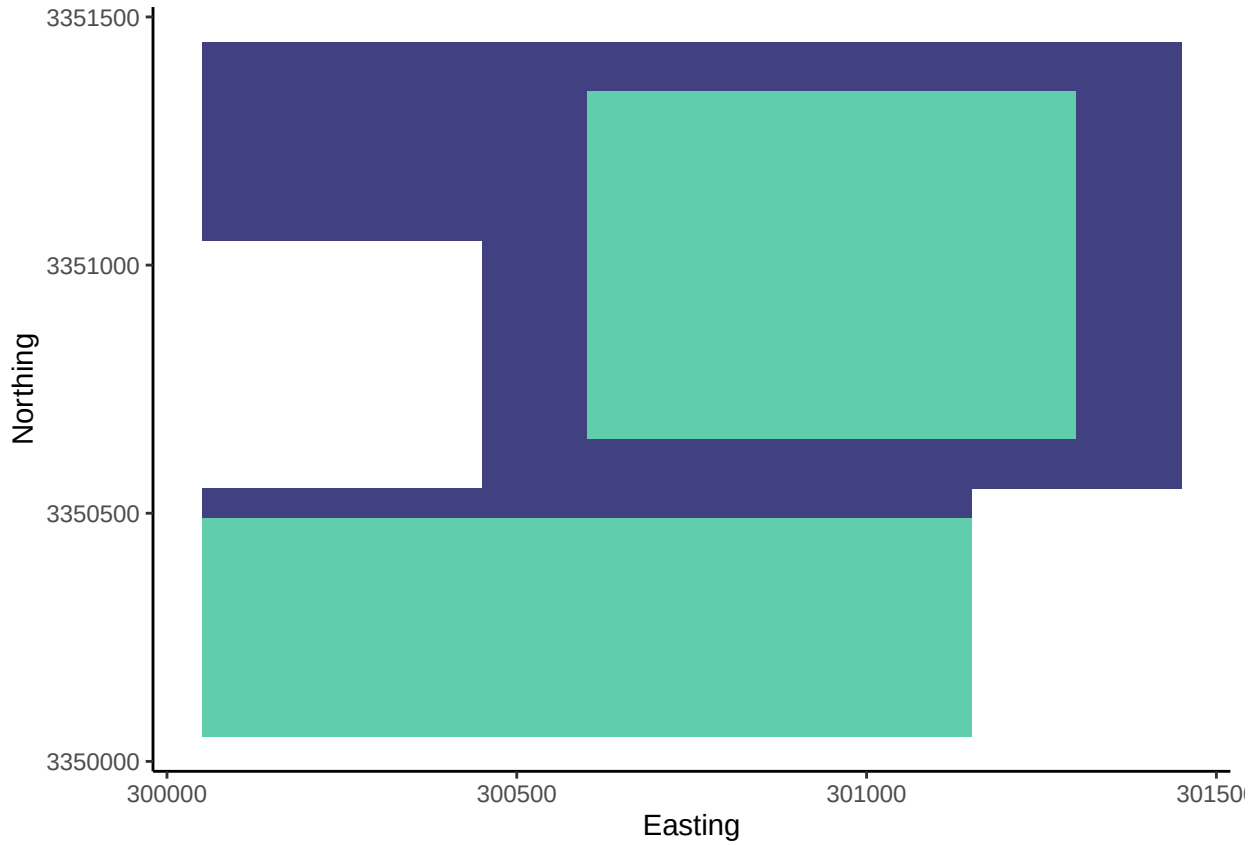
For number 4, most recipient sites in Florida will be in UTM zone 17, but refer to Figure 3 of the SOP to check that your shapefile is not in zone 16. We will use zone 17 for this tutorial, which can be verified by:

```
#Call the shapefile object to check the site CRS (Coordinate Reference System)
site.rast
```

```
#> class       : SpatRaster
#> dimensions  : 280, 280, 1  (nrow, ncol, nlyr)
#> resolution  : 5, 5  (x, y)
#> extent      : 300050, 301450, 3350050, 3351450  (xmin, xmax, ymin, ymax)
#> coord. ref. : +proj=utm +zone=17 +ellps=GRS80 +towgs84=0,0,0,0,0,0,0 +units=m +no_defs
#> source      : memory
#> name        : lyr.1
#> min value   :     1
#> max value   :     3
```

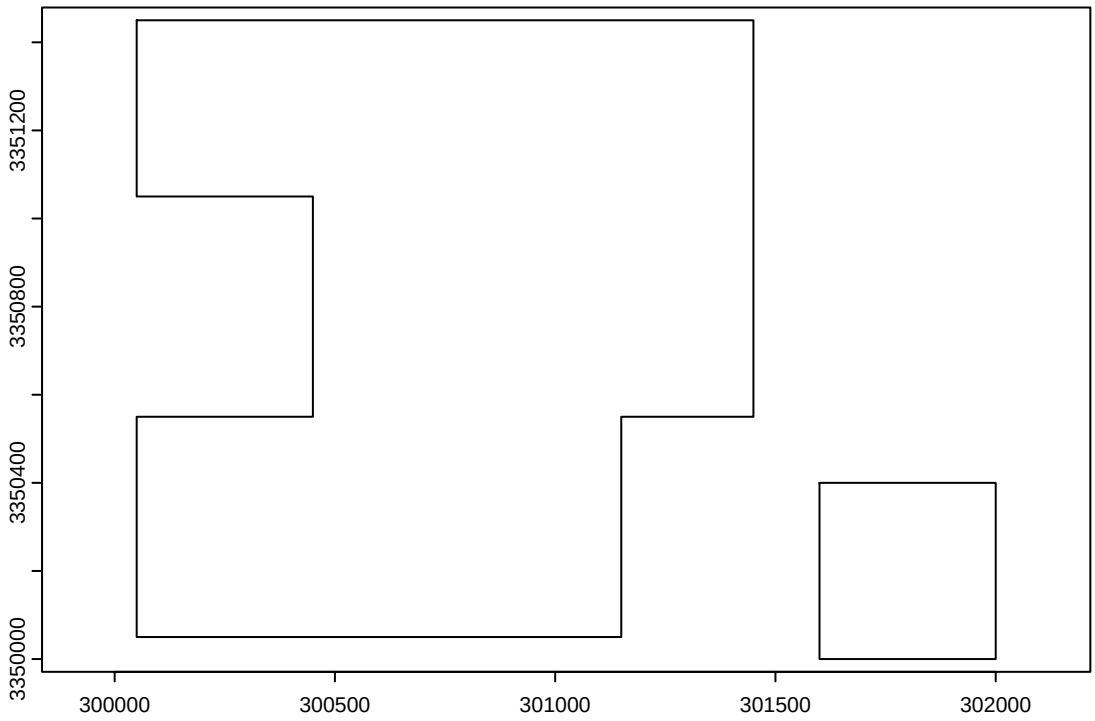
For number 5, we have created our own raster with pen edges, so we know where they are. A shapefile or raster file with pen locations would be the easiest method of visualizing pen locations. Otherwise, an aerial map or site visit would be another option. Plot raster as a reminder:

```
ggplot(temp) +
  geom_tile(aes(x = x, y = y, fill = lyr.1)) +
  scale_fill_viridis_c(option = "mako", begin = 0.3, end = 0.8) +
  xlab("Easting") +
  ylab("Northing") +
  theme_classic() +
  theme(legend.position = "none")
```



If you have multiple shapefiles which need to be brought together to make up the entire study site, we can do this using existing R functions. For the below code to work, the shapefiles need to be converted into StatVector objects using the vect() function. Then we can use rbind to merge them.

```
#We can enter as many shapefiles as needed to merge into one
new_site <- rbind(site.shp, site.shp.2)
#We will use base plot to visualize this newly merged site
plot(new_site)
```



Function:placing transect lines - *tran_place*

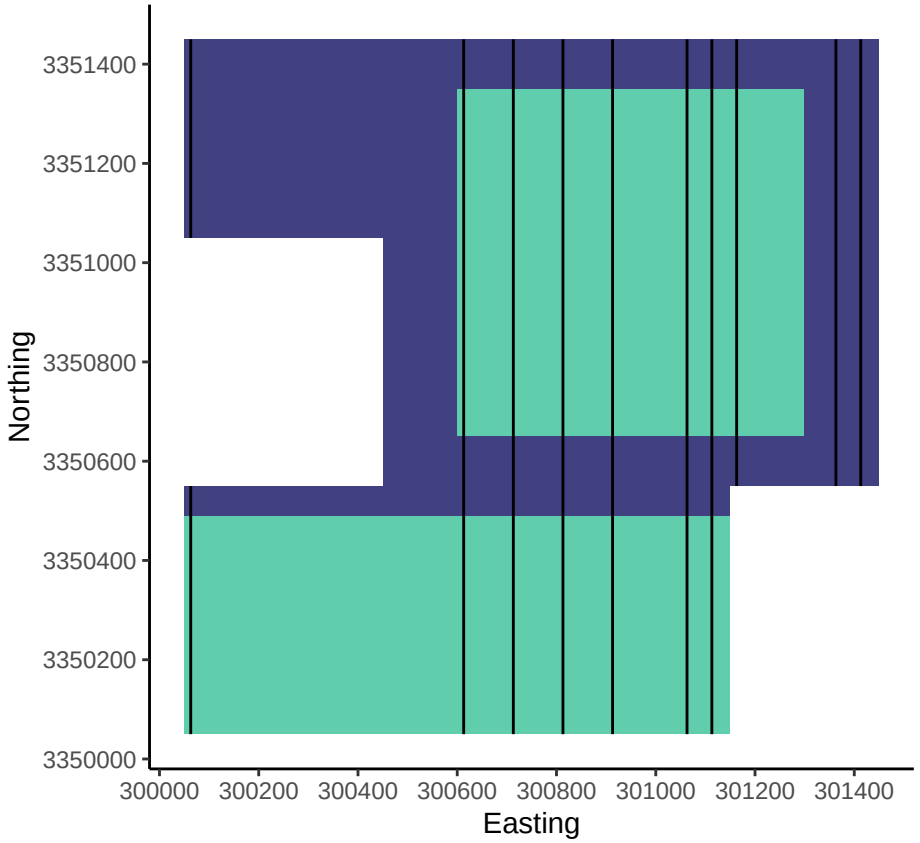
Check function help documentation:

```
?tran_place
```

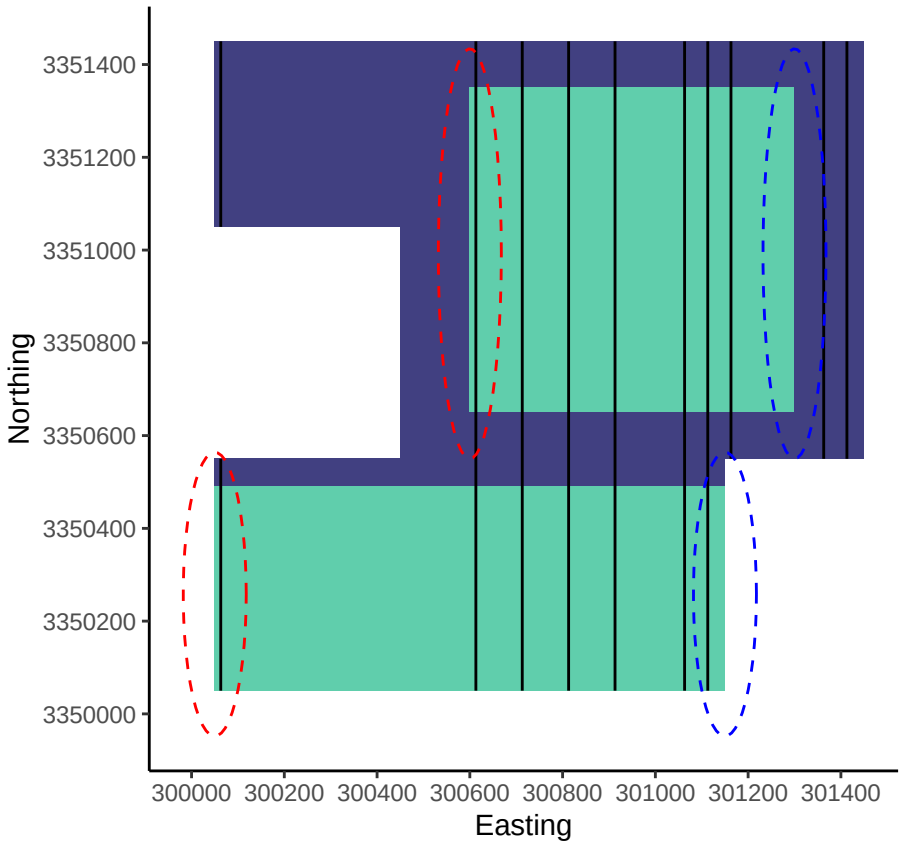
We are going to use our first function, *tran_place*, to delineate transect lines across our recipient site depicting the lines to be walked for the pilot survey. We need to provide the function with some key information, depending on the size of the area to be surveyed (number 3 of checklist). We have a 400 acre site, so we can refer to figure 4 of the SOP to know what characteristics are needed. For this size, we need 50% of the overall transects to be surveyed for the pilot survey, with a distance of 50m between adjacent transects.

Importantly, this is where we decide which direction the transect lines will be running. The options are either North-South ("N-S"), East-West ("E-W"), Northeast-Southwest ("NE-SW"), or Northwest-Southeast ("NW-SE"). In an attempt to troubleshoot potential problems, we will run the function using a "N-S" orientation. We need to provide the site shapefile, UTM zone and if we want the lines to be plotted and saved to the working directory.

```r
#Set the distance between transects
dist <- 50

#The tran_place function produces a SpatVector object
pilot.lines <- tran_place(tran.dist = dist, pil.perc = 50, direction = "N-S",
                          site.poly = site.shp, zone = 17, plot = F, save = F)
#We can review information about the SpatVector object by running:
pilot.lines
#>  class       : SpatVector
#>  geometry    : lines
#>  dimensions  : 10, 0  (geometries, attributes)
#>  extent      : 300063, 301413, 3350050, 3351450  (xmin, xmax, ymin, ymax)
#>  coord. ref. :

#Plot the raster and lines to assess position relative to pens
temp<-as.data.frame(site.rast, xy = T)
temp <- temp %>% na.omit()
ggplot() +
  geom_tile(data = temp, aes(x = x, y = y, fill = lyr.1)) +
  geom_spatvector(data = pilot.lines, aes(),
                  color = "black") +
  #geom_point(data = crd.test, aes(x = x, y = y)) +
  scale_fill_viridis_c(option = "mako", begin = 0.3, end = 0.8) +
  xlab("Easting") +
  ylab("Northing") +
  theme_classic() +
  theme(legend.position = "none")
```



We now have the first potential pilot lines that we could survey. However, when choosing transect direction, it is extremely important to consider the position of pen edges in relation to the transect lines. In our example, consider the following locations specifically:



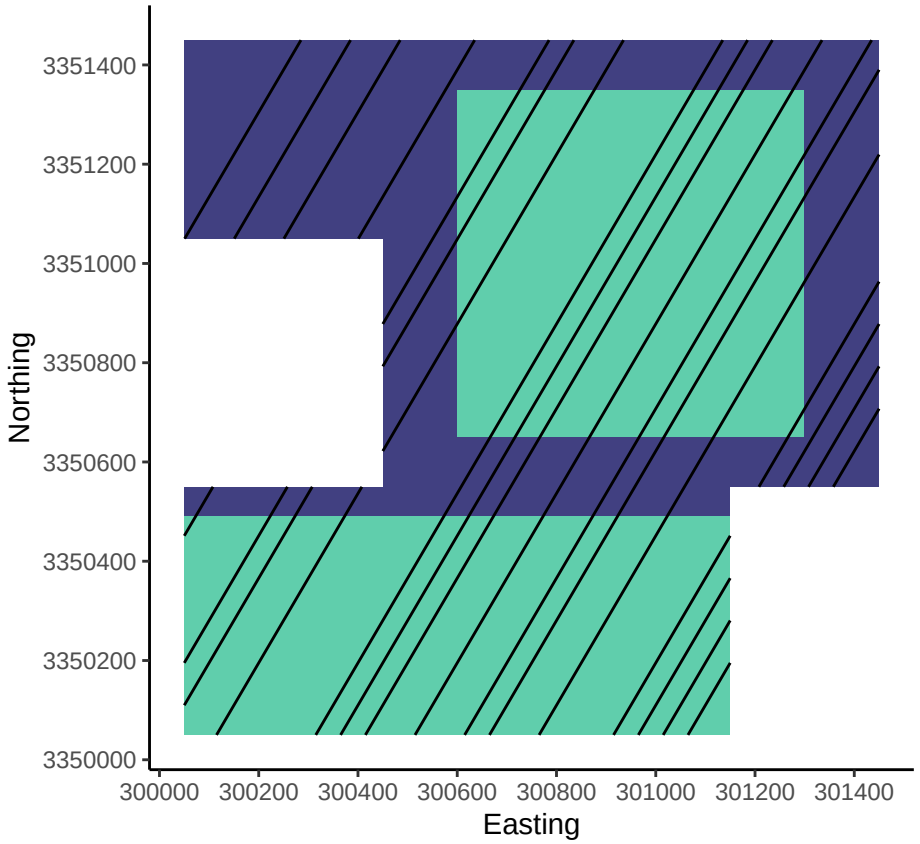
The dashed red lines show locations where the transect lines run directly on top of pen edge locations. Where this occurs, we are likely to overestimate the true density of tortoises. This is because translocated gopher tortoises are likely to cluster around these pen edges, because they typically offer built-up sandy habitat (perfect for burrowing), but also limit outward movement - which is typical for translocated reptiles. As a result of the transect line running along one of these highly clustered edges, the encounter rate of a survey is inflated and ultimately the estimates are highly overestimated (SOP Figure 15).

On the other hand, the blue dashed lines show where a pen edge has been completely missed by the placement of transect lines. We have also found that underestimations of density can occur due to an uneven sampling across pen edges, particularly where highly clustered pen edges are missed entirely (SOP Figure 15).

Due to the poor placement of transect lines shown in the above figure, we will now try running the *tran_place* function again using a NE-SW direction. If we tried an E-W direction, we could easily encounter the same issue as N-S, due to the high linearity of our simulated study site. Let's see what the NE-SW looks like:

```
#Now trying the same parameters, but with NE-SW
pilot.lines <- tran_place(tran.dist = dist, pil.perc = 50, direction = "NE-SW",
                          site.poly = site.shp, zone = 17, plot = F, save = T)
#We can review information about the SpatialLines object by running:
pilot.lines
#>  class       : SpatVector
#>  geometry    : lines
#>  dimensions  : 18, 0  (geometries, attributes)
#>  extent      : 300050, 301450, 3350050, 3351450  (xmin, xmax, ymin, ymax)
#>  coord. ref. :

#Plot the raster and lines to assess position relative to pens
```

```
temp<-as.data.frame(site.rast, xy = T)
temp <- temp %>% na.omit()
ggplot() +
  geom_tile(data = temp, aes(x = x, y = y, fill = lyr.1)) +
  geom_spatvector(data = pilot.lines, aes(),
                  color = "black") +
  scale_fill_viridis_c(option = "mako", begin = 0.3, end = 0.8) +
  xlab("Easting") +
  ylab("Northing") +
  theme_classic() +
  theme(legend.position = "none")
```



These new pilot lines give a better distribution of transects across pen edges. So we will be moving forward using these pilot transect lines.

Function:getting start and end locations - *start_end*

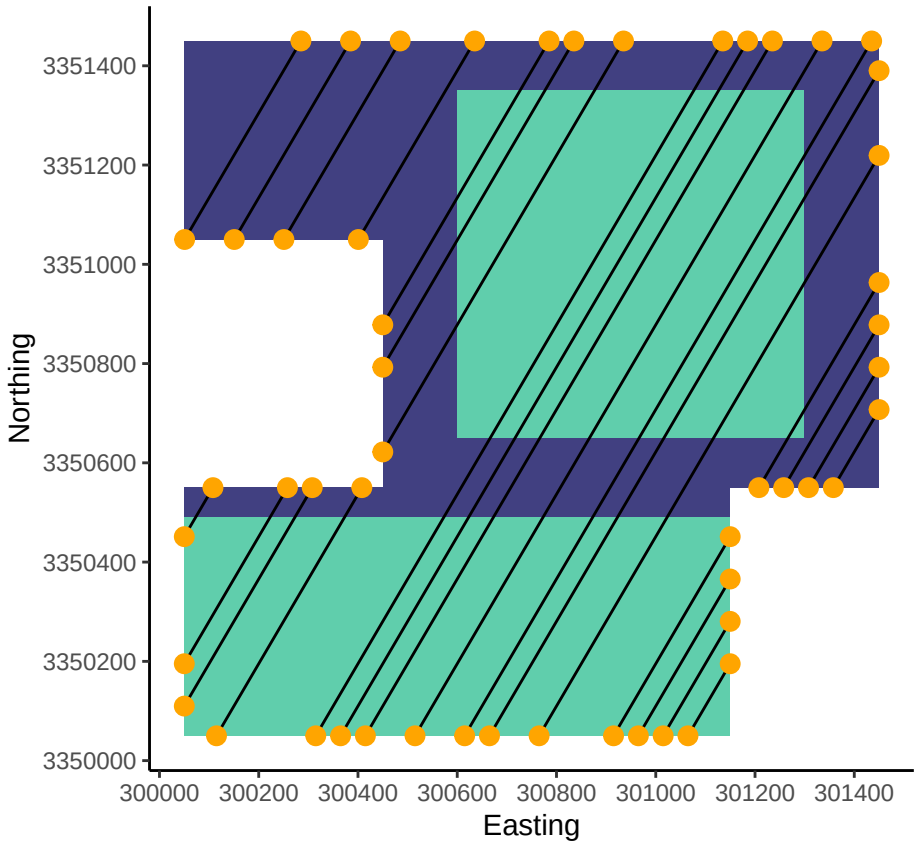
Check function help documentation:

```
?start_end
```

Now we have the transect lines needed for our pilot surveys, we are going to extract the start and end locations of each line. These will be exported to a GPS device so that we can adhere to line trajectories in the field.

For the start_end function to work, we need to provide the pilot lines that we created using the *tran_place* function, and the utm zone as before. By setting save = T, this will save the points to the working directory.

```
se.pts <- start_end(lines = pilot.lines, zone = 17, save = T, plot = F)
#> Warning in showSRID(uprojargs, format = "PROJ", multiline = "NO", prefer_proj
#> = prefer_proj): Discarded datum Unknown based on WGS84 ellipsoid in Proj4
#> definition

#Change to dataframe for plotting
se_loc <- data.frame(se.pts)
#We can also check the coordinated by plotting against the site and transects
ggplot() +
  geom_tile(data = temp, aes(x = x, y = y, fill = lyr.1)) +
  geom_spatvector(data = pilot.lines, aes(),
                  color = "black") +
  geom_point(data = se_loc, aes(x = Start_x, y = Start_y), size = 3, color = "orange") +
  geom_point(data = se_loc, aes(x = End_x, y = End_y), size = 3, color = "orange") +
  scale_fill_viridis_c(option = "mako", begin = 0.3, end = 0.8) +
  xlab("Easting") +
  ylab("Northing") +
  theme_classic() +
  theme(legend.position = "none")
```



The orange points now delineate either where a transect line should start or end during a survey. Observers should survey from one orange point, to the opposite orange point which is connected via the black transect line.

Function:delineate locations for vegetation sampling - *veg_plot*

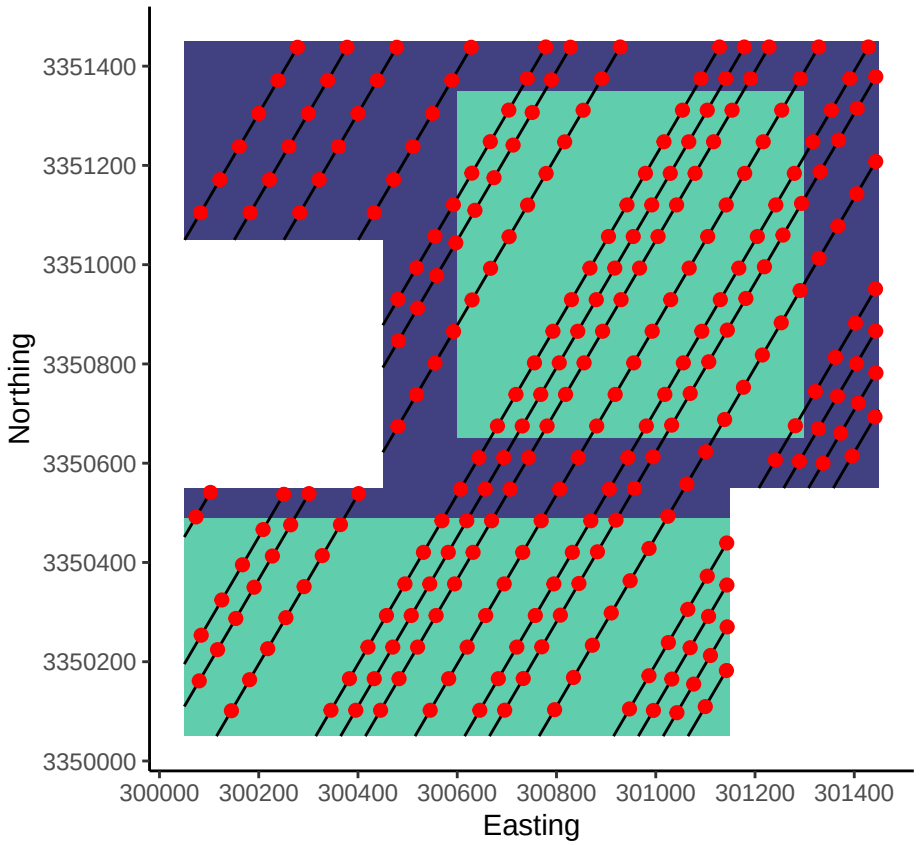
Check function help documentation:

```
?veg_plot
```

While doing the surveys, collecting vegetation obstruction data can be extremely important for accurate tortoise density estimation. In the next function, we are going to determine where these vegetation plots are going to be conducted.

The locations are determined in a very similar manner to the *start_end* function where we need to provide the pilot lines. However, we need to also provide the set distance in meters between vegetation points (intv; default = 75m). We provide the lines, via the start and end points created using the start_end function.

```
#The csv file will be in the working directory
start.end <- read_csv("./start_end_coord.csv")
#> Rows: 26 Columns: 5
#> -- Column specification --------------------------------------------------------
#> Delimiter: ","
#> dbl (5): Start_x, Start_y, End_x, End_y, Transect_ID
#>
#> i Use `spec()` to retrieve the full column specification for this data.
#> i Specify the column types or set `show_col_types = FALSE` to quiet this message.
veg.loc <- veg_plot(points = start.end, intv = 75, save = T, plot = F)
#Change to dataframe for plotting
veg_loc <- data.frame(veg.loc)
#We can also check the coordinated by plotting against the site and transects
ggplot() +
  geom_tile(data = temp, aes(x = x, y = y, fill = lyr.1)) +
  geom_spatvector(data = pilot.lines, aes(),
                  color = "black") +
  geom_point(data = veg_loc, aes(x = coords.x1, y = coords.x2), size = 2,
             color = "red") +
  scale_fill_viridis_c(option = "mako", begin = 0.3, end = 0.8) +
  xlab("Easting") +
  ylab("Northing") +
  theme_classic() +
  theme(legend.position = "none")
```



The red points now show where observers will stop and collect vegetation obstruction data. Consistent with our function, these are set at 75m apart.

We have come to the end of our "before sampling" section. Let's review our working directory folders to see if we have everything needed to get out in the field and perform our pilot survey.

Below shows all the files we have in our working directory. We need the following files to continue:

1. Transect_lines.shp
2. Start_end_locations.shp
3. Veg_plots.shp

```
#List all files in working directory
list.files()
```

After pilot

Function:calculate pilot effort - *samp_effort*

Check function help documentation:

```
?samp_effort
```

Once the pilot survey has been conducted, as outlined in the SOP, we can take the start and end locations for all surveyed lines and calculate the effort. Our *start_end* function creates a csv file with start and end

locations for each transect line. This mirrors the expected dataframe from an actual field survey. The order of the columns is the most important for the function. See the SOP for information on how to properly format the coordinate data during and following an LTDS survey.

To calculate the effort (distance in meters), we provide the start and end locations of each transect line walked during the pilot survey. As before, we also need the utm zone. We also have the option to plot the transect lines. Since we are using lines that we created, we will not re-plot the lines, however, this could give us the opportunity to assess if there are any errors with the data collection or function. For example, if the lines do not look anything like what was actually walked (besides being perfectly straight), then there is likely an issue.

```
#The csv file will be in the working directory
start.end <- read_csv("./start_end_coord.csv")
#> Rows: 26 Columns: 5
#> -- Column specification --------------------------------------------------------
#> Delimiter: ","
#> dbl (5): Start_x, Start_y, End_x, End_y, Transect_ID
#>
#> i Use `spec()` to retrieve the full column specification for this data.
#> i Specify the column types or set `show_col_types = FALSE` to quiet this message.

effort <- samp_effort(points = start.end, zone = 17, plot = F)

#View resulting effort - this is another place to check if the distance is
#corresponding to the actual effort in the field.
effort
#> [1] 19464.4
```

As we can see from above, the effort for the pilot survey was 19464m (~19.5km).

Function:determine lines for full LTDS survey - *samp_full*

Check function help documentation:

```
?samp_full
```

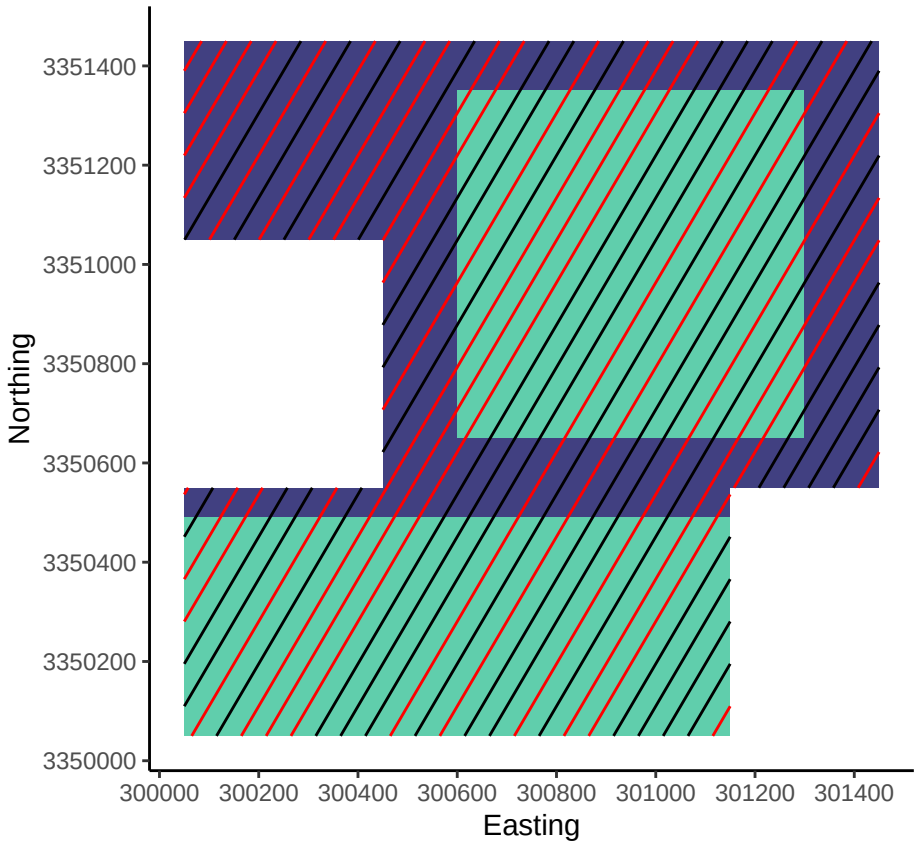
Now we have the distance covered during the pilot survey, we can calculate the remaining effort needed to attain our target CV (default = 17%). The following function works similarly to the *tran_place* function used at the beginning of the tutorial. However, we do not want to duplicate effort, so we now build upon the pilot survey to add additional transect lines. This means that the pilot line effort will be included within the final density estimation.

There are several bits of information that we need to provide to get the function to work. Firstly, we need the effort from the *samp_effort* function. We need to state the direction of the transects (which needs to match the direction of the pilot survey), which in this case is NE-SW. Next, we need the number of occupied burrows detected during the pilot survey. If there were a lot of occupied burrows discovered, we can pull this information from the data. However, for the purpose of this vignette, we will set the number of occupied burrows (ntort) to 120. We then need to provide the desired CV from the survey. For our tutorial, we will aim for a CV of 17%. As with the other functions, we lastly need to provide the utm zone.

```
full.ltds <- samp_full(effort = effort, site.poly = site.shp, direction = "NE-SW",
                       ntort = 120, cv = 0.17, zone = 17, plot = F, save = F)
#> [1] "Target CV attained. CV = 15.81%"
```

By setting the ntort to 120, the object is not created. In the console, we can see the statement "Target CV attained". This means that by finding 120 occupied burrows with the current amount of pilot effort, we have already achieved a CV that is lower than our target. In this case, the CV is 15.81%. Now, let's try the opposite, what if we only detected 50 occupied burrows?

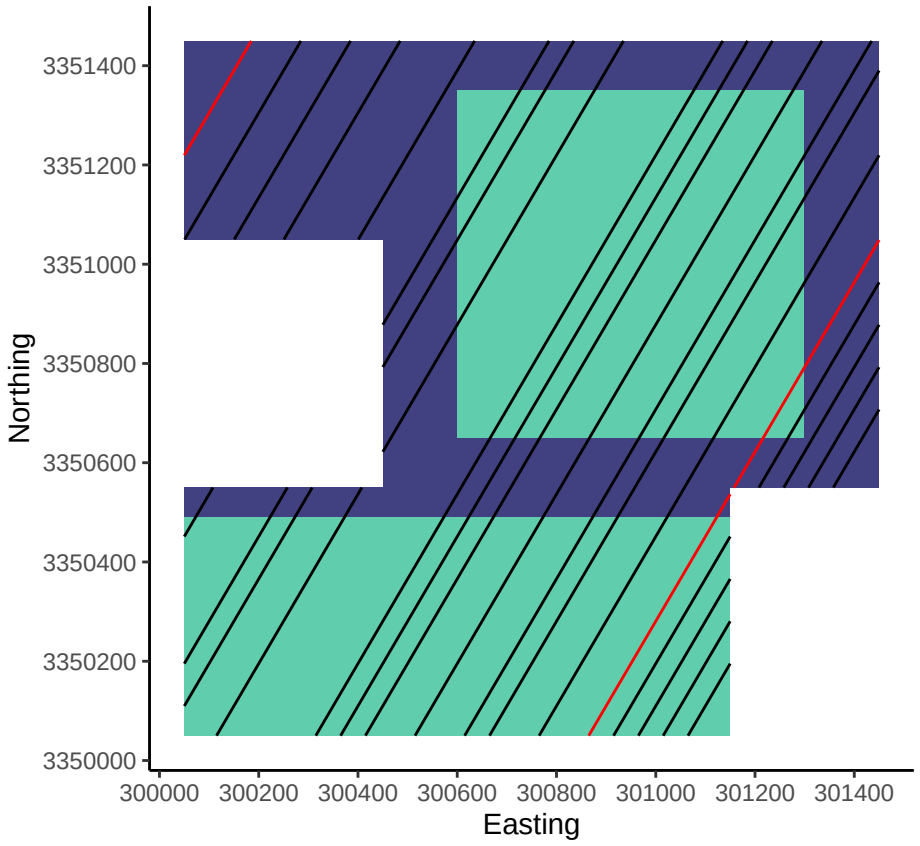
```
full.ltds <- samp_full(effort = effort, site.poly = site.shp, direction = "NE-SW",
                       ntort = 50, cv = 0.17, zone = 17, plot = F, save = F)
#> [1] "CV = 24.49%"

ggplot() +
  geom_tile(data = temp, aes(x = x, y = y, fill = lyr.1)) +
  geom_spatvector(data = pilot.lines, aes(),
                  color = "black") +
  geom_spatvector(data = full.ltds, aes(),
                  color = "red") +

  scale_fill_viridis_c(option = "mako", begin = 0.3, end = 0.8) +
  xlab("Easting") +
  ylab("Northing") +
  theme_classic() +
  theme(legend.position = "none")
```



With this much lower encounter rate, our CV was much higher at 24.49%. We can see from the accompanying plot that we would now need to survey all possible transects to achieve a more adequate CV.

Let's set the number of detected burrows to 100 to move forward with the vignette example.

```
full.ltds <- samp_full(effort = effort, site.poly = site.shp, direction = "NE-SW",
                       ntort = 100, cv = 0.17, zone = 17, plot = F, save = F)
#> [1] "CV = 17.32%"

ggplot() +
  geom_tile(data = temp, aes(x = x, y = y, fill = lyr.1)) +
  geom_spatvector(data = pilot.lines, aes(),
                  color = "black") +
  geom_spatvector(data = full.ltds, aes(),
                  color = "red") +
  scale_fill_viridis_c(option = "mako", begin = 0.3, end = 0.8) +
  xlab("Easting") +
  ylab("Northing") +
  theme_classic() +
  theme(legend.position = "none")
```

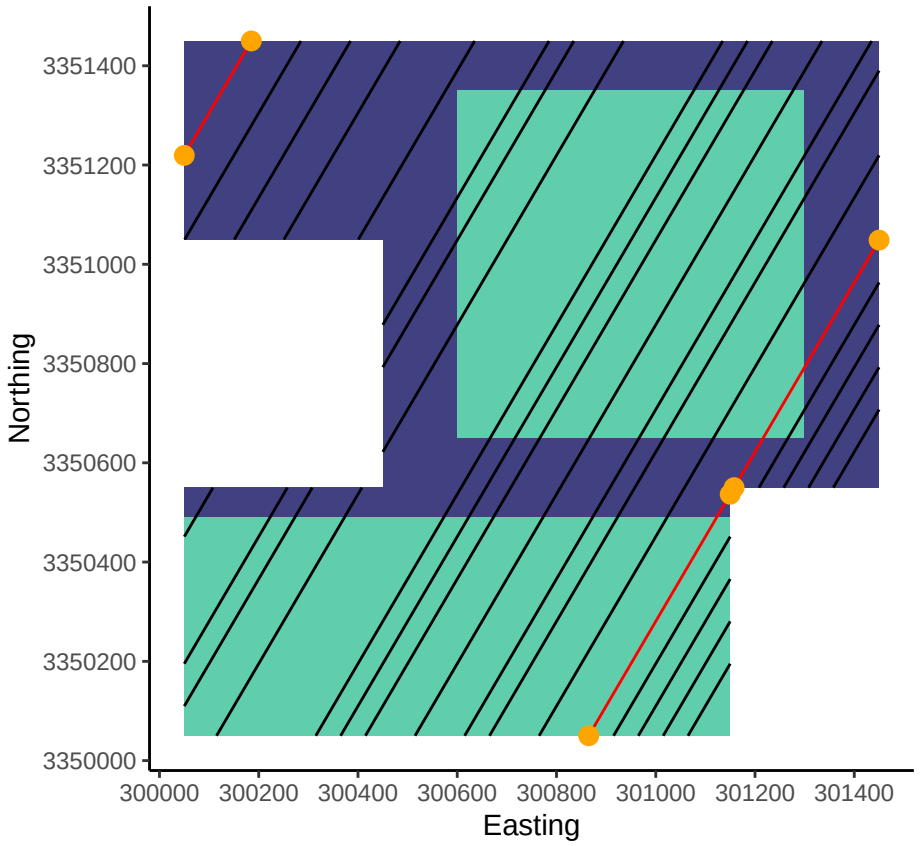


By assigning 100 discovered, occupied burrows we reached a CV of 17.32%. The function has delineated new lines (red), that need to be surveyed to attain a CV under 17%. We will move forward in the vignette using the lines from discovering 100 burrows.

As with the pilot transects, we need to know the start and end locations to make it easier to find and traverse the transect lines in the field. To do this, we can simply re-run the *start_end* function again using our full.ltds lines.

IMPORTANT - Before running the next lines, rename the start_end files so that they are not overwritten by the function.

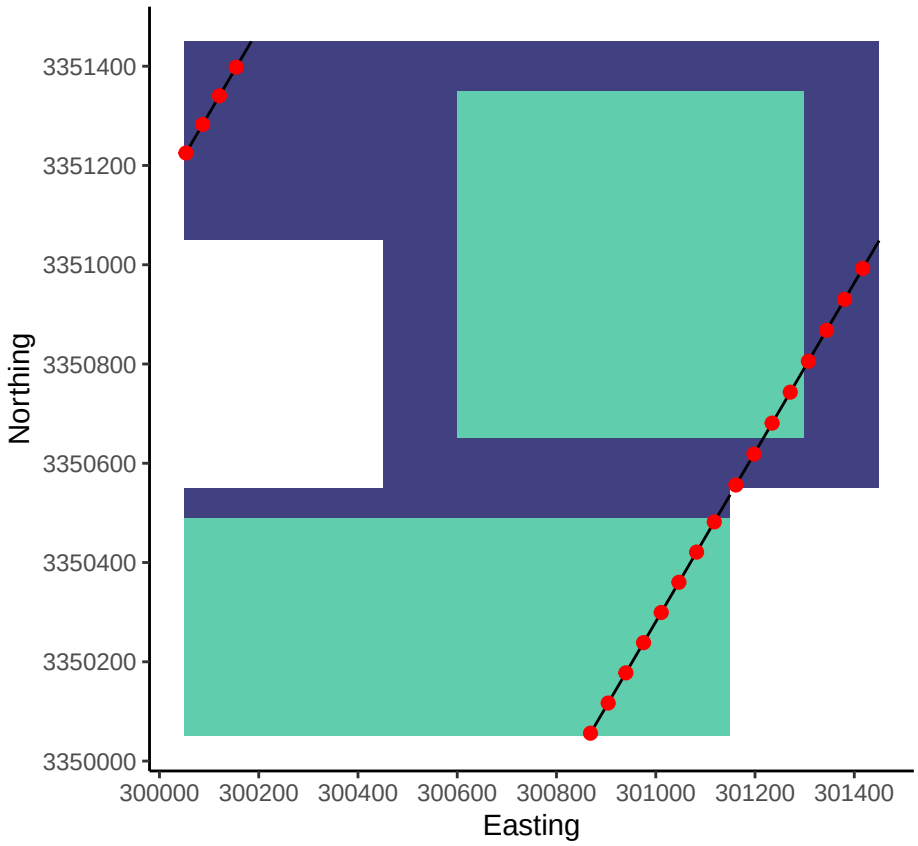
```r
se.pts <- start_end(lines = full.ltds, zone = 17, save = T, plot = F)

#We can also check the coordinates by plotting against the site and transects
#Change to dataframe for plotting
se_loc <- data.frame(se.pts)
#We can also check the coordinated by plotting against the site and transects
ggplot() +
  geom_tile(data = temp, aes(x = x, y = y, fill = lyr.1)) +
  geom_spatvector(data = pilot.lines, aes(),
                  color = "black") +
  geom_spatvector(data = full.ltds, aes(),
                  color = "red") +
  geom_point(data = se_loc, aes(x = Start_x, y = Start_y), size = 3, color = "orange") +
  geom_point(data = se_loc, aes(x = End_x, y = End_y), size = 3, color = "orange") +
  scale_fill_viridis_c(option = "mako", begin = 0.3, end = 0.8) +
  xlab("Easting") +
  ylab("Northing") +
  theme_classic() +
  theme(legend.position = "none")
```



Lastly, we need to know the location for collecting vegetation obstruction data as before. Again, re-run the veg_plot function using the full.ltds lines to delineate the locations.

IMPORTANT - Before running the next lines, rename the veg_plot files so that they are not overwritten by the function.

```
#Re-run the veg_plot function, using the new start_end csv in the wd.
start.end <- read_csv("./start_end_coord.csv")
#> Rows: 3 Columns: 5
#> -- Column specification --------------------------------------------------------
#> Delimiter: ","
#> dbl (5): Start_x, Start_y, End_x, End_y, Transect_ID
#>
#> i Use `spec()` to retrieve the full column specification for this data.
#> i Specify the column types or set `show_col_types = FALSE` to quiet this message.
veg.loc <- veg_plot(start.end, intv = 75, save = T, plot = F)

#Change to dataframe for plotting
veg_loc <- data.frame(veg.loc)
ggplot() +
  geom_tile(data = temp, aes(x = x, y = y, fill = lyr.1)) +
  geom_spatvector(data = full.ltds, aes(),
                  color = "black") +
  geom_point(data = veg_loc, aes(x = coords.x1, y = coords.x2), size = 2,
             color = "red") +
  scale_fill_viridis_c(option = "mako", begin = 0.3, end = 0.8) +
  xlab("Easting") +
  ylab("Northing") +
  theme_classic() +
  theme(legend.position = "none")
```



We have come to the end of our "after pilot" section. Let's review our working directory folders to see if we

have everything needed to get out in the field and perform our full LTDS survey.

Below shows all the files we have in our working directory. We need the following files to continue:

1. Add_transect_lines.shp
2. Start_end_locations.shp
3. Veg_plots.shp

```
#List all files in working directory
list.files()
```

After full LTDS

We now presume that the full LTDS survey has been conducted. This survey will ultimately give us a dataframe with several important variables. Import the data by:

```
full.data <- read_csv("./LTDS_example_data_new.csv")
#> Rows: 263 Columns: 7
#> -- Column specification --------------------------------------------------------
#> Delimiter: ","
#> chr (2): Occupied, Burrow_condition
#> dbl (5): Transect_ID, Burrow_width, Distance, X, Y
#>
#> i Use `spec()` to retrieve the full column specification for this data.
#> i Specify the column types or set `show_col_types = FALSE` to quiet this message.

#We can summarize the data
summary(full.data)
#>   Transect_ID     Burrow_width      Distance          Occupied
#>  Min.   : 1.00   Min.   : 50.5   Min.   : 0.02674   Length:263
#>  1st Qu.: 9.00   1st Qu.:149.1   1st Qu.: 1.71504   Class :character
#>  Median :14.00   Median :241.0   Median : 3.72894   Mode  :character
#>  Mean   :13.25   Mean   :210.0   Mean   : 4.36837
#>  3rd Qu.:17.00   3rd Qu.:279.6   3rd Qu.: 6.48641
#>  Max.   :26.00   Max.   :307.9   Max.   :14.83788
#>  Burrow_condition         X               Y
#>  Length:263         Min.   :300058   Min.   :3350053
#>  Class :character   1st Qu.:300542   1st Qu.:3350402
#>  Mode  :character   Median :300769   Median :3350777
#>                     Mean   :300783   Mean   :3350750
#>                     3rd Qu.:301053   3rd Qu.:3351087
#>                     Max.   :301444   Max.   :3351447
```

Function:truncate data - *LTDS_crop*

Check function help documentation:

```
?LTDS_crop
```

So that we can use the data to estimate the density of tortoises at the site, we are going to slightly format the input. Specifically, we are going to truncate the data so that any burrows found over half the distance between the lines are removed; and we are also going to remove the top distance outliers (5%).

The function we are using simply needs two bits of information. We need the data from the full survey, and we also need to set the distance between transects, which will determine the truncation distance. We have created the object "dist" earlier in the vignette which we can input into the function.

```
data.new <- LTDS_crop(ltds_data = full.data, tran.dist = dist)

#We can summarize the data again for comparison.
summary(data.new)
#>   Transect_ID     Burrow_width      Distance          Occupied
#>  Min.   : 1.00   Min.   : 50.5   Min.   : 0.02674   Length:249
#>  1st Qu.: 9.00   1st Qu.:149.6   1st Qu.: 1.64439   Class :character
#>  Median :14.00   Median :246.1   Median : 3.51532   Mode  :character
#>  Mean   :13.24   Mean   :211.6   Mean   : 3.94488
#>  3rd Qu.:17.00   3rd Qu.:279.8   3rd Qu.: 6.06748
#>  Max.   :26.00   Max.   :307.9   Max.   :10.20368
#>  Burrow_condition         X                Y
#>  Length:249         Min.   :300058   Min.   :3350053
#>  Class :character   1st Qu.:300546   1st Qu.:3350401
#>  Mode  :character   Median :300765   Median :3350781
#>                     Mean   :300783   Mean   :3350751
#>                     3rd Qu.:301050   3rd Qu.:3351087
#>                     Max.   :301444   Max.   :3351447
```

Function:estimate the density - *dens_est*

Check function help documentation:

```
?dens_est
```

We are finally at the penultimate function of the *LTDSMethod* package. We are now ready to calculate the density of tortoises at the recipient site.

For the function to work and estimate the density, we need to provide three main bits of information. We need the cropped ltds data from *LTDS_crop*, the effort resulting from *samp_effort*, and lastly we need the total area of the recipient site in acres.

Remember, we have now conducted the full LTDS survey, which means that our resulting effort will be greater than the *samp_effort* function has previously calculated (unless no further transect lines were walked during the full LTDS; i.e., the target CV had been attained during the pilot). We therefore need to re-run the *samp_effort* function using the additional lines. You will need to combine the start and end csv files from the pilot and full survey to calculate the correct amount of effort.

For this vignette, we have provided a combined start and end coordinate csv file to work with. This can be read in as before:

```
#The csv file will be in the working directory
start.end.2 <- read_csv("./start_end_coord_full.csv")
#> Rows: 29 Columns: 5
#> -- Column specification --------------------------------------------------------
#> Delimiter: ","
#> dbl (5): Start_x, Start_y, End_x, End_y, Transect_ID
#>
#> i Use `spec()` to retrieve the full column specification for this data.
```

```
#> i Specify the column types or set `show_col_types = FALSE` to quiet this message.

effort.2 <- samp_effort(points = start.end.2, zone = 17, plot = F)

#View resulting effort - this is another place to check if the distance is
#corresponding to the actual effort in the field.
effort.2
#> [1] 21299.14
```

The final effort for the full LTDS survey was 21299m (~21.3km). We can enter the effort.2 object into our function.

```
#Calculate density
tort.dens <- dens_est(ltds_data = data.new, effort = effort.2, area = 400)
#> Defining model
#> Building model
#> Setting data and initial values
#> Running calculate on model
#>   [Note] Any error reports that follow may simply reflect missing values in model variables.
#> Checking model sizes and dimensions
#>   [Note] This model is not fully initialized. This is not an error.
#>          To see which variables are not initialized, use model$initializeInfo().
#>          For more information on model initialization, see help(modelInitialization).
#> Checking model calculations
#> [Note] NAs were detected in model variables: dist.a, logProb_dist.a, sigma, logProb_sigma, sigma2, o
#> Compiling
#>   [Note] This may take a minute.
#>   [Note] Use 'showCompilerOutput = TRUE' to see C++ compilation details.
#> Running chain 1 ...
#> |-------------|-------------|-------------|-------------|
#> |-------------------------------------------------------|
#> Running chain 2 ...
#> |-------------|-------------|-------------|-------------|
#> |-------------------------------------------------------|
#> Running chain 3 ...
#> |-------------|-------------|-------------|-------------|
#> |-------------------------------------------------------|
#> Warning: `as_data_frame()` was deprecated in tibble 2.0.0.
#> i Please use `as_tibble()` instead.
#> i The signature and semantics have changed, see `?as_tibble`.
#> i The deprecated feature was likely used in the LTDSMethod package.
#>   Please report the issue to the authors.

#The dens_est function produces a small dataframe with the estimated density
#and 95% credible interval around that estimate. This gives us a range of
#possible densities at the recipient site.
tort.dens$density #Mean estimate
#> [1] 1.573504
tort.dens$density.lo #95% CI Low
#> [1] 1.284838
tort.dens$density.hi #95% CI High
#> [1] 1.918179
```

```
#This means that our total number of tortoises estimated at the site is:
tort.pop <- tort.dens*400 #Change number according to acreage at site
tort.pop$density #Mean estimate
#> [1] 629.4016
tort.pop$density.lo #95% CI Low
#> [1] 513.9351
tort.pop$density.hi #95% CI High
#> [1] 767.2715
```

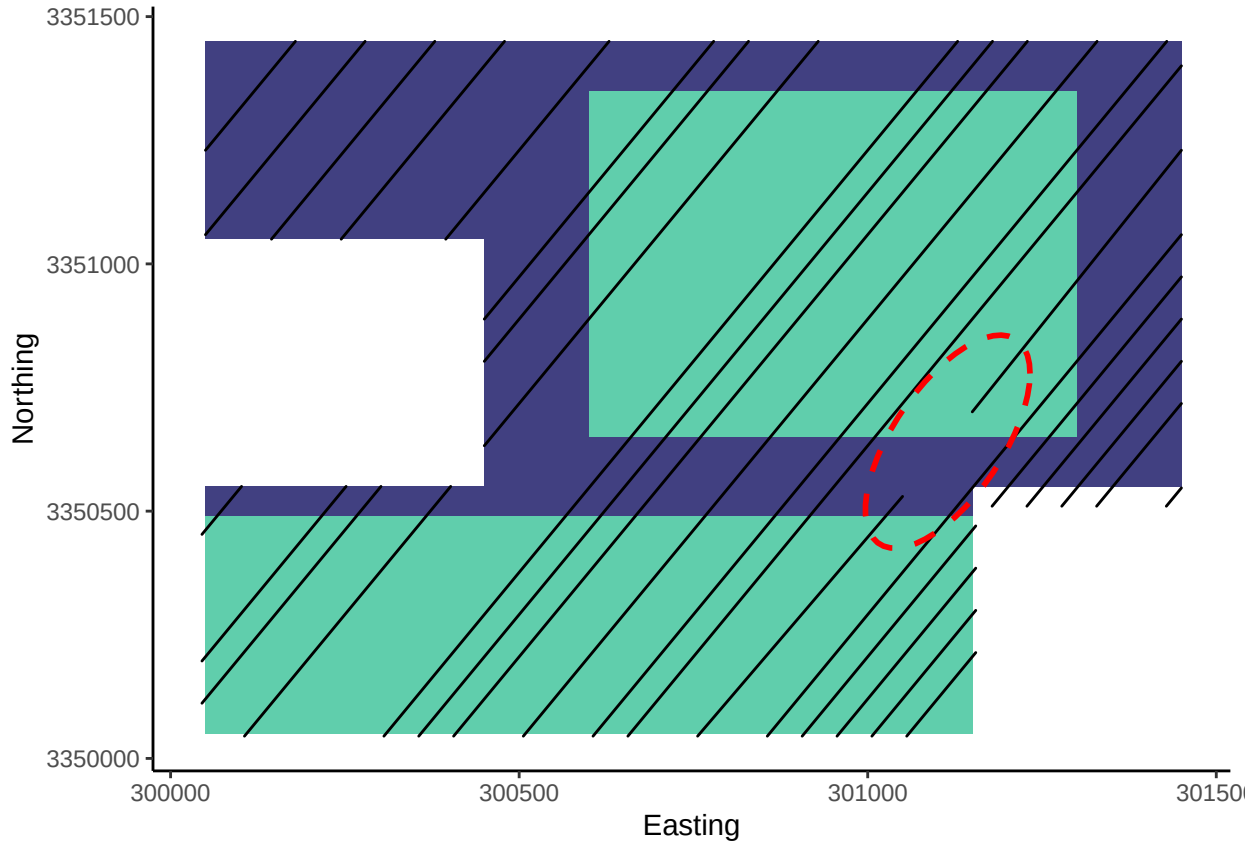
Our final density estimate at the site is 1.57 tortoises per acre with a range of 1.28 - 1.92 tortoises/acre based on our 95% credible interval values. This means that there is an estimated approximately 629 tortoises at our site, with a possible range of 513-767 tortoises. This is fabricated data to fit in with the vignette training, so the accuracy of this estimation is unknown. See the SOP for simulation results.

What if there was an obstruction on one of the transect lines? Instructions for encountering an obstruction in the field (such as a flooded area or fallen tree), can be found in the accompanying SOP. The new coordinates will be read in as before:

```
start.end.3 <- read_csv("./start_end_coord_obst.csv")
#> Rows: 30 Columns: 5
#> -- Column specification --------------------------------------------------------
#> Delimiter: ","
#> dbl (5): Start_x, Start_y, End_x, End_y, Transect_ID
#>
#> i Use `spec()` to retrieve the full column specification for this data.
#> i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

These depict the following effort in the field:

```
#> Rows: 30 Columns: 5
#> -- Column specification --------------------------------------------------------
#> Delimiter: ","
#> dbl (5): Start_x, Start_y, End_x, End_y, Transect_ID
#>
#> i Use ‘spec()‘ to retrieve the full column specification for this data.
#> i Specify the column types or set ‘show_col_types = FALSE‘ to quiet this message.
```



Our new effort can be calculated as before.

```
#The csv file will be in the working directory
start.end.3 <- read_csv("./start_end_coord_obst.csv")
#> Rows: 30 Columns: 5
#> -- Column specification --------------------------------------------------------
#> Delimiter: ","
#> dbl (5): Start_x, Start_y, End_x, End_y, Transect_ID
#>
#> i Use `spec()` to retrieve the full column specification for this data.
#> i Specify the column types or set `show_col_types = FALSE` to quiet this message.

effort.3 <- samp_effort(points = start.end.3, zone = 17, plot = F)

#View resulting effort - this is another place to check if the distance is
#corresponding to the actual effort in the field.
effort.3
#> [1] 21101.17
```

The effort when factoring in the site obstruction is now 21101m (21.1km). Which can be included in the *dens_est* function as above.

Function:produce LTDS final report - *LTDS_summary*

Now we can produce our final report. There is slightly more manual input for the report, since we wanted to make this adaptable to changes in data and results which cannot be captured by the LTDSMethod package.

The parameters of the function and report are as follows:

To move forward, we need to read in our other example dataset. During the surveys, observers have the opportunity to record an other observations made. Specifically, any observations of commensal species, or discoveries of gopher tortoise carcasses.

```
obs.data <- read_csv("./Other_observation_form_v2.csv")
#> Rows: 10 Columns: 4
#> -- Column specification --------------------------------------------------------
#> Delimiter: ","
#> chr (2): Species, Status
#> dbl (2): Easting, Northing
#>
#> i Use `spec()` to retrieve the full column specification for this data.
#> i Specify the column types or set `show_col_types = FALSE` to quiet this message.
#Summarize the data
head(obs.data)
#> # A tibble: 6 x 4
#>   Species         Easting Northing Status
#>   <chr>             <dbl>    <dbl> <chr>
#> 1 Gopher tortoise  300521  3350451 Moderate decay
#> 2 Indigo snake     300866  3351245 Alive
#> 3 Florida mouse    300126  3350262 Alive
#> 4 Florida mouse    300126  3350262 Alive
#> 5 Florida mouse    300126  3350262 Alive
#> 6 Gopher tortoise  301019  3351150 Carapace only
#effort: the effort from the full LTDS survey. We will use the pilot effort that
#we calculated earlier for this example.
effort <- as.numeric(effort.2)
#nburr: the total number of burrows found during the survey. We can axtract this
#from our example dataset, which will likely be applicable for future use.
nburr <- as.numeric(length(full.data$Burrow_width))
#burr.w.mn: the mean width of detected burrows.
burr.w.mn <- as.numeric(round(mean(full.data$Burrow_width), digits = 2))
#burr.w.min: the minimum width of detected burrows.
burr.w.min <- as.numeric(round(min(full.data$Burrow_width), digits = 2))
#burr.w.max: the maximum width of detected burrows.
burr.w.max <- as.numeric(round(max(full.data$Burrow_width), digits = 2))
#comm.comm: the name of the most common commensal species detected. We can
#extract the number of detections for the full dataset by:
table(obs.data$Species)
#>
#>       Burrowing owl Florida gopher frog       Florida mouse     Gopher tortoise
#>                   1                   2                   4                   2
#>        Indigo snake
#>                   1
#This shows that the most common species was the Florida mouse
comm.comm <- "Florida mouse"
#occ: the number of occupied burrows discovered.
#unocc: the number of unoccupied burrows discovered.
#unk: the number of unknown burrows discovered.
#We can use the same method as above to pull these numbers out of the data
table(full.data$Occupied)
#>
```

```
#>      No Unknown     Yes
#>     131      25     107
occ <- 107
unocc <- 131
unk <- 25
#site.name: the name of the recipient site. We can set this to whatever we want.
site.name <- "Tortoise Wonderland"
#site.size: total area of site, or sampling frame (acres)
site.size <- 400
#tort.dens: the estimated tortoise density at the site. We calculated this earlier.
tort.dens.new <- round(tort.dens$density, digits = 2)
tort.dens.lo <- round(tort.dens$density.lo, digits = 2)
tort.dens.hi <- round(tort.dens$density.hi, digits = 2)
#tort.pop: the estimated tortoise population at the site. We calculated this earlier.
tort.pop.new <- round(tort.pop$density)
tort.pop.lo <- round(tort.pop$density.lo)
tort.pop.hi <- round(tort.pop$density.hi)
```

The above information is used for the text part of the documents, however, any figures produced in the report will pull data directly from the working directory.

We can now input all of these into our function. As a warning, if you have previously run this function, and viewed the resulting pdf, the function will not run if you still have the pdf open.

```
LTDS_summary(effort = effort, nburr = nburr, burr.w.mn = burr.w.mn,
             burr.w.min = burr.w.min, burr.w.max = burr.w.max, comm.comm = comm.comm,
             occ = occ, unocc = unocc, unk = unk, site.name = site.name,
             site.size = site.size,
             tort.dens = tort.dens.new, tort.dens.lo = tort.dens.lo,
             tort.dens.hi = tort.dens.hi, tort.pop = tort.pop.new,
             tort.pop.lo = tort.pop.lo, tort.pop.hi = tort.pop.hi)
```

End of vignette.

Final LTDS Report

2023-03-13

LTDS Summary - Tortoise Wonderland

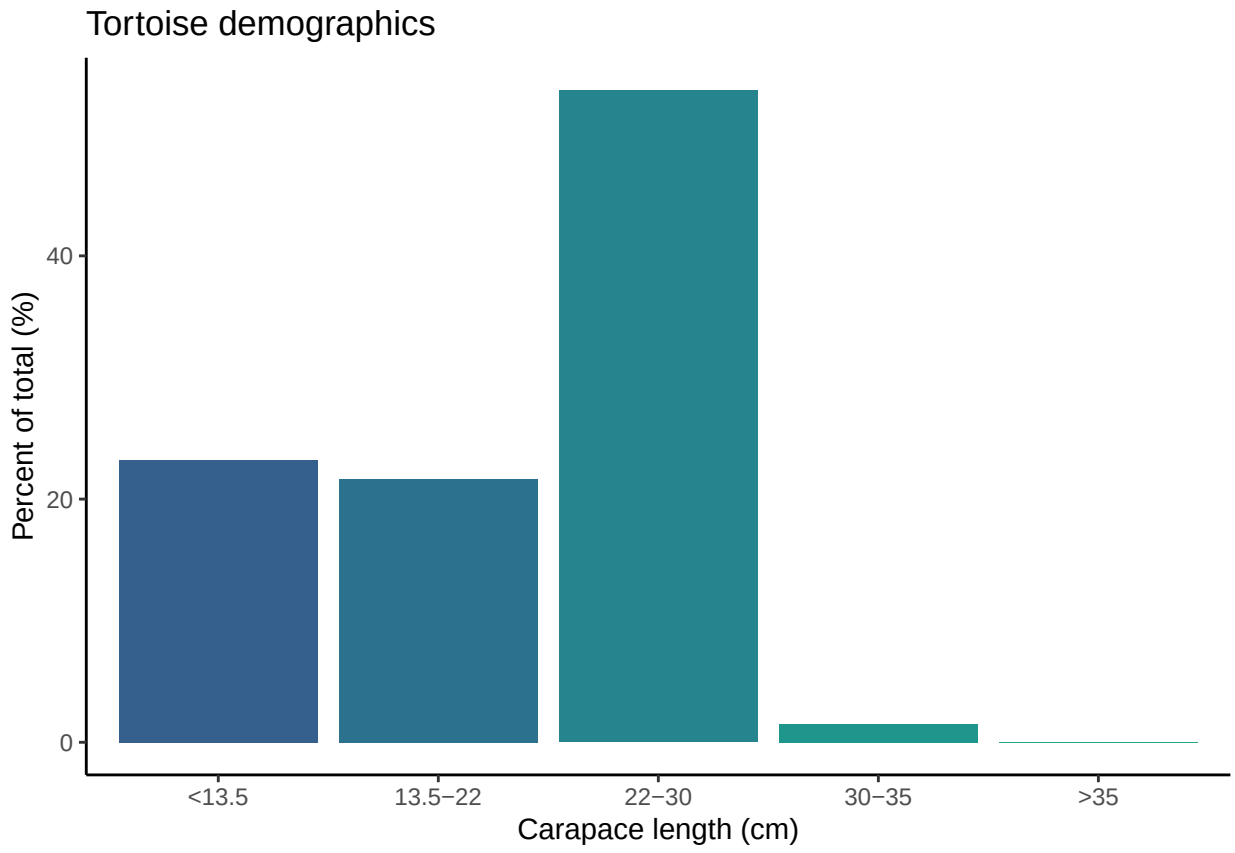
Test report for LTDSMethod package.

Overview

We walked a total of **21299.14** m during our full LTDS survey, where we found a total of **263** burrows. We calculated a density of **1.57** tortoises/acre (range = **1.28** - **1.92** tortoises/acre), resulting in approximately **629** tortoises (range = **514** - **767**) at the recipient site. We calculated a mean burrow width of **210.02** mm (range = **50.5** - **307.94**).

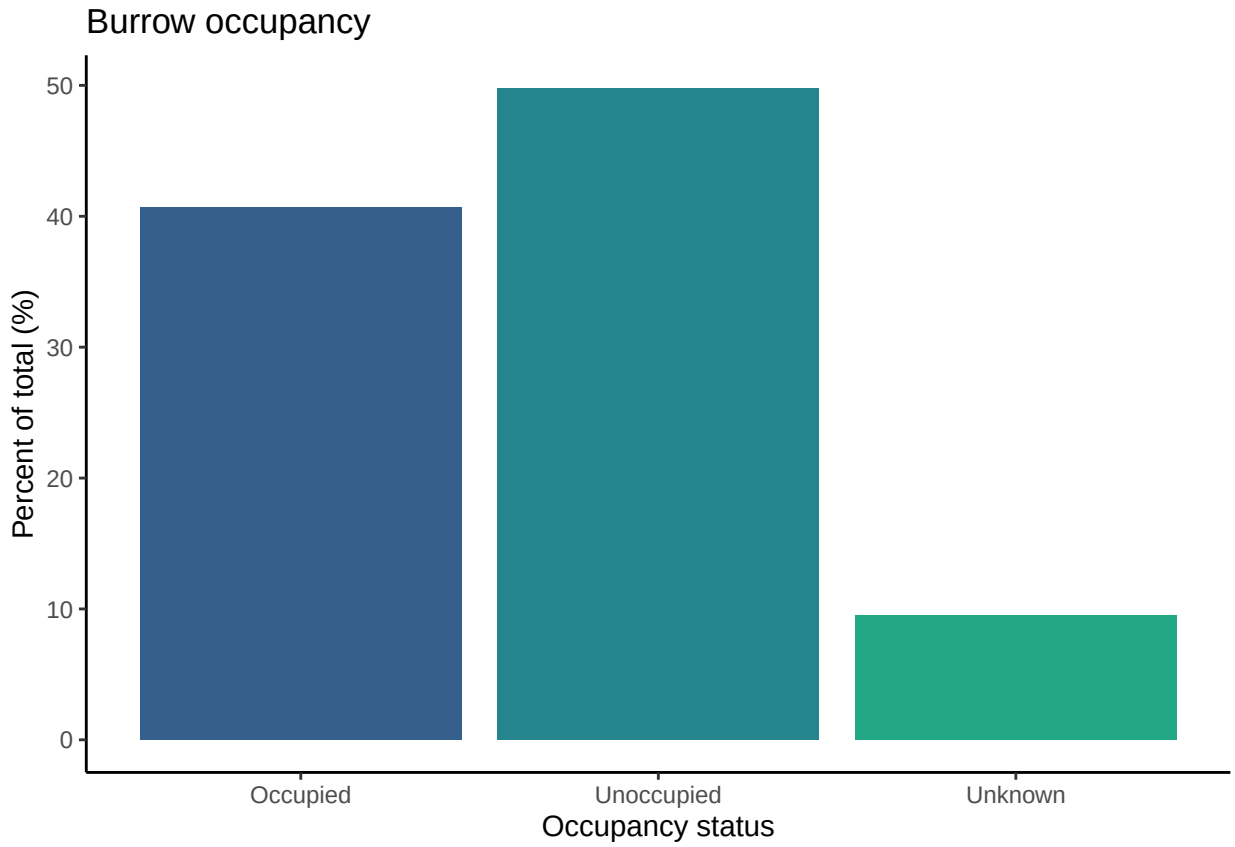
Population and density estimates

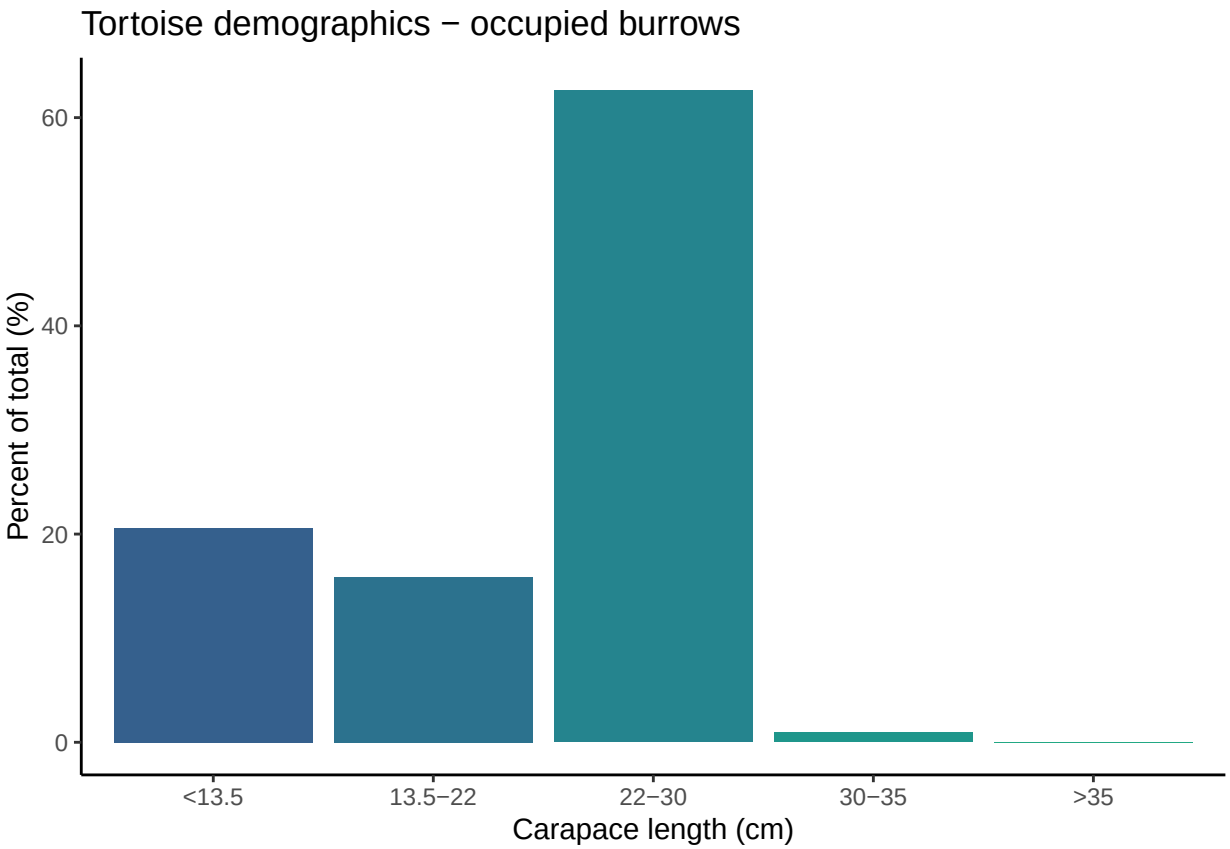
N.occ	Effort (m)	D	D UCI	D LCI	CV (%)	N	N UCI	N LCI
107	21299.14	1.57	1.28	1.92	16.74	629	514	767



Burrow scoping results

Total burrows	Occupied burrows	Unoccupied burrows	Unknown burrows	Burrow occupany (%)
263	107	131	25	40.7

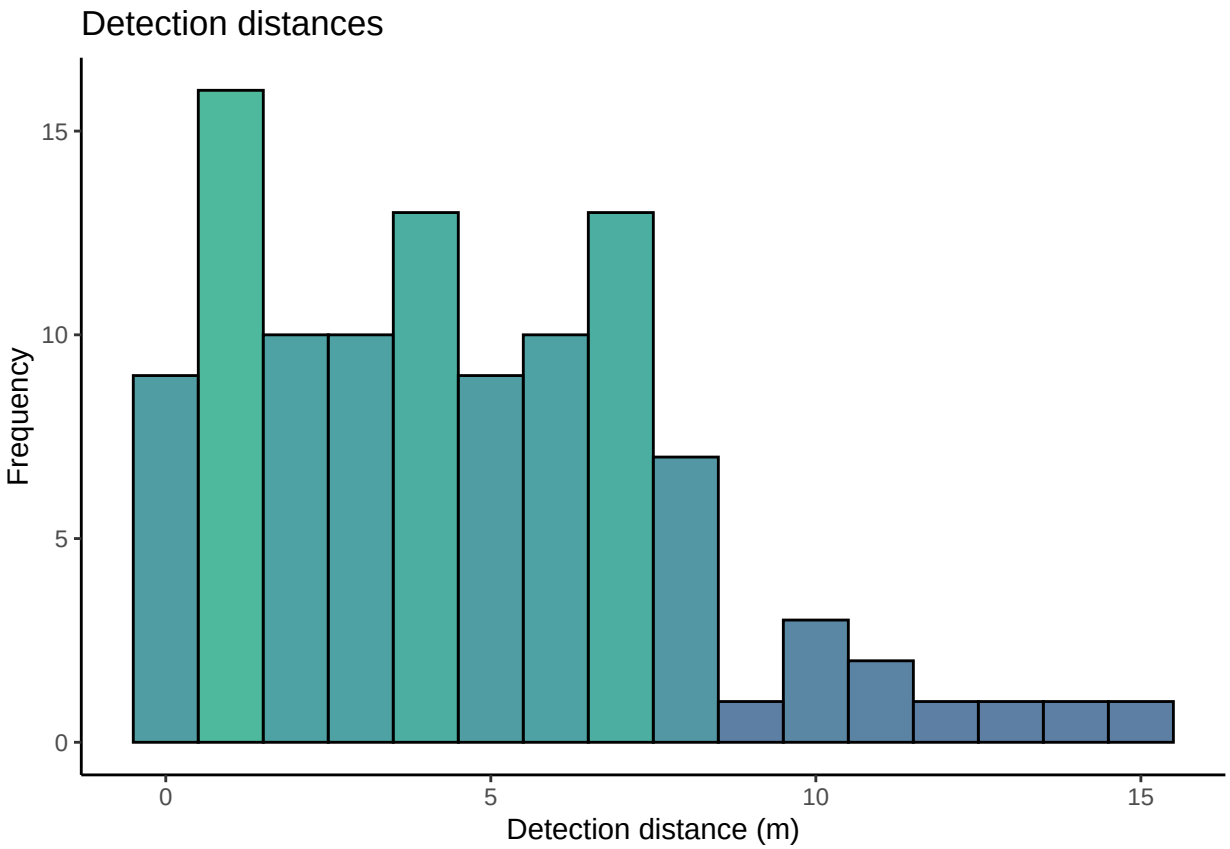


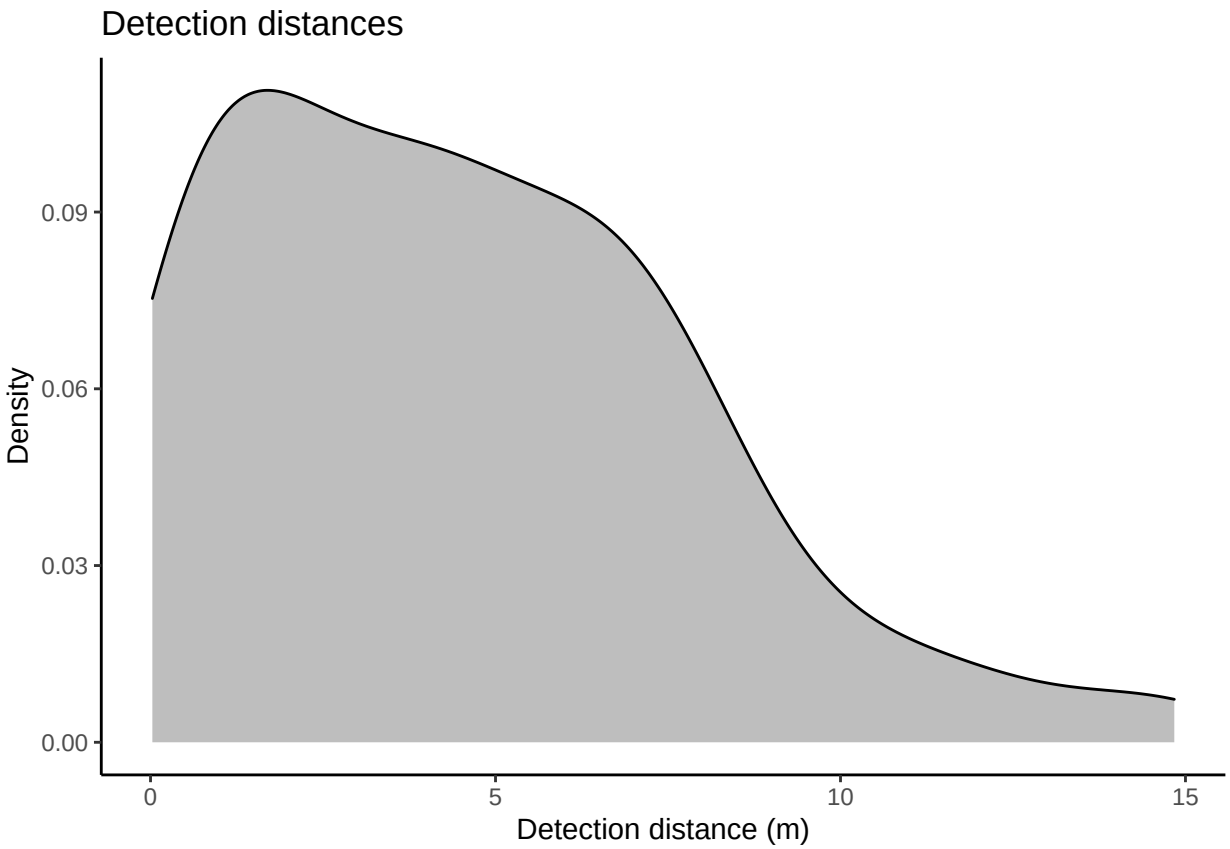


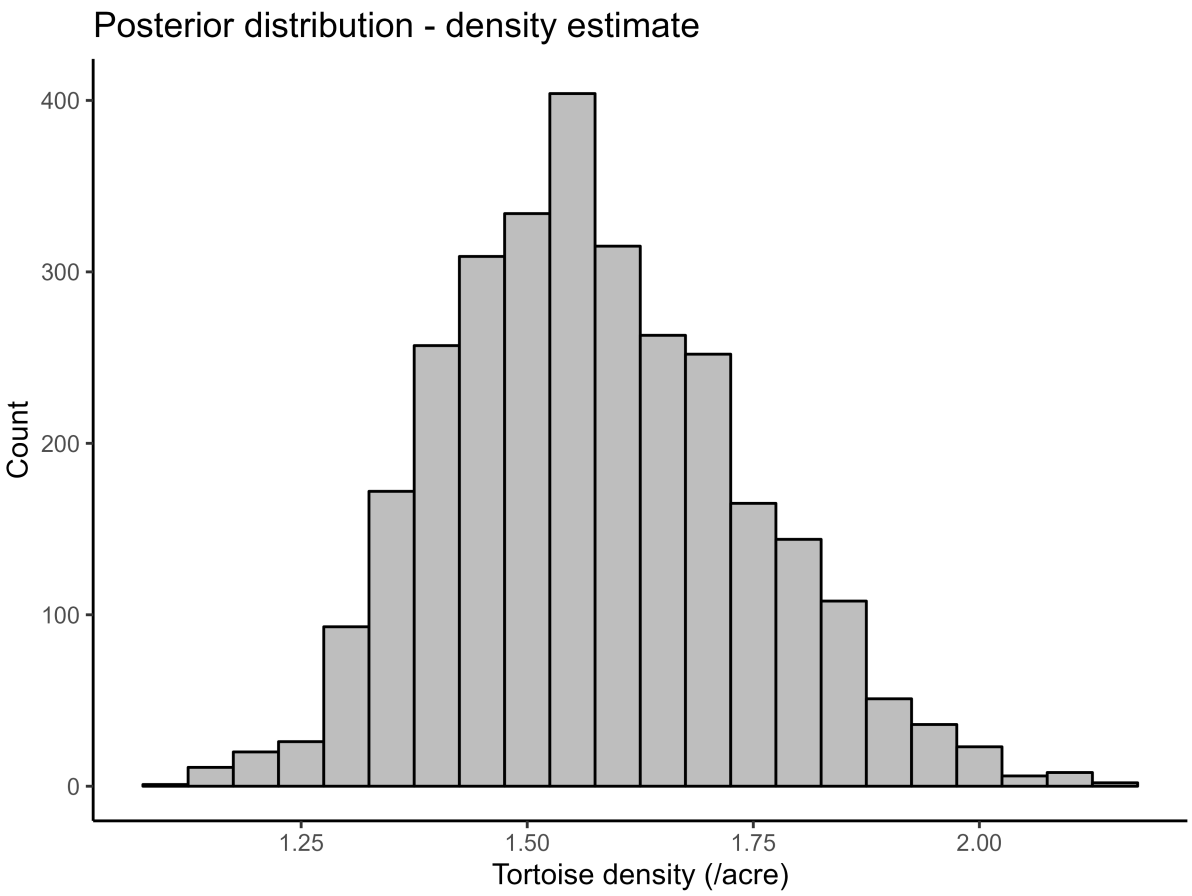
Widths of occupied burrows

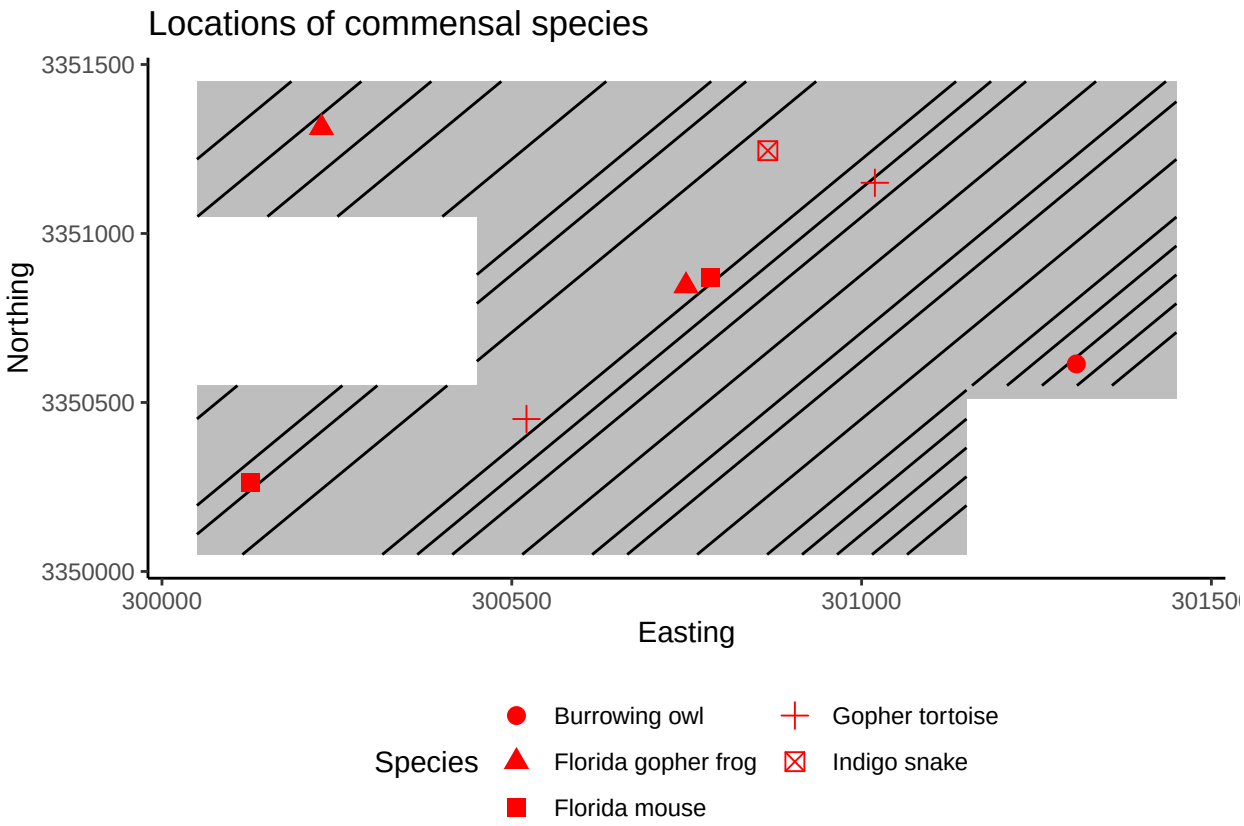
Mean burrow width (mm)	Min burrow width (mm)	Max burrow width (mm)
219.76	50.5	304.67


Locations of occupied burrows
3351500
3351000
3350500
3350000
Northing
300000
300500
301000
Easting


Detection distances
Frequency
15
10
5
0
0
5
10
15
Detection distance (m)


Detection distances
Density
0.09
0.06
0.03
0.00
0
5
10
15
Detection distance (m)

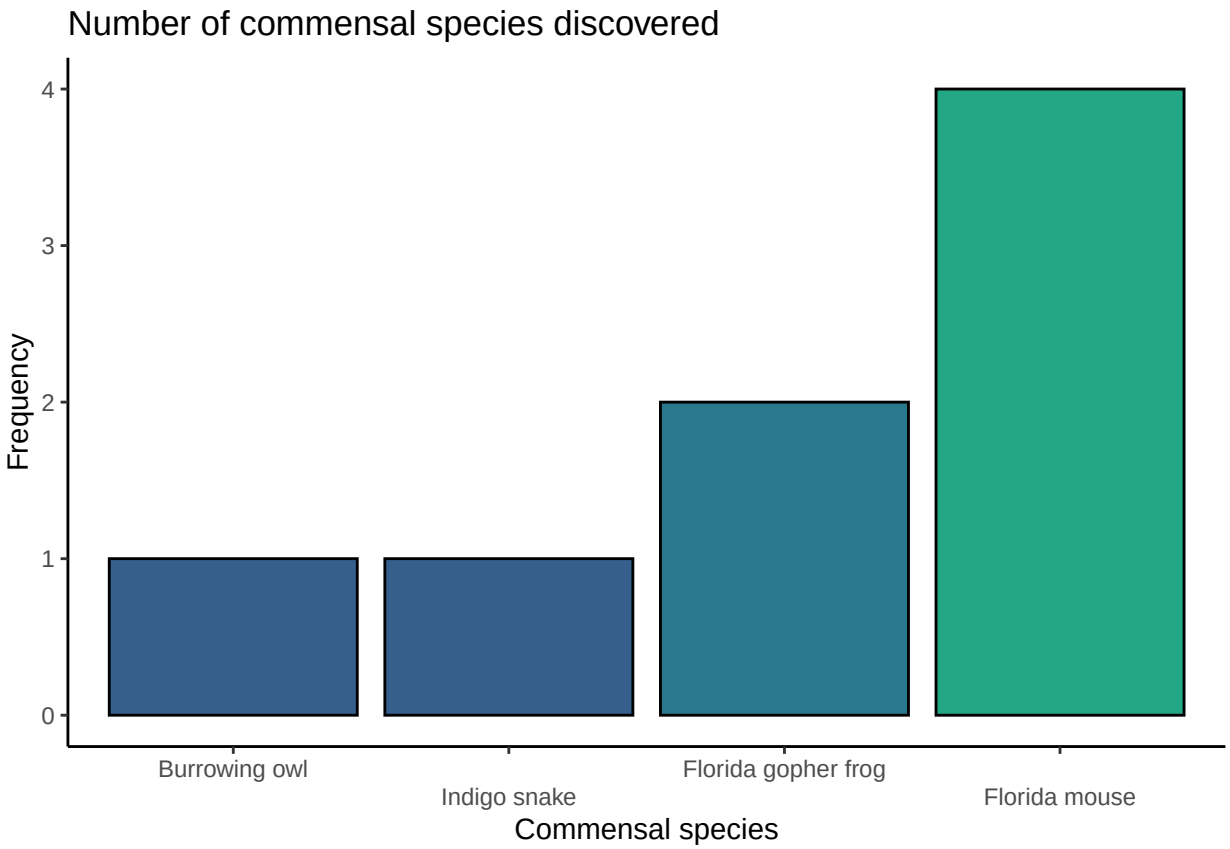

Posterior distribution - density estimate
Count
400
300
200
100
0
1.25
1.50
1.75
2.00
Tortoise density (/acre)



The most common commensal species discovered in burrows during scoping was the **Florida mouse**. A breakdown of the commensal species detection can be seen in the following table and figure:

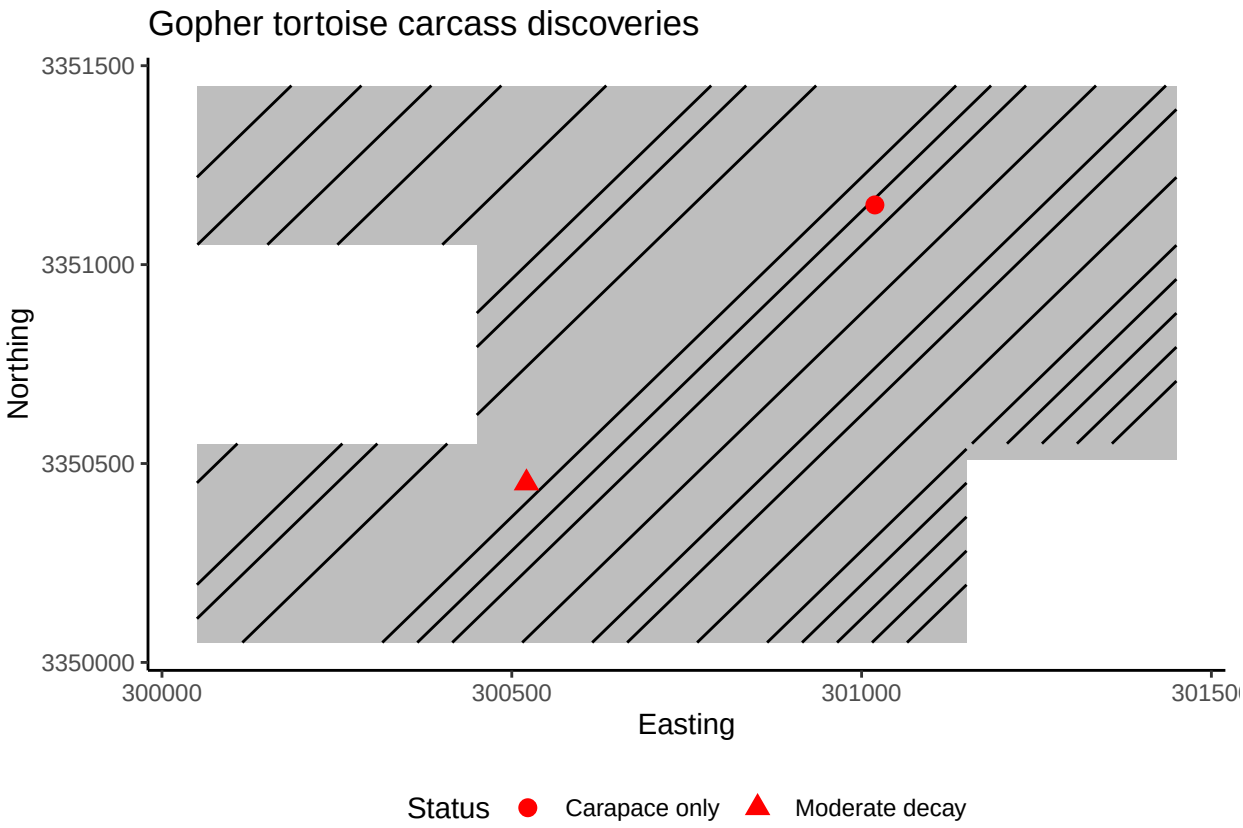
Commensal species discovered

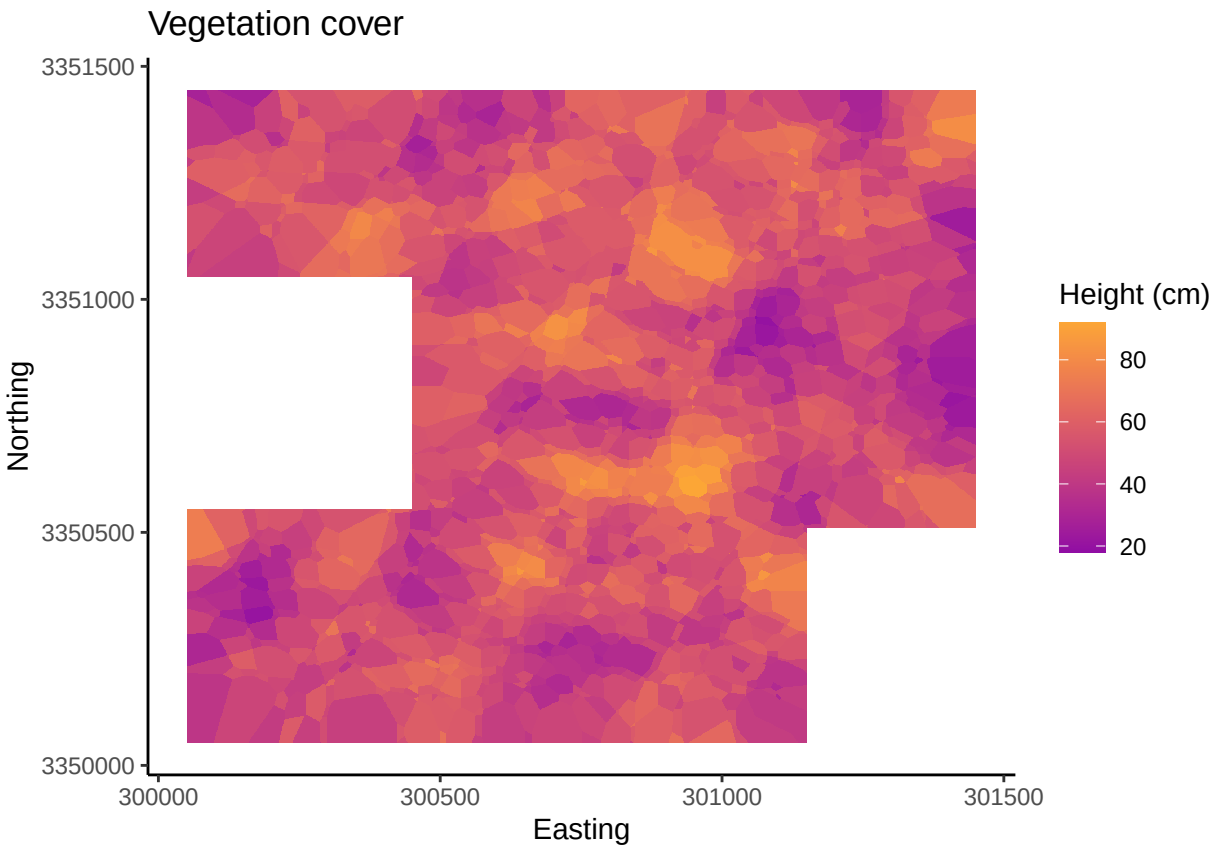
Species	No.found
Indigo snake	1
Burrowing owl	1
Florida gopher frog	2
Florida mouse	4



Gopher tortoise mortalities

Species	Easting	Northing	Status
Gopher tortoise	300521	3350451	Moderate decay
Gopher tortoise	301019	3351150	Carapace only


Gopher tortoise carcass discoveries
3351500
3351000
3350500
3350000
Northing
300000
300500
301000
Easting
Status
Carapace only
Moderate decay



End of report.