BIOM4031 Tutorial 5: Raster-Vector Operations

Figure 1: Slash-and-burn agriculture is a major driver of forest loss in Madagascar (photo credit: MongaBay)

Introduction

In Weeks 1 and 2, you learned how to interact with, plot and manipulate spatial vector and raster data in R. In Week 3, we will introduce a range of common operations that combine vector and raster data, building on case studies used in previous tutorials and practicals. This tutorial will focus on how to create or modify raster data using vectors (masking and cropping, rasterization, distance grids) while tutorial 6 will introduce operations that use rasters to create and modify vectors.

Today's tutorial builds on a case study introduced in Practical 1 that investigated threats posed by anthropogenic fire to tropical moist forest in Madagascar's Ranomafana National Park. In the Practical, forest cover data had been polygonized to allow us to demonstrate vector-vector operations. However, this is a very memory hungry way of storing landcover data and it would not be possible to perform large scale analyses of fire risk (e.g. for the whole of Madagascar) in this way. In the original publications from which this data was sourced, forest cover was represented and analysed in raster form. In this tutorial we will repeat some of the analyses from Practical 1 and demonstrate how to achieve the same outcomes (mapping fire distribution and measuring areas at risk from fire) using vector-raster operations.

Learning objectives

By the end of the tutorial you will know how to:

1. Crop and mask a raster to a vector layer

2. Rasterize point and polygon data
3. Generate distance grids (i.e. raster surfaces of distance to vector features)

Data

In the Tutorial 5 data folder you will find the following datasets:

1. *ranomafana.gpkg* A shapefile of the Ranomafana National Park downloaded from the World Database of Protected Areas https://www.protectedplanet.net
2. *fires.gpkg* A geopackage containing the locations of fires detected by the VIIRS satellite in the Ranomafana National Park area in 2017, downloaded from https://firms.modaps.eosdis.nasa.gov/download/ (used in Practical 1).
3. *forest2017.tif* A raster of forest cover in Madagscar in 2017 downloaded from the Supplementary Material of Vieilledent et al. 2018
4. *fire_distance.tif* A distance raster of distance from the nearest fire in the Ranomafana NP area. This raster is generated in the code below but can take some time to run, so you can bypass that step and read this raster in directly if your computer has memory issues.

Load packages

This Tutorial uses our standard set of vector and raster processing and plotting packages introduced in our previous Tutorials.

```
library(sf)
library(tidyverse)
library(terra)
library(tidyterra)
library(colorspace)
```

1. Introducing `SpatVectors`

Before we start working on raster-vector data operations we need to introduce an additional object class that we have not used before. One of the quirks of R is that packages are contributed by different teams of developers who aren't always working together to a single, interoperable standard. So far we have used `sf` for working with spatial vectors as it is the most feature-rich and well supported vector processing package. However, at the time of writing, `terra` has limited support for `sf` objects. Instead it has its own spatial vector format called a `SpatVector` that we need to use for operations that involve interacting vector and `SpatRaster` datasets. Fortunately, it is extremely easy to convert between the two formats. We can demonstrate by reading in our 'ranomafana' NP shapefile:

```
rnp = st_read('data/ranomafana.gpkg')
```

We convert it to a `SpatVector` using the `vect` function:

```
rnp = vect(rnp)
```

And back to `sf` using the `st_as_sf` function we introduced previously

```
rnp = st_as_sf(rnp)
```

We can also use `vect` to read vector files directly into `SpatVector` form:

```
rnp = vect('data/ranomafana.gpkg')
```

It is very common to convert back and forth between these two vector formats depending on which operation you want to perform (e.g. to `sf` for geoprocessing and to `SpatVector` for interactions with raster data). For example, we could transform our national park boundary to projected coordinates (EPSG:32738) using `sf` then convert back to a `SpatVector` for interacting with raster data.

```
rnp = st_as_sf(rnp) %>%
      st_transform(32738) %>%
      vect()
```

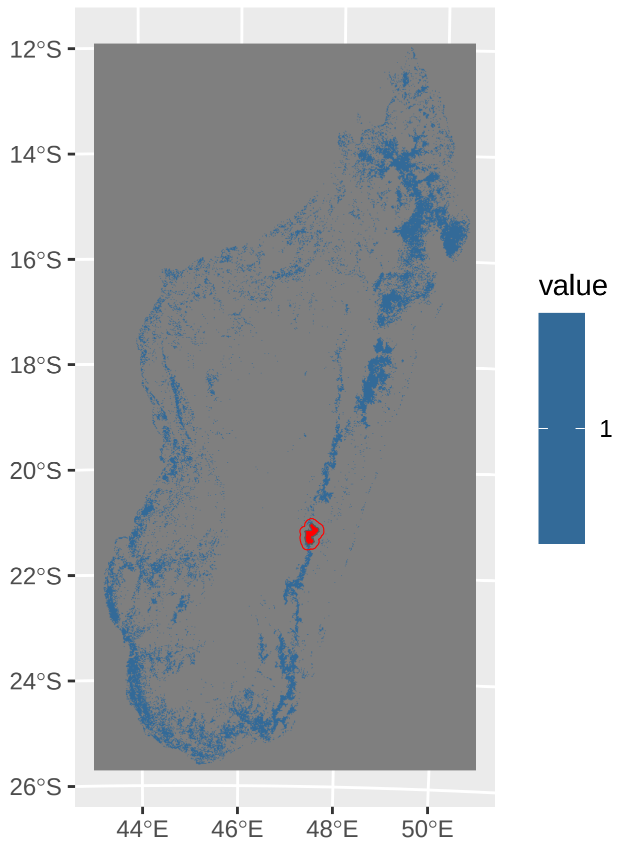
We could also convert back to sf for generating our 10 km buffer zone around the national park using `st_buffer`, although `terra` also has its own `buffer` function for buffering `SpatVectors`

```
#buffer = st_as_sf(rnp) %>% st_buffer(10000) %>% vect()
buffer = buffer(rnp,10000)
```

2. Masking and cropping

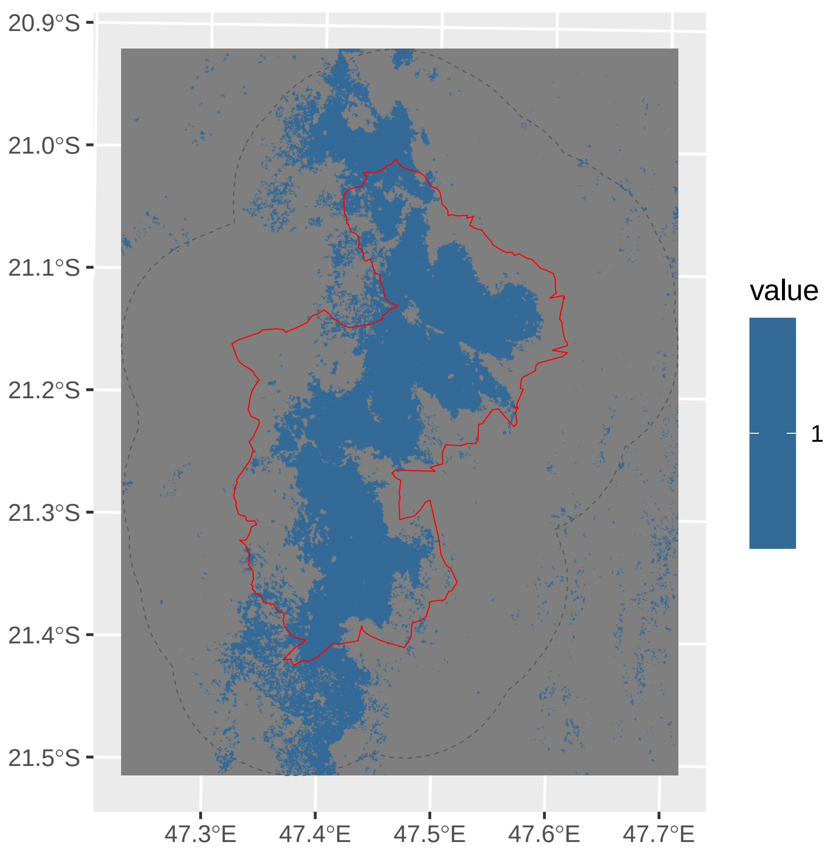
Now we have our national park and buffer zone in the correct format, we can start working with some raster data. Let's start by reading in the complete forest cover dataset for Madagascar in 2017 that our study was derived from. We'll plot it with our RNP boundary and buffer zone `SpatVector` layers overlaid, which we can plot using `geom_spatvector`:

```
forest = rast('data/forest2017.tif')

ggplot() +
  geom_spatraster(data = forest) +
  geom_spatvector(data = rnp, fill = 'red') +
  geom_spatvector(data = buffer, fill = 'transparent',colour = 'red')
```



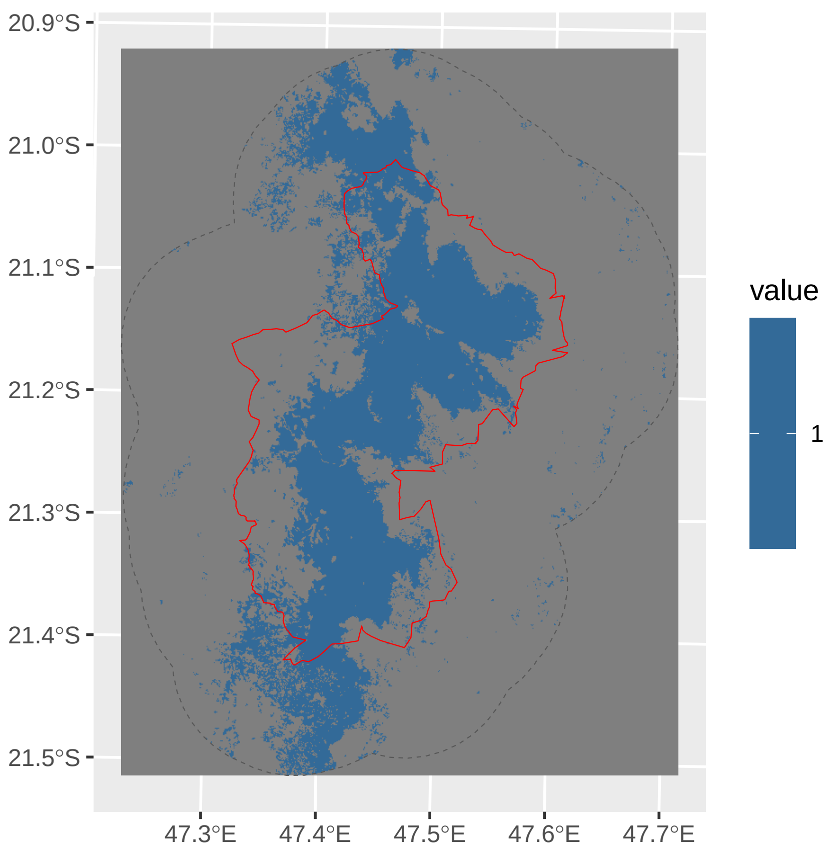
This raster dataset is very large and too big to work with for our example. So, the next we want to clip our raster to retain just the forest cover data we are interested in within RNP buffer. This involves the use of a pair of `terra` functions called `crop` and `mask`. As with `st_crop` in the `sf` package, `crop` clips a raster to the bounding box surrounding another vector or raster layer:

```
forest = crop(forest, buffer)

ggplot() +
  geom_spatraster(data = forest) +
  geom_spatvector(data = rnp, fill = 'transparent', colour = 'red') +
  geom_spatvector(data = buffer, fill = 'transparent',linetype='dashed')
```



To remove the bits of forest outside the buffer we then `mask` the raster.

```
forest = mask(forest, buffer)

ggplot() +
  geom_spatraster(data = forest) +
  geom_spatvector(data = rnp, fill = 'transparent', colour = 'red') +
  geom_spatvector(data = buffer, fill = 'transparent',linetype='dashed')
```



Essentially `mask` sets any values that lie outside of the masking polygon to `NA`.This is why the sequential use of `crop` and `mask` is important. If we had just masked our original forest raster before cropping we would have ended up with this (and it would have taken ages to run):

(resources/mask_only.jpg){width=70%}

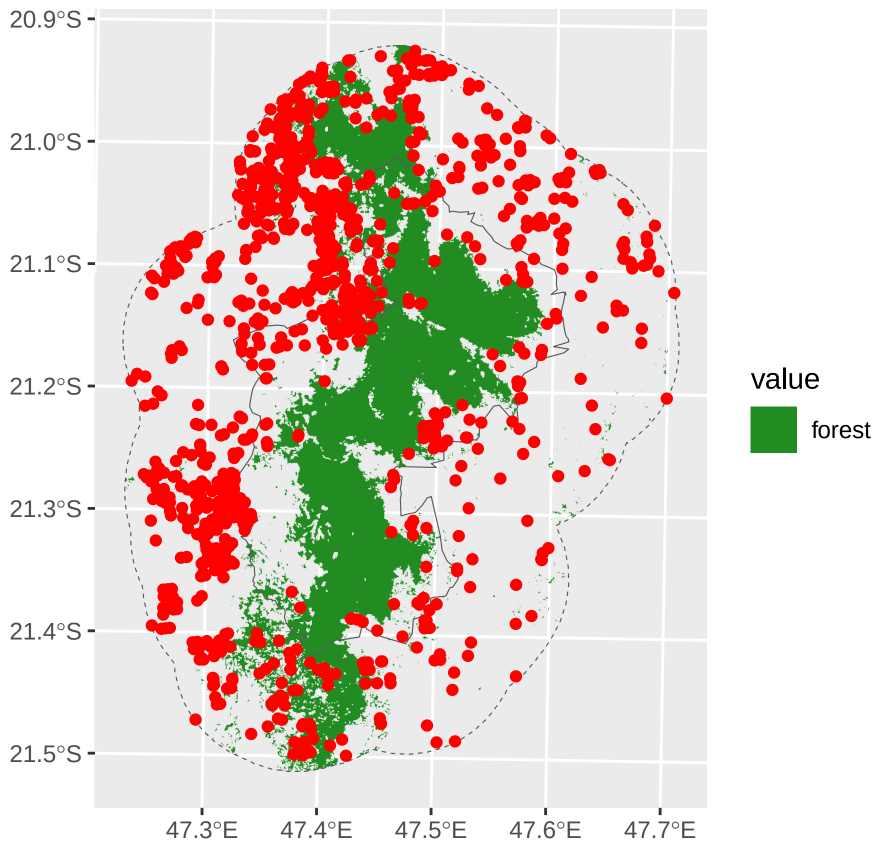
Now that we've trimmed our raster we can style our plot a bit. We have only a single cell value in our `forest` raster, so we might as well make it a categorical raster by setting the levels, and then style it using a discrete fill scale.

```
levels(forest) <- data.frame(id=1, name='forest')

rnp.plot =
ggplot() +
  geom_spatraster(data = forest) +
  geom_spatvector(data = rnp, fill = 'transparent',size=2) +
  geom_spatvector(data = buffer, fill = 'transparent',linetype='dashed') +
  scale_fill_manual(values = 'forestgreen', na.translate=F)
```

3. Rasterization

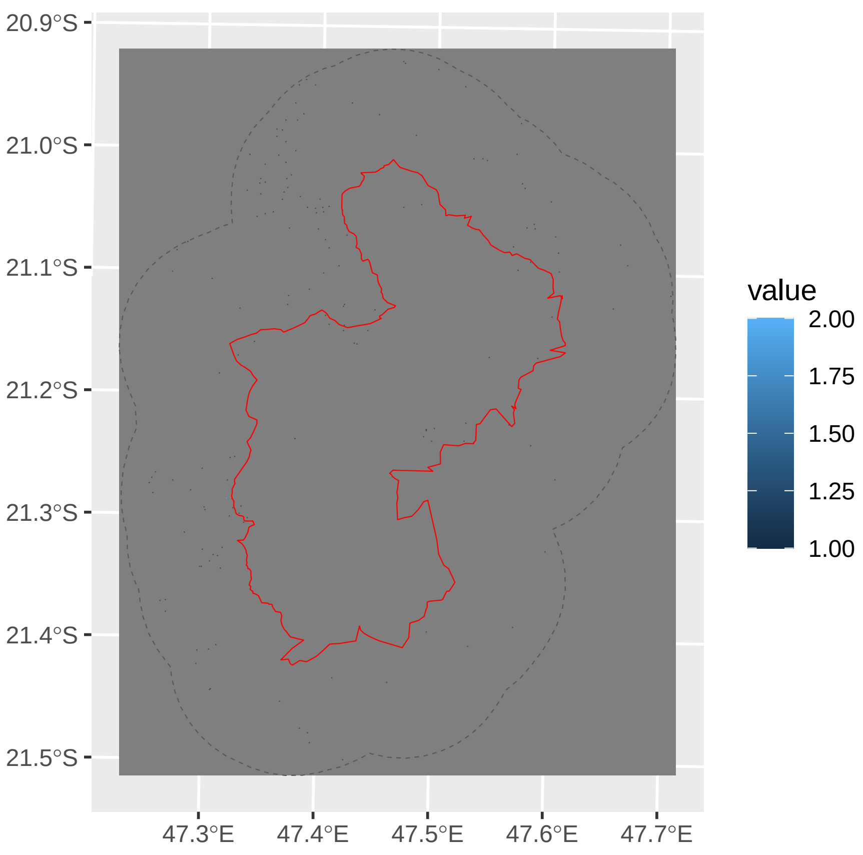
Now that we've masked and cropped our forest data, let's read in our VIIRS fires dataset again. This time we'll read it in directly as a `SpatVector`:

```r
fires = vect('data/fires.gpkg')

rnp.plot + geom_spatvector(data = fires, colour = 'red')
```



As in Practical 1, We may want to summarize this data on a grid to better visualise the distribution of fires. However, this time, rather than counting points in vector (polygon) grid cells, we will turn our fire point data into a raster layer. This is called 'rasterization' and is performed using the `rasterize` function. The premise is exactly the same as intersecting points with polygons, but this time the points are intersected with the grid cell of the raster that they overlap or fall inside. The first two arguments of the `rasterize` function are the vector layer we want to rasterize and and a raster layer that is used as a template (in this case we will use our `forest` raster). We then need to specify a function to be used for rasterizing the points. To count the number of points that intersect each cell we can use the `count` function:

```r
fire.grid = rasterize(fires, forest, fun = 'count')

ggplot() +
  geom_spatraster(data = fire.grid) +
  geom_spatvector(data = rnp, fill = 'transparent', colour = 'red') +
  geom_spatvector(data = buffer, fill = 'transparent',linetype='dashed')
```

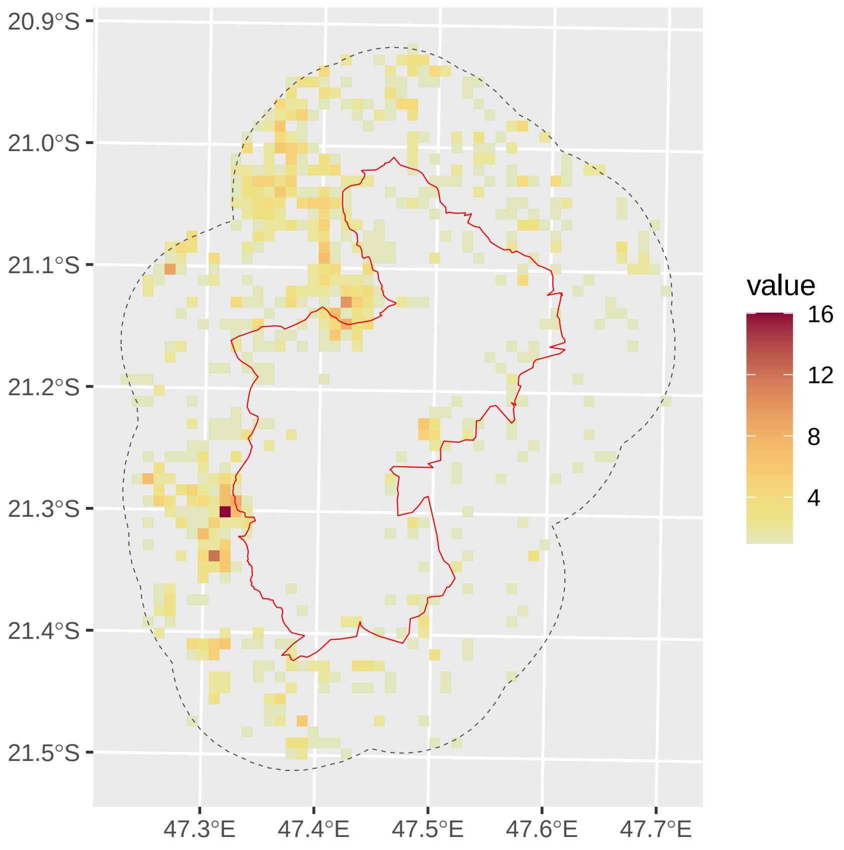


This doesn't look like much because the cell sizes in our template raster are very small (30 x 30 m); so we have counted fires per 30 x 30 m cell, which has a maximum of 2. If we want to use a different template raster with larger cells (e.g. 1 km x 1 km) we can make a new blank raster using `rast`. To do that, we give `rast` an object to take its spatial extent from (e.g. our `buffer` polygon) and then specify the resolution we want:

```
template = rast(buffer, resolution = c(1000,1000))
```

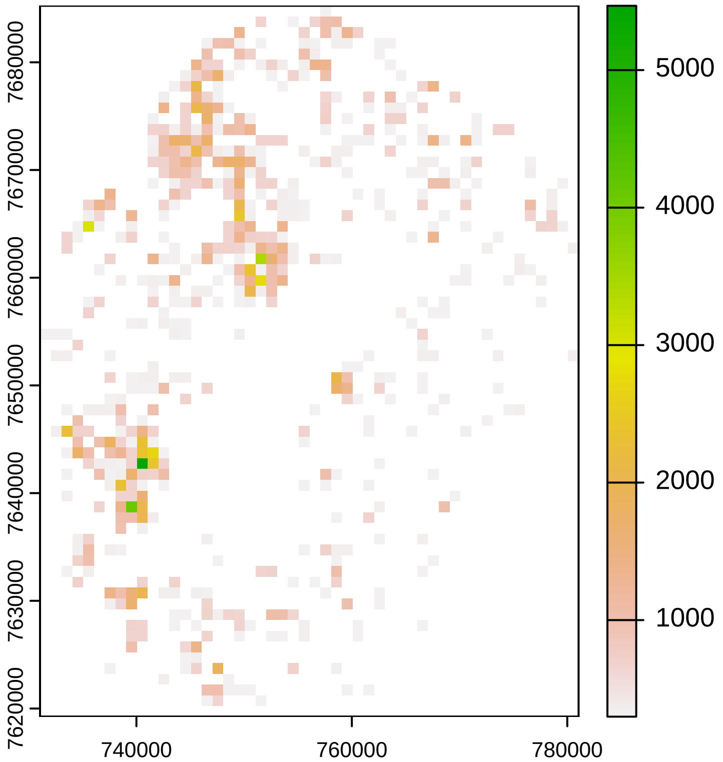
Then we rasterize our fire points onto the new template raster:

```
fire.grid = rasterize(fires, template, fun = 'count')

ggplot() +
  geom_spatraster(data = fire.grid) +
  geom_spatvector(data = rnp, fill = 'transparent', colour = 'red') +
  geom_spatvector(data = buffer, fill = 'transparent',linetype='dashed') +
  scale_fill_continuous_sequential('Heat',na.value = 'transparent')
```



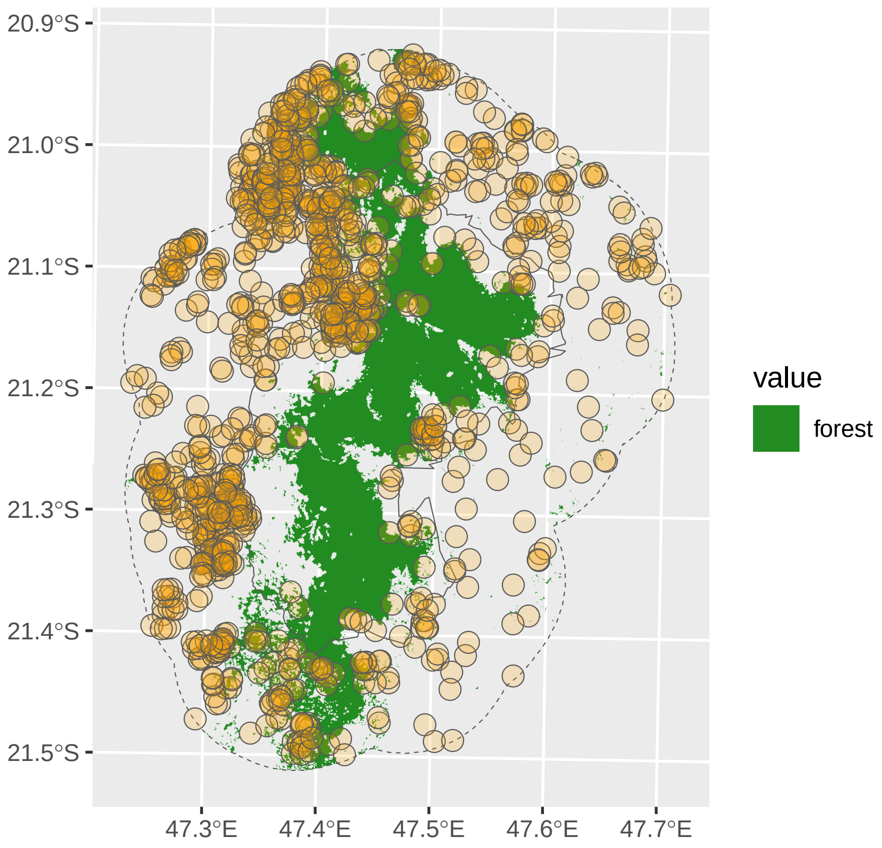
That looks a bit more sensible. You can also rasterize using a field from your data. For example, if we look at the attribute table of our `fires` dataset you will see it contains a column called 'brightness' which indicates how bright (or large) the fire was. So, rather than simply counting fires that fall inside each cell we might want to look at the cumulative brightness of fires in each cell (this could be the difference between a few small and a few large blazes, for example). We can do that by summing the brightness of points falling in each cell using rasterize:

```
rasterize(fires, template,field = 'brightness',fun = sum) %>%
plot
```



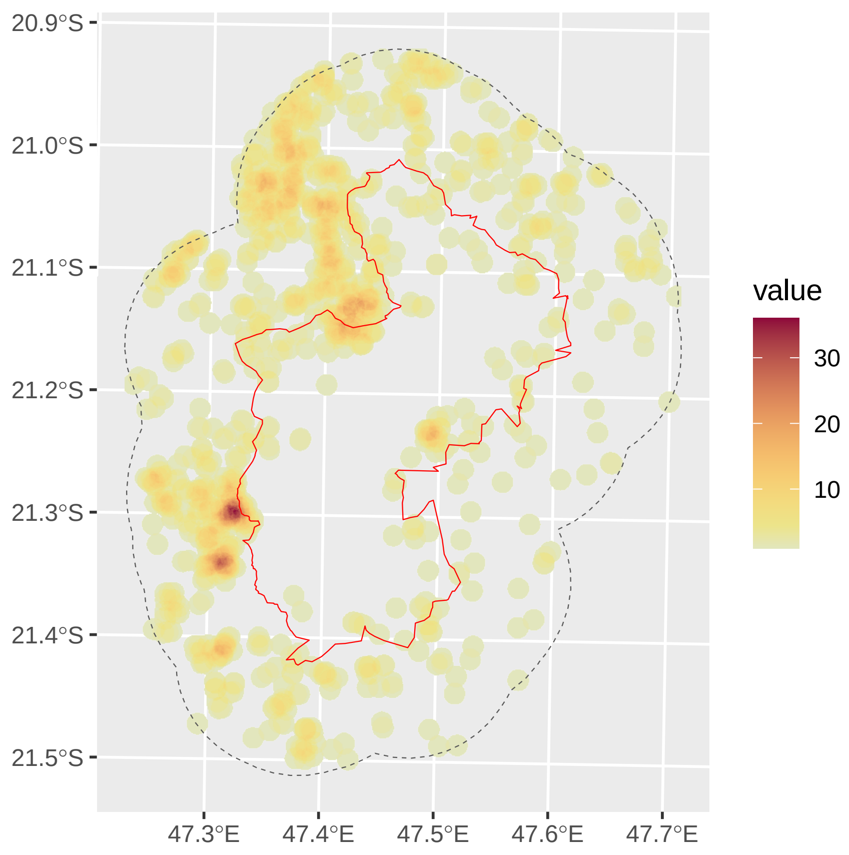
We can also rasterize polygons and lines in a similar way. To demonstrate let's map the number of fires that occur within 1 km of each cell in our `forest` raster. We start by buffering our fire points using `buffer` to create polygons

```
fire.buf = buffer(fires, 1000)

rnp.plot +
  geom_spatvector(data=fire.buf, fill = 'orange', alpha=.2)
```



Then we rasterize them using our `forest` raster as a template to count how many polygons in `fire.buf` overlap each cell.

```
fire.buf.grid = rasterize(fire.buf, forest, fun = 'count')

# We can also set any cells that have 0 fires to missing so they aren't plotted
fire.buf.grid[fire.buf.grid==0] <- NA

ggplot() +
  geom_spatraster(data = fire.buf.grid) +
  geom_spatvector(data = rnp, fill = 'transparent', colour = 'red') +
  geom_spatvector(data = buffer, fill = 'transparent',linetype='dashed') +
  scale_fill_continuous_sequential('Heat',na.value = 'transparent')
```

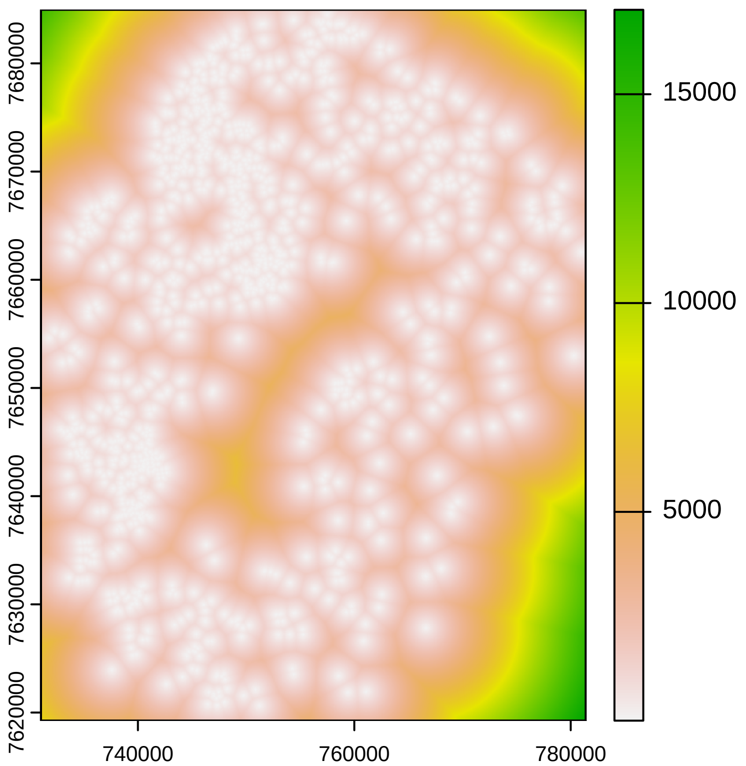


That gives us a pretty good representation of spatial variation in the density of fires.

4. Distance grids

Another common operation in which we use vectors to create raster datasets is in the creation of 'distance grids', which measure the distance from each raster cell to the nearest point on a vector feature. We'll use this operation to measure the distance from each forest filled cell to the nearest fire. Distance grids are created using the `distance` function as shown below [**Note that while `terra` is generally very quick, the `distance` function is currently an exception - so don't attempt this step if your computer has memory issues. It takes about 5 minutes to complete on my high spec machine. If you want to bypass this step you can read in the fire.distance raster directly from the Tutorial 5 data directory**].

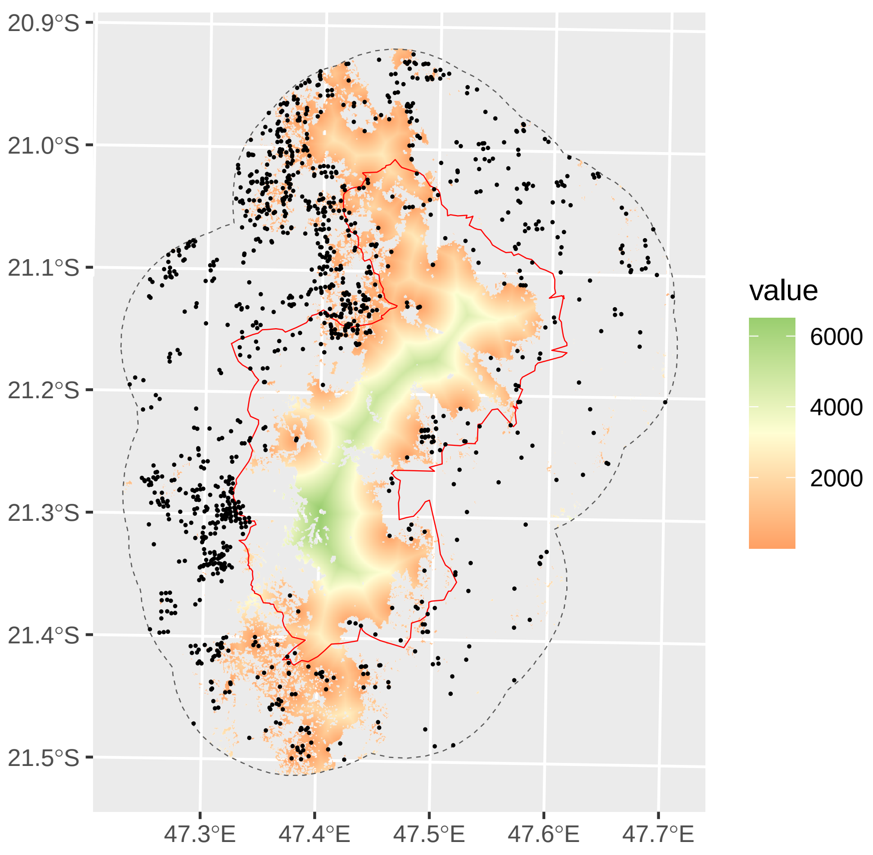
```
fire.distance = distance(forest,fires)
plot(fire.distance)
```



We can then mask this layer to just our forested area. The `mask` function that we introduced a minute ago also works for masking one raster by another. In the raster-raster case, any cells that are NA in the second (mask) raster become NA in the first:

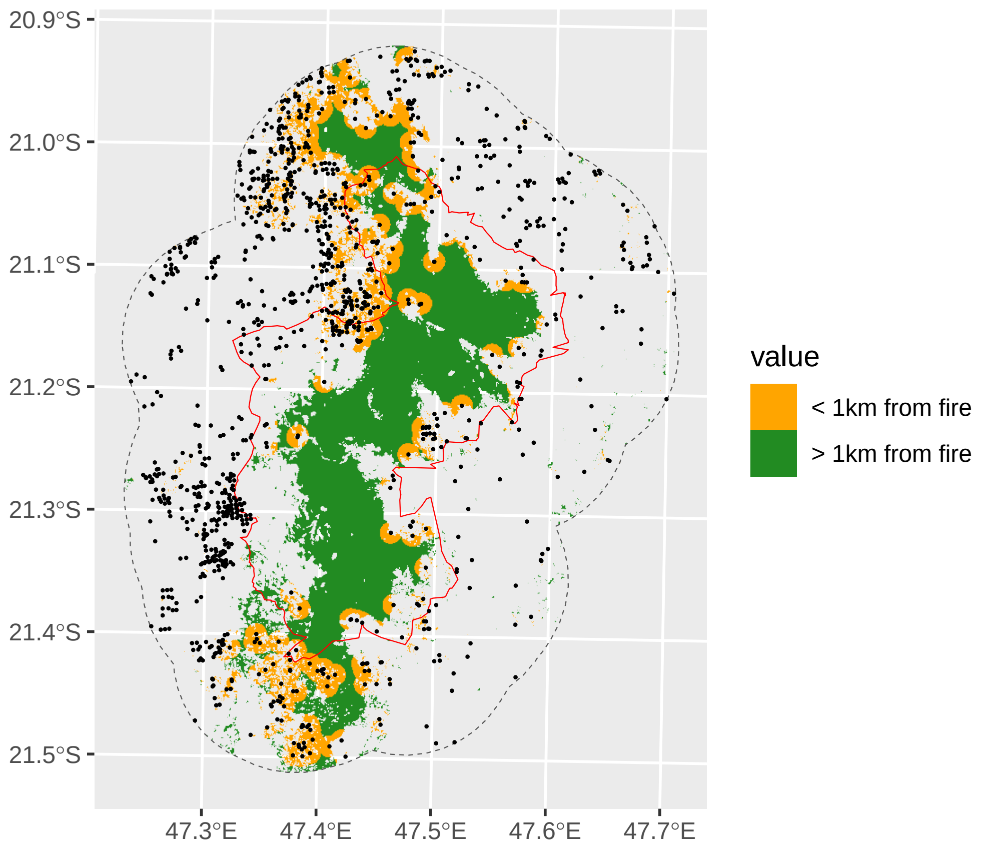
```
fire.distance = mask(fire.distance,forest)


ggplot() +
  geom_spatraster(data = fire.distance) +
  geom_spatvector(data = rnp, fill = 'transparent', colour = 'red') +
  geom_spatvector(data = buffer, fill = 'transparent',linetype='dashed') +
  geom_spatvector(data = fires, size=.1) +
  scale_fill_whitebox_c('soft',direction = -1)
```

```
## SpatRaster resampled to ncells = 500152
```



So that gives us the distance of each forested pixel from the closest fire. We could then use some of the simple raster reclassification techniques we introduced in Tutorial 3 to highlight areas that are within 1 km of a fire. First I'll duplicate fire.distance to a new object and then relabel cells that < or > 1000m from fire. We can then make it categorical by adding an attribute table to the levels.

```r
d = fire.distance

d[d <= 1000] <-1
d[d > 1000] <-2

levels(d) <- data.frame(id = 1:2, label = c('< 1km from fire','> 1km from fire'))

ggplot() +
  geom_spatraster(data = d) +
  geom_spatvector(data = rnp, fill = 'transparent', colour = 'red') +
  geom_spatvector(data = buffer, fill = 'transparent',linetype='dashed') +
  geom_spatvector(data = fires, size=.1) +
  scale_fill_manual(values = c('orange','forestgreen'), na.translate = F)
```

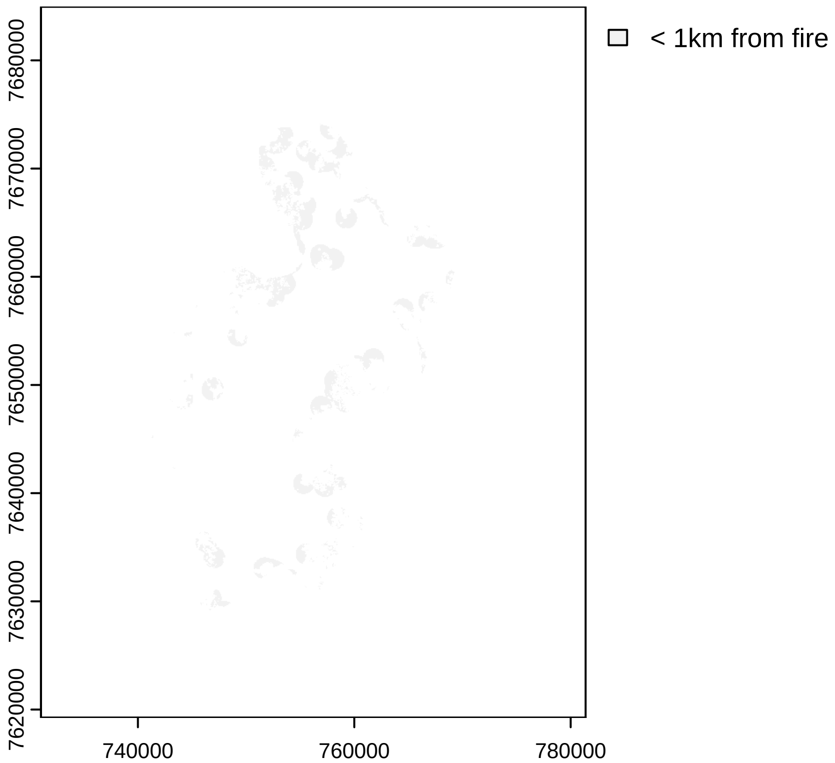


Finally, we can measure the proportion of forested area located within 1 km of a fire inside and outside the National Park. First we mask our classified distance grid, `d`, to the area inside the national park and outside the national park (using `inverse = TRUE` for the latter to keep only cells outside the masking layer):

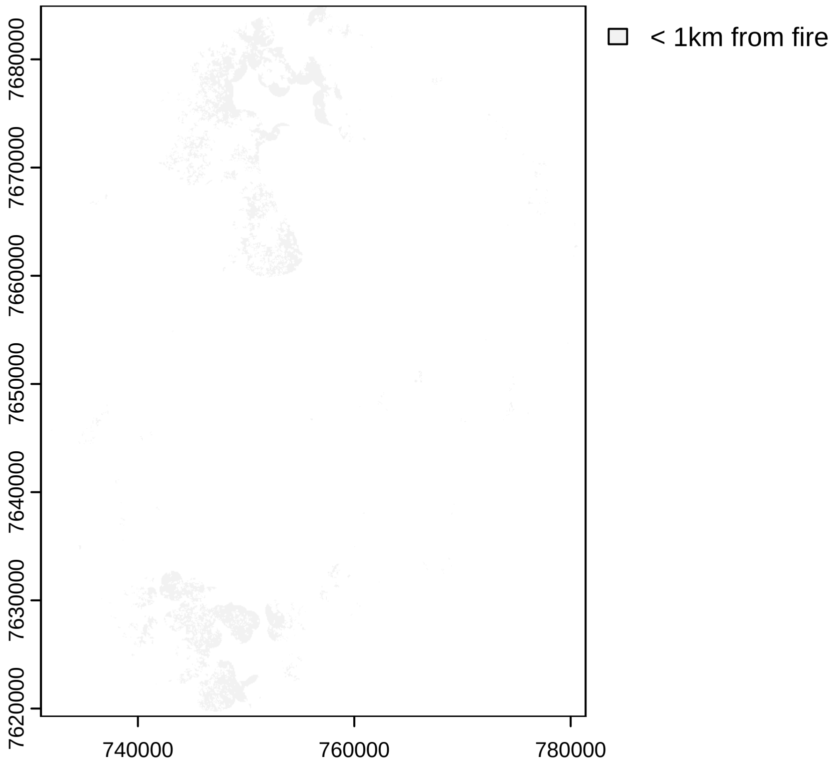
```
rnp.forest = mask(d,rnp)
buf.forest = mask(d,rnp,inverse=T)

plot(buf.forest)
plot(rnp.forest)
```

Then we make an 'at risk' layer for each zone by duplicating our forest layers and keeping only cells that are $<$ 1km from a fire (setting any cells $>$ 1 km, which had value 2 in our classied raster, to NA)

```
rnp.risk = rnp.forest
rnp.risk[rnp.risk==2]<-NA
plot(rnp.risk)
```



```
buf.risk = buf.forest
buf.risk[buf.risk==2]<-NA
plot(buf.risk)
```



And finally we can measure the area of non-NA pixels in our risk and total forest layers to calculate the propotion of forested area that is 'at risk' (i.e. < 1km from a fire) inside and outside the Ranomafana NP.

```
expanse(buf.risk,unit='km') / expanse(buf.forest,unit='km')
expanse(rnp.risk,unit='km') / expanse(rnp.forest,unit='km')
```

This tells us that 63% of remaining forest in the buffer area is at risk from fire (i.e. < 1km) compared to 17% of forest in the National Park.