Combining Raster and Vector Data

HES 505 Fall 2022: Session 13

Matt Williamson

Today's Plan

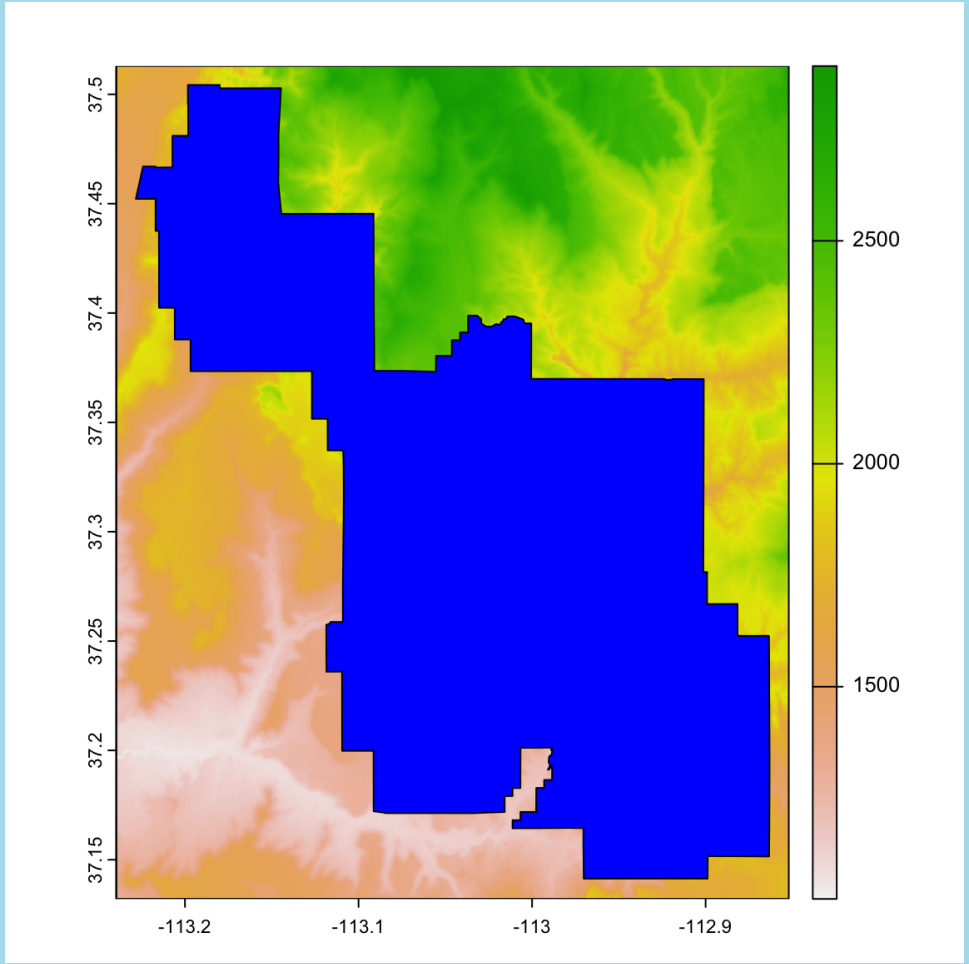
Objectives

- By the end of today, you should be able to:
  - Clip, crop, or extend vector and raster data so that extents align
  - Convert between raster and vector datasets
  - Generate new rasters describing the spatial arrangement of vector data
  - Extract raster values as attributes of vector data

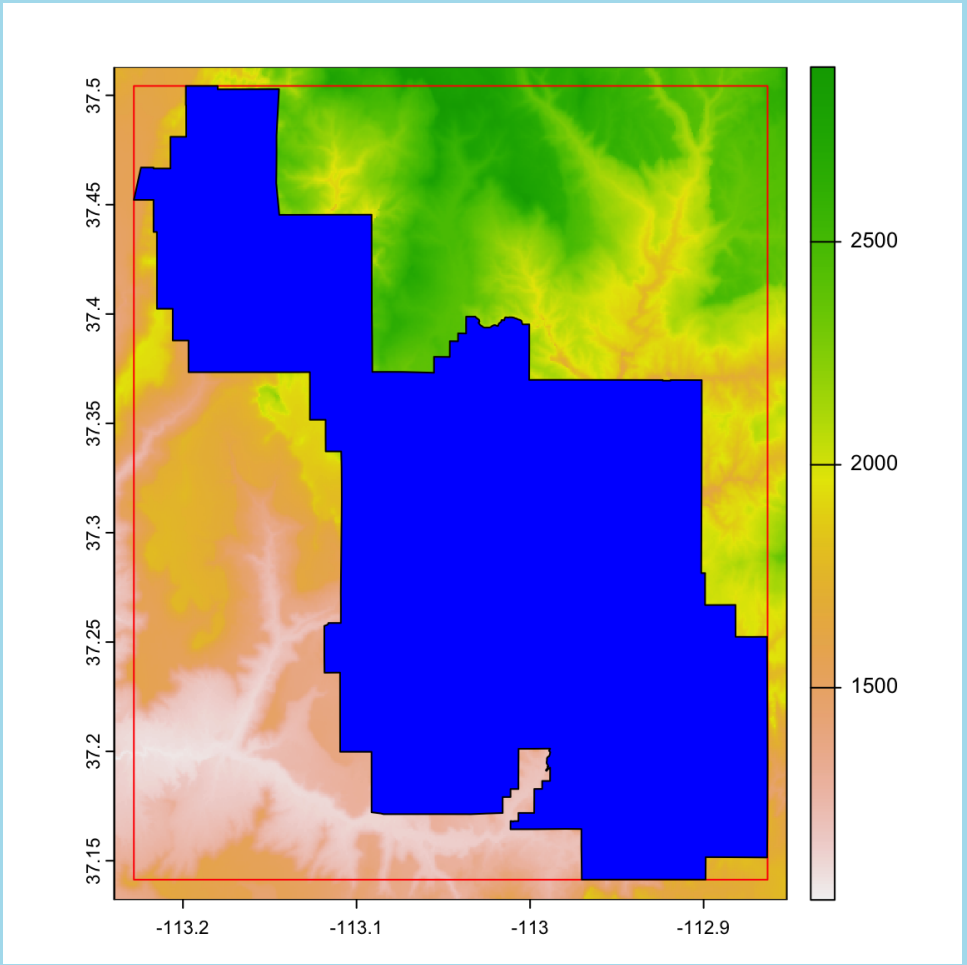
Modifying the Extent

Dealing with Different Extents

- Raster extents often larger than our analysis
- Reducing memory and computational resources
- Making attractive maps



Using `terra::crop()`



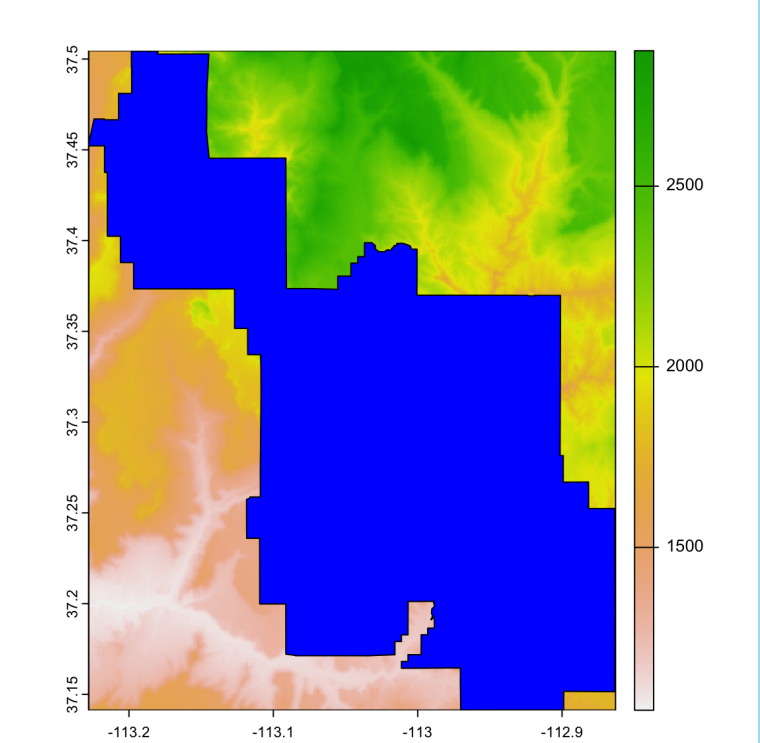
- Coordinate Reference System must be the same for both objects
- Crop is based on the (converted) `SpatExtent` of the 2nd object
- `snap` describes how `y` will be aligned to the raster
- Returns all data within the extent

Using `terra::crop()`

```
1  library(sf)
2  library(terra)
3  library(spDataLarge)
4  srtm = rast(system.file("raster/srtm.tif", package = "spDataLa
5  zion = read_sf(system.file("vector/zion.gpkg", package = "spDa
6  zion = st_transform(zion, crs(srtm))
7
8  crs(srtm) == crs(zion)
```

```
[1] TRUE
```

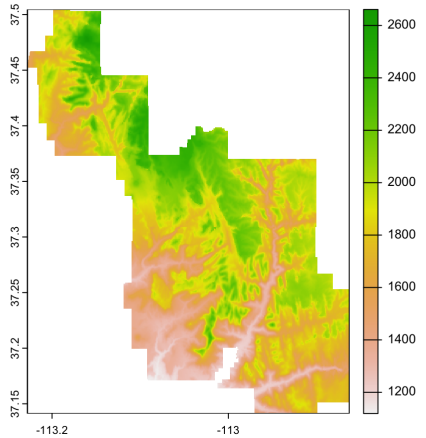
```
1  srtm.crop <- crop(x=srtm, y=zion, snap="near")
```



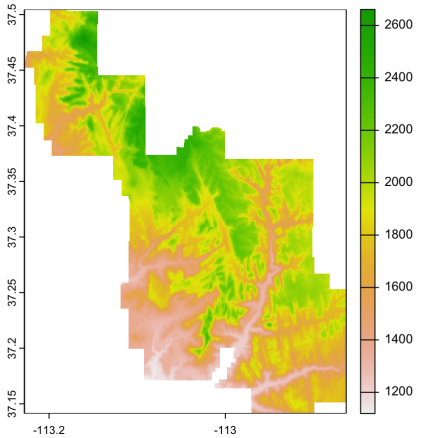
Using `mask()`

- Often want to get rid of all values outside of vector
- Can set `mask=TRUE` in `crop()` (`y` must be `SpatVector`)
- Or use `mask()`

```
1 srtm.crop.msk <- crop(x=srtm, y=ve
2 plot(srtm.crop.msk)
```



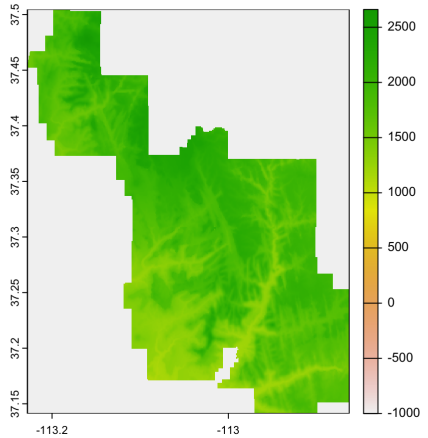

```
1 srtm.msk <- mask(srtm.crop, vect(z
2 plot(srtm.msk)
```



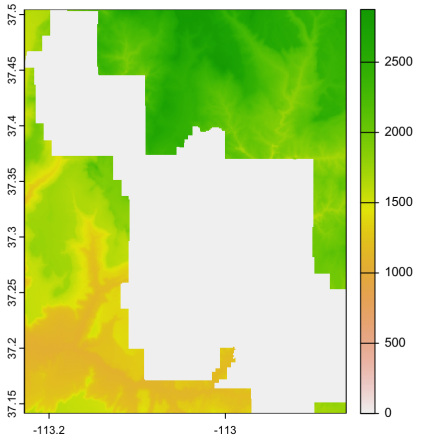
Using `mask()`

- Allows more control over what the mask does
- Can set `maskvalues` and `updatevalues` to change the resulting raster
- Can also use `inverse` to mask out the vector

```
1 srtm.msk <- mask(srtm.crop, vect(z
2 plot(srtm.msk)
```




```
1 srtm.msk <- mask(srtm.crop, vect(z
2 plot(srtm.msk)
```



Extending boundaries

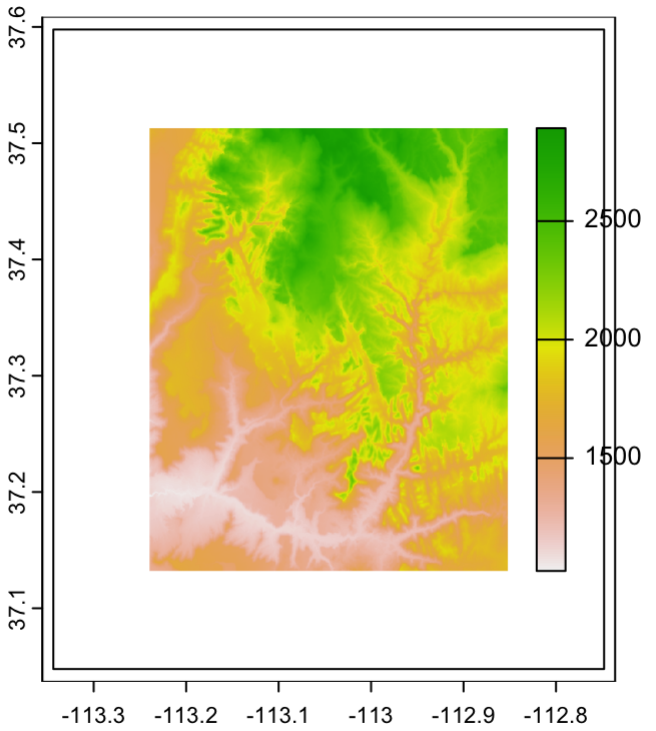
- Vector slightly larger than raster
- Especially when using buffered datasets
- Can use `extend`
- Not exact; depends on `snap()`

```
zion.buff <-  zion %>%
  st_buffer(., 10000)
srtm.ext <- extend(srtm, vect(zion
ext(srtm.ext)
```

```
SpatExtent : -113.343749879444,
-112.74791654615, 37.0479167631968,
37.5979167631601 (xmin, xmax, ymin,
ymax)
```

```
ext(vect(zion.buff))
```

```
SpatExtent : -113.343652923976,
-112.747986193365, 37.0477357596604,
37.5977812137969 (xmin, xmax, ymin,
ymax)
```



Converting Between Formats

Converting Between Formats

- Using coercion (`as`, `rast`, `vect`) can change `class`, but not data model
- Sometimes we need to actually change the data model

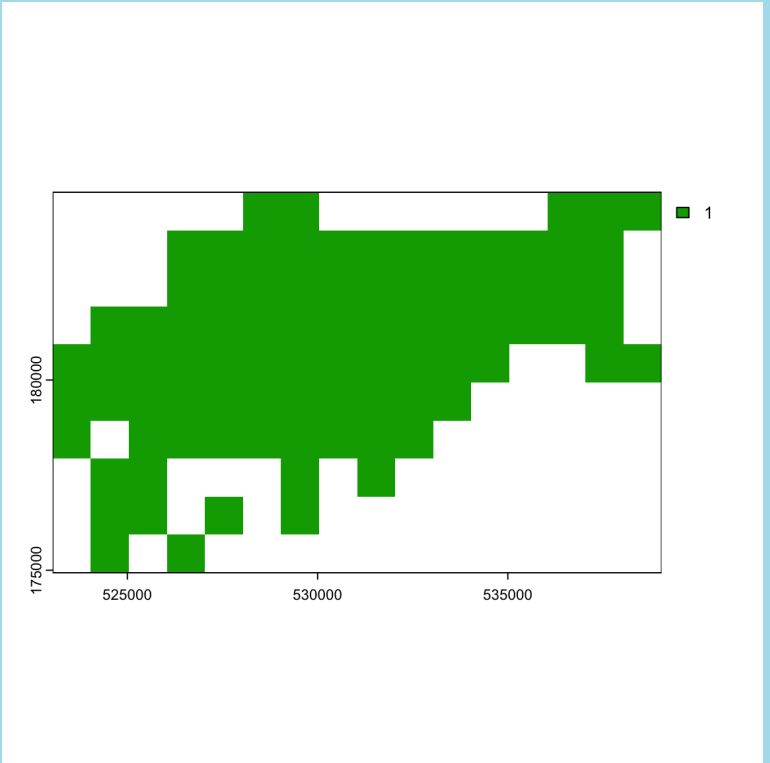
Converting Vectors to Rasters Using `rasterize`

- A special kind of data aggregation
- `x` is your `SpatVector` object
- `y` is a template raster with the appropriate CRS, resolution, and extent
- `fun` allows you to specify the value of the resulting raster

Using `rasterize`

- Presence / Absence
- `field` specifies which value should be returned to non-empty cells

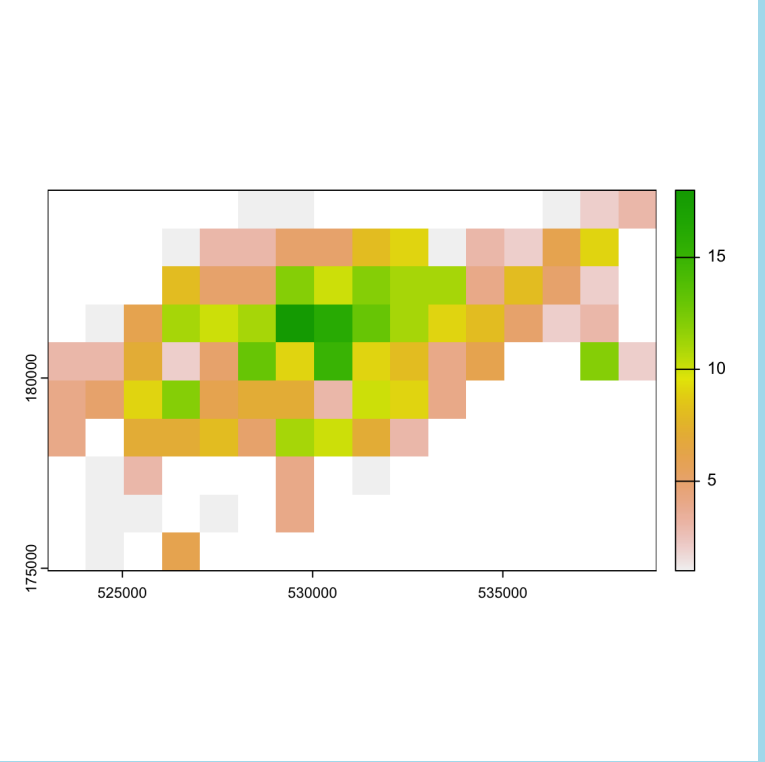
```
1 cycle_hire_osm = spData::cycle_hire_osm
2 cycle_hire_osm_projected = st_transform(cycle_hire_osm, "EPSG:
3 raster_template = rast(ext(cycle_hire_osm_projected), resoluti
4                        crs = st_crs(cycle_hire_osm_projected)$
5 ch_raster1 = rasterize(cycle_hire_osm_projected, raster_templa
6                        field = 1)
```



Using `rasterize`

- The `fun` argument specifies how we aggregate the data
- Useful for counting occurrences (using `length`)

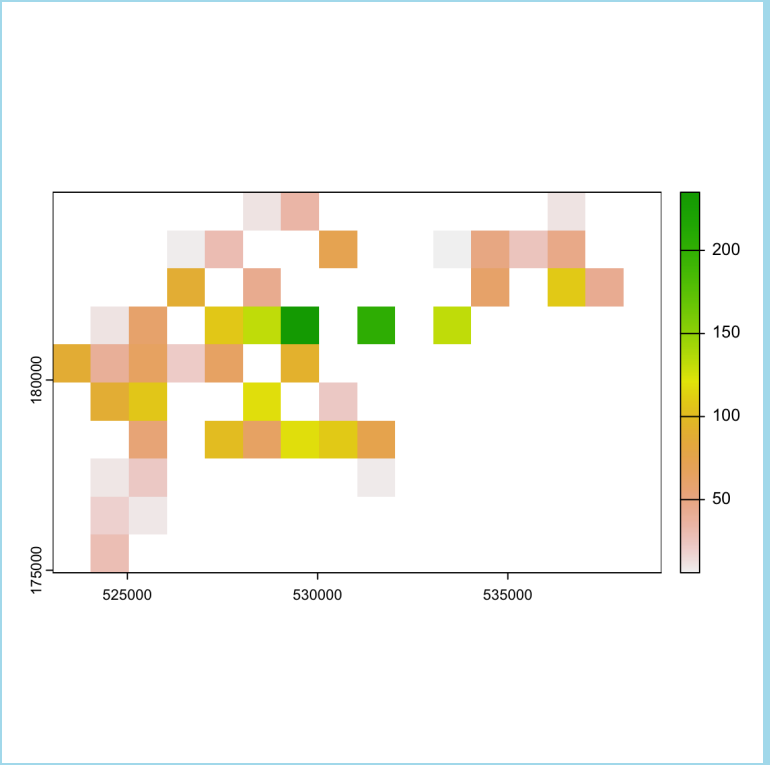
```
1  ch_raster2 = rasterize(cycle_hire_osm_projected, raster_templa
2                         fun = "length")
```



Using `rasterize`

- The `fun` argument specifies how we aggregate the data
- Can use a variety of functions

```
ch_raster3 = rasterize(cycle_hire_osm_projected, raster_templa
                       field = "capacity", fun = sum)
```



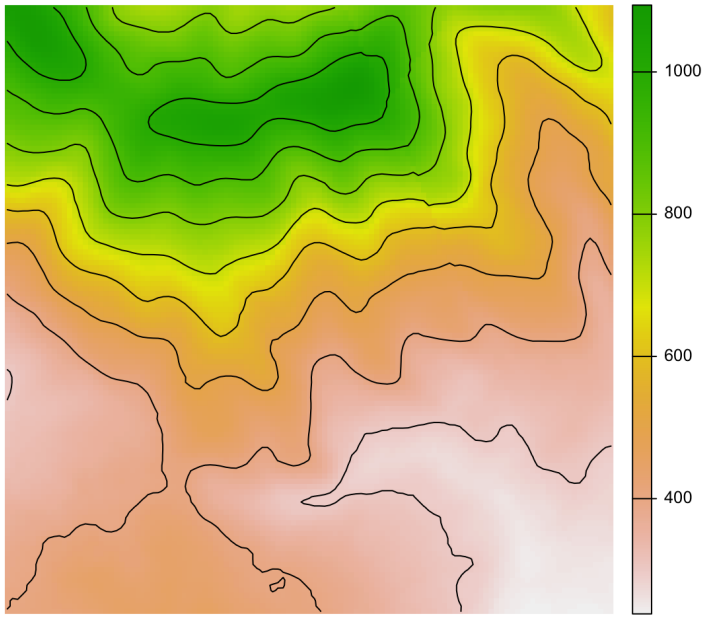
Lines and Polygons

- Can use `rasterize` or `stars::st_rasterize`
- Result depends on the `touches` argument

Converting rasters to vectors

- Less common, but can convert to vector data
- `as.points`, `as.countour`, and `polygonize`

```
1  dem = rast(system.file("raster/dem.tif", package = "spDataLarg
2  cl = as.contour(dem)
```



Generating New Data

Generating New Data

- Sometimes we want a raster describing the spatial context of vector data
- `distance` is a simple method
- We'll use interpolation in the next few weeks

Generating Distance Rasters

- returns a distance matrix or `SpatRaster`

```
cycle_hire_osm = spData::cycle_hire_osm
cycle_hire_osm_projected = st_transform(cycle_hire_osm, "EPSG:27700")

cycle_dist <- distance(vect(cycle_hire_osm_projected))
head(as.matrix(cycle_dist))[1:5, 1:5]
```

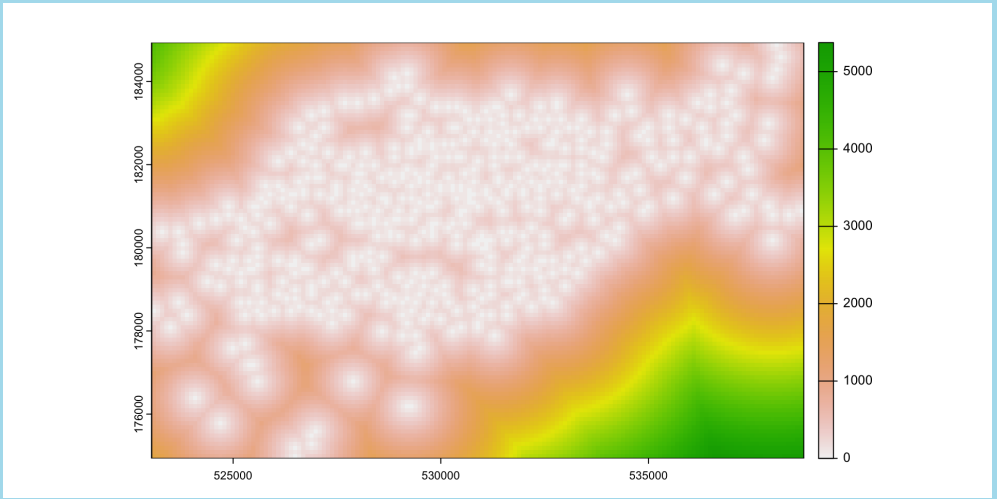
```
          1         2        3         4         5
1    0.0000 2550.9619 1736.118  407.1558 1922.3728
2 2550.9619    0.0000 1072.401 2820.4804  725.2172
3 1736.1178 1072.4011    0.000 1908.1413  360.7490
4  407.1558 2820.4804 1908.141    0.0000 2148.0087
5 1922.3728  725.2172  360.749 2148.0087    0.0000
```

Generating Distance Rasters

- returns a distance matrix or SpatRaster

```
raster_template = rast(ext(cycle_hire_osm_projected), resolution = 100,
                       crs = st_crs(cycle_hire_osm_projected)$wkt)
ch_raster1 = rasterize(cycle_hire_osm_projected, raster_template,
                       field = 1)

ch_dist_rast <- distance(ch_raster1)
plot(ch_dist_rast)
```



Creating Vector Data by Extraction

- Sometimes we want to use rasters to create new attributes
- `fun` controls how the cells are aggregated

```
cycle_hire_osm = spData::cycle_hire_osm
cycle_hire_osm_proj_buff <- st_transform(cycle_hire_osm, "EPSG:27700") %>%
  st_buffer(., 5000) %>%
  as(., "SpatVector")

cycle_ext <- extract(ch_dist_rast, cycle_hire_osm_proj_buff)
head(cycle_ext)
```

```
  ID     last
1  1 1360.147
2  1 1280.625
3  1 1204.159
4  1 1131.371
5  1 1063.015
6  1 1000.000
```