

Operations on Raster Data I

HES 505 Fall 2023: Session 13

Matt Williamson



Today's Plan

Objectives

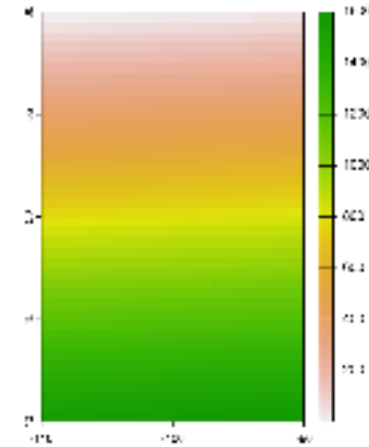
- By the end of today, you should be able to:
 - Align rasters for spatial processing
 - Adjust the resolution of raster data
 - Combine (or reduce) rasters to match the extent of your analysis

Aligning rasters for spatial processing

Projecting raster data

- Transformation from lat/long to planar CRS involves some loss of precision
- New cell values estimated using overlap with original cells
- Interpolation for continuous data, nearest neighbor for categorical data
- Equal-area projections are preferred; especially for large areas

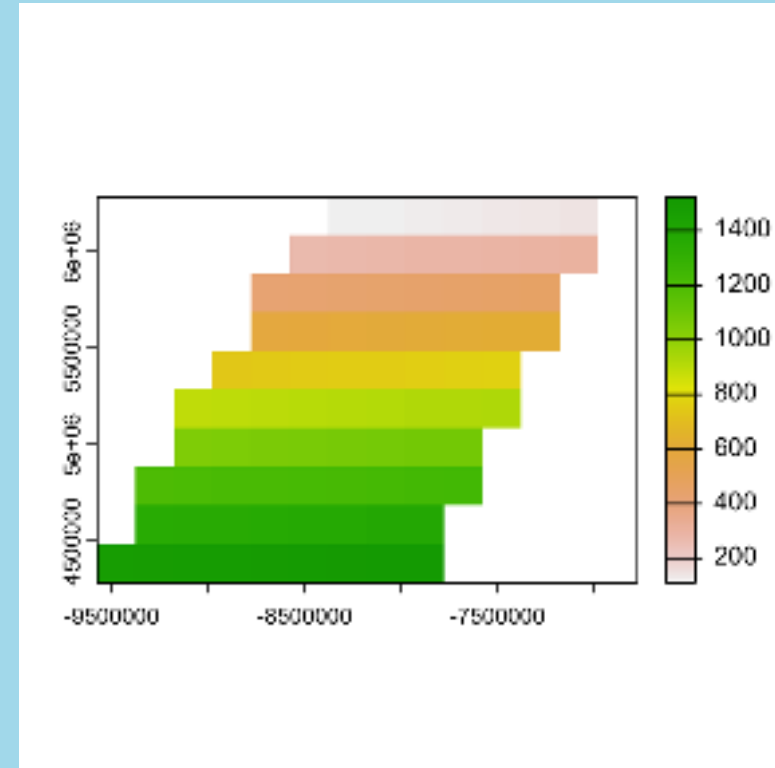
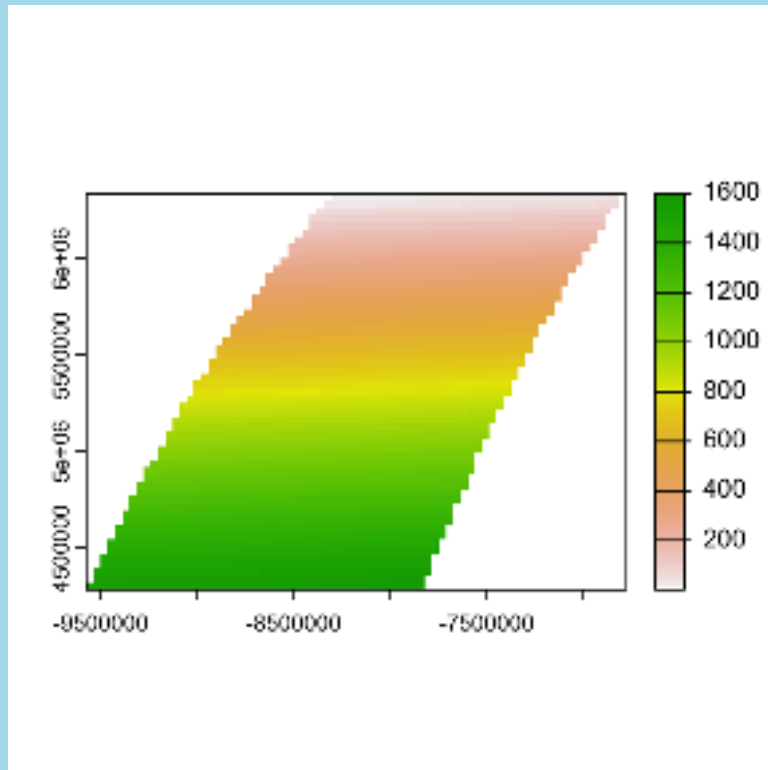
```
1 library(sf)
2 library(terra)
3 library(spDataLarge)
4 r <- rast(xmin=-110, xmax=-90, ymin=40, ymax=60, n
5 values(r) <- 1:ncell(r)
6 plot(r)
```



Projecting raster data

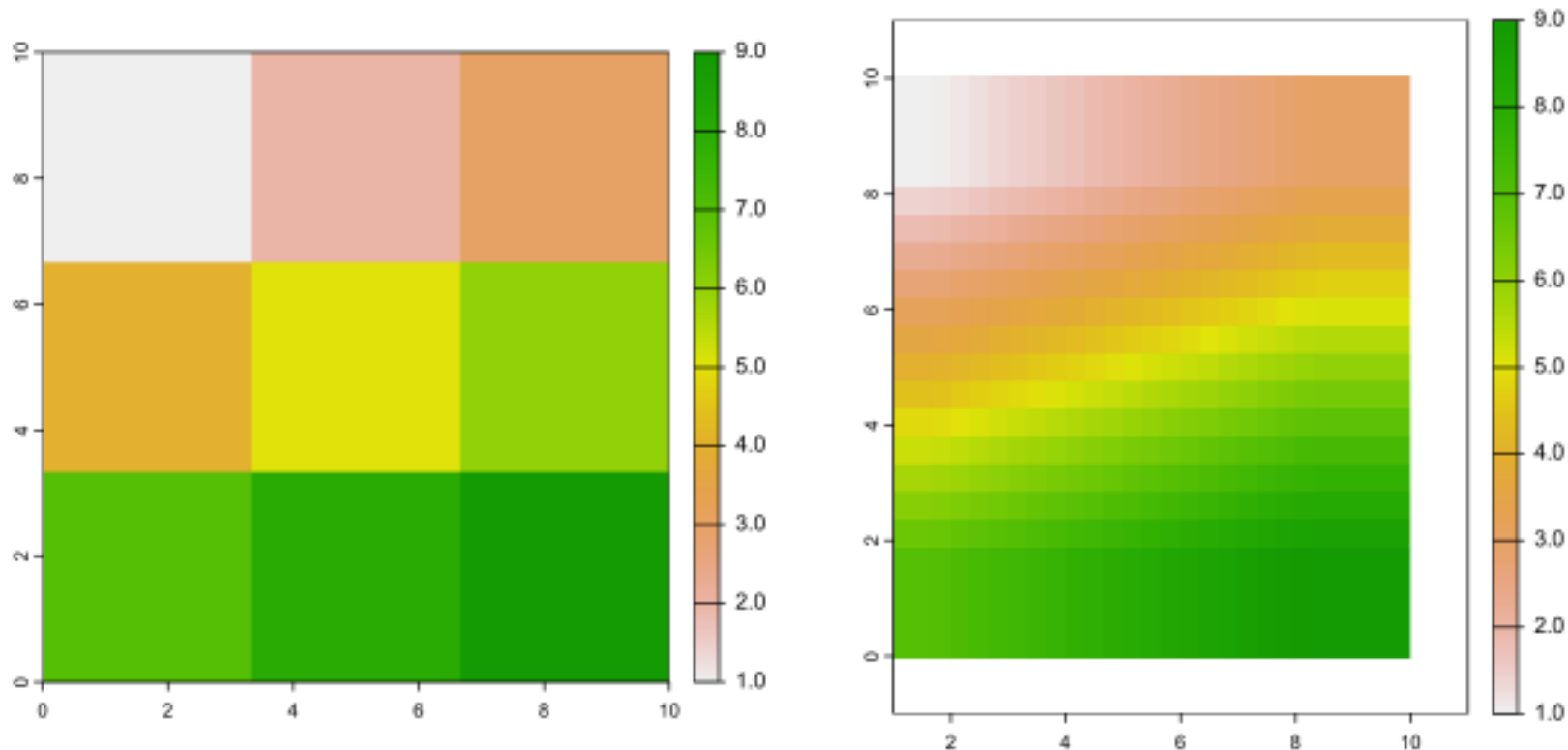
- simple method; alignment not guaranteed
- providing a template to ensure alignment

```
1 newcrs <- "+proj=robin +datum=WGS84"
2 pr1 <- terra::project(r, newcrs)
3 plot(pr1)
```



Aligning Data: **resample**

```
1 r <- rast(nrow=3, ncol=3, xmin=0, xmax=10, ymin=0, ymax=10)
2 values(r) <- 1:ncell(r)
3 s <- rast(nrow=25, ncol=30, xmin=1, xmax=11, ymin=-1, ymax=11)
4 x <- resample(r, s, method="bilinear")
```



Adjusting resolution

Downscaling and Upscaling

- Aligning data for later analysis
- Remembering *scale*
- Thinking about support

Changing resolutions

- **aggregate**, **disaggregate**, **resample** allow changes in cell size
- **aggregate** requires a function (e.g., **mean()** or **min()**) to determine what to do with the grouped values
- **resample** allows changes in cell size **and** shifting of cell centers (slower)

Changing resolutions: aggregate

• **Aggregating** is the process of combining data from multiple sources into a single, unified view.

• This process is often used to create a **data warehouse** or a **data lake**, which can then be used for **analytics** and **reporting**.

• The process of aggregating data involves **extracting** data from various sources, **transforming** it into a common format, and **loading** it into a central repository.

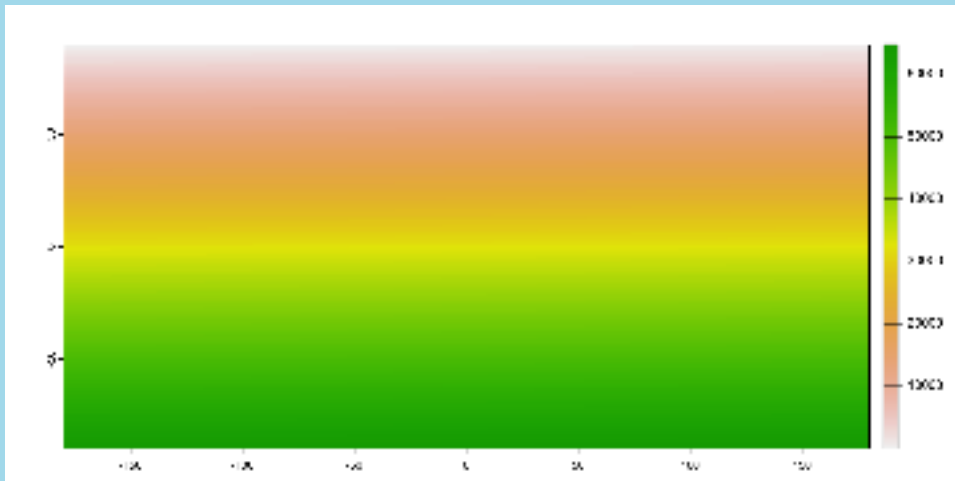
• This process is often automated, allowing for **real-time** data aggregation and analysis.

• The process of aggregating data is a key component of **data integration** and **data management**.

```
1 r <- rast()
2 r
```

```
class      : SpatRaster
dimensions : 180, 360, 1 (nrow,
ncol, nlyr)
resolution : 1, 1 (x, y)
extent      : -180, 180, -90, 90
(xmin, xmax, ymin, ymax)
coord. ref. : lon/lat WGS 84
```

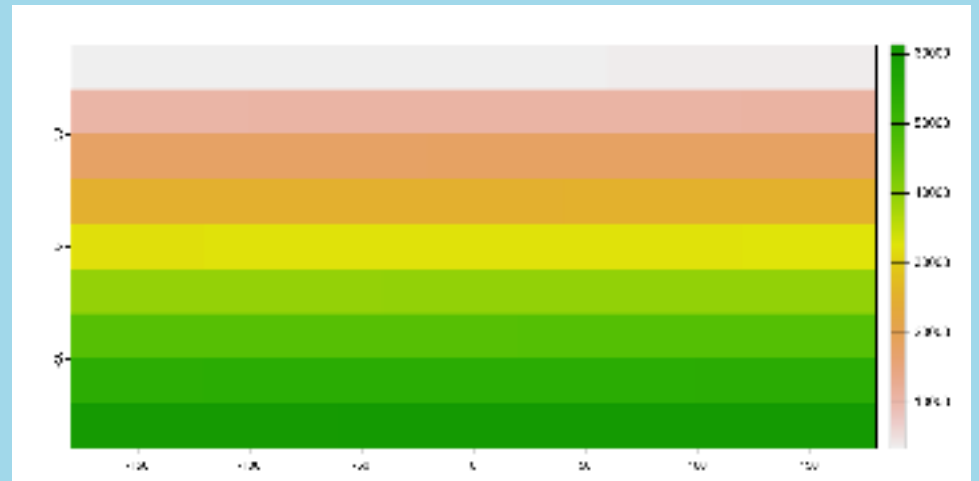
```
1 values(r) <- 1:ncell(r)
2 plot(r)
```



```
1 ra <- aggregate(r, 20)
2 ra
```

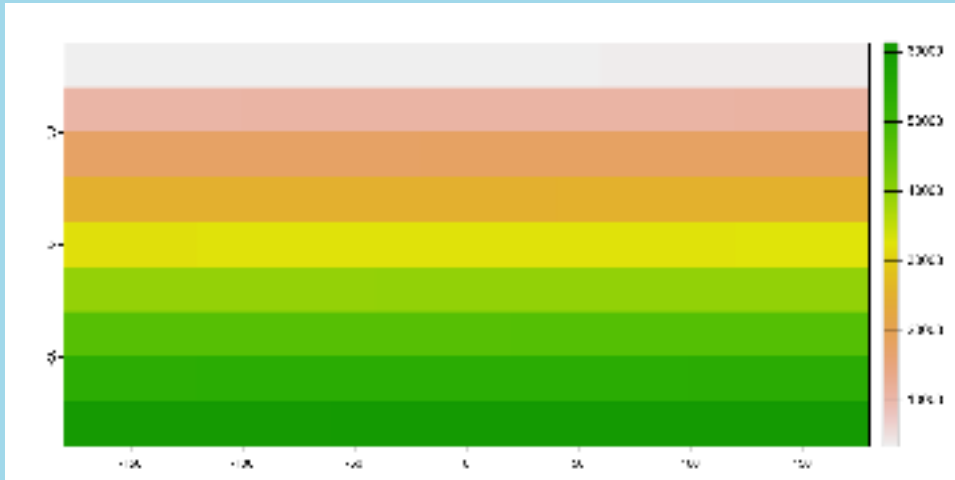
```
class      : SpatRaster
dimensions : 9, 18, 1 (nrow, ncol,
nlyr)
resolution : 20, 20 (x, y)
extent      : -180, 180, -90, 90
(xmin, xmax, ymin, ymax)
coord. ref. : lon/lat WGS 84
source(s)   : memory
name         : lyr.1
min value    : 3430.5
max value    : 61370.5
```

```
1 plot(ra)
```



Changing resolutions: disagg

```
1 ra <- aggregate(r, 20)
2 plot(ra)
```



```
1 rd <- disagg(r, 20)
```

```
|-----|-----|-----|-----|
---|
=====
```

```
1 rd
```

```
class      : SpatRaster
dimensions : 3600, 7200, 1  (nrow,
ncol, nlyr)
resolution : 0.05, 0.05  (x, y)
extent      : -180, 180, -90, 90
(xmin, xmax, ymin, ymax)
coord. ref. : lon/lat WGS 84
source      :
spat_CcWFmodP7kx8nTX_22640.tif
name         : lyr.1
min value    : 1
max value    : 64800
```

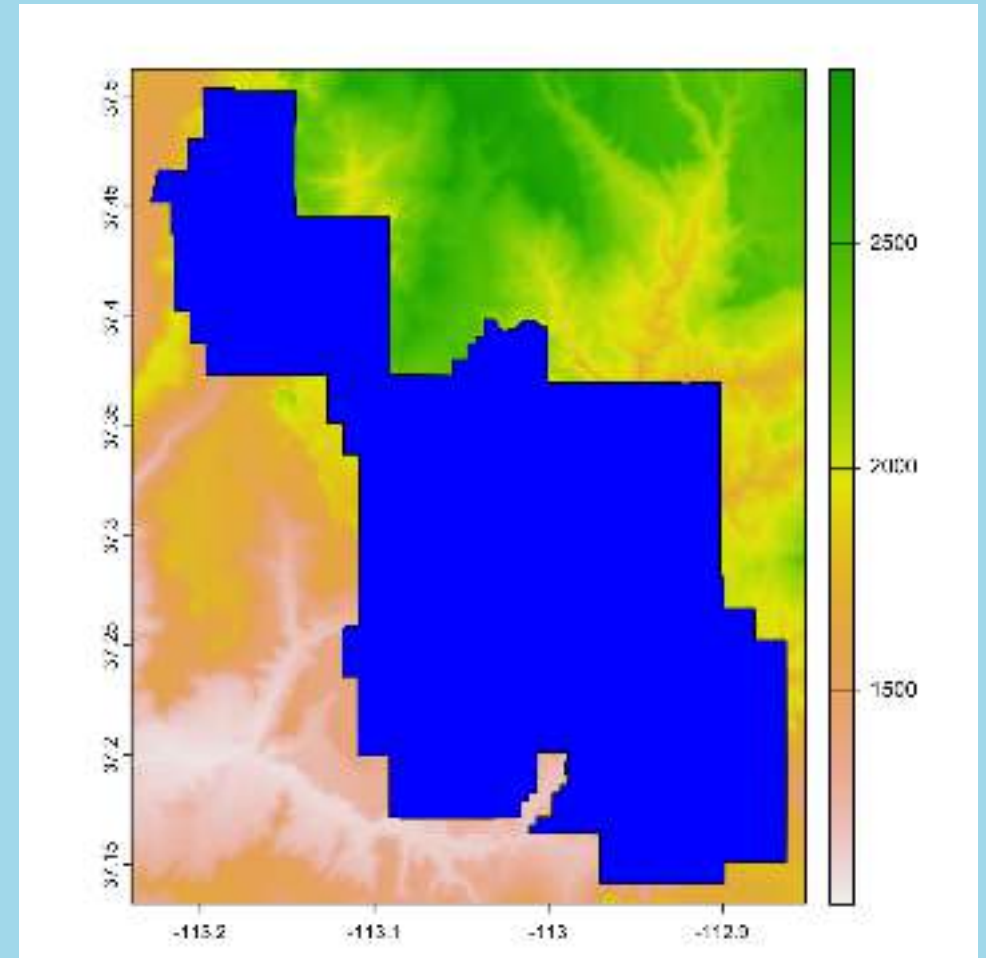
```
1 plot(rd)
```



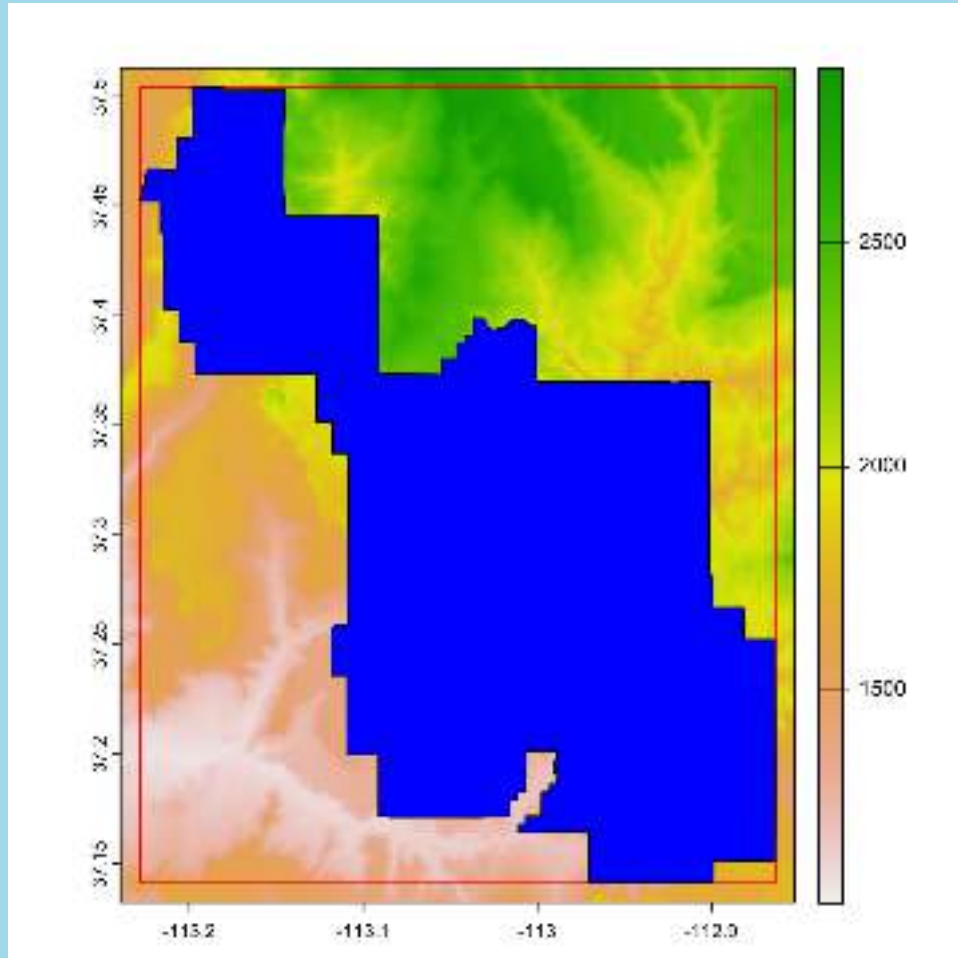
Modifying the Extent

Dealing with Different Extents

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



Using `terra::crop()`



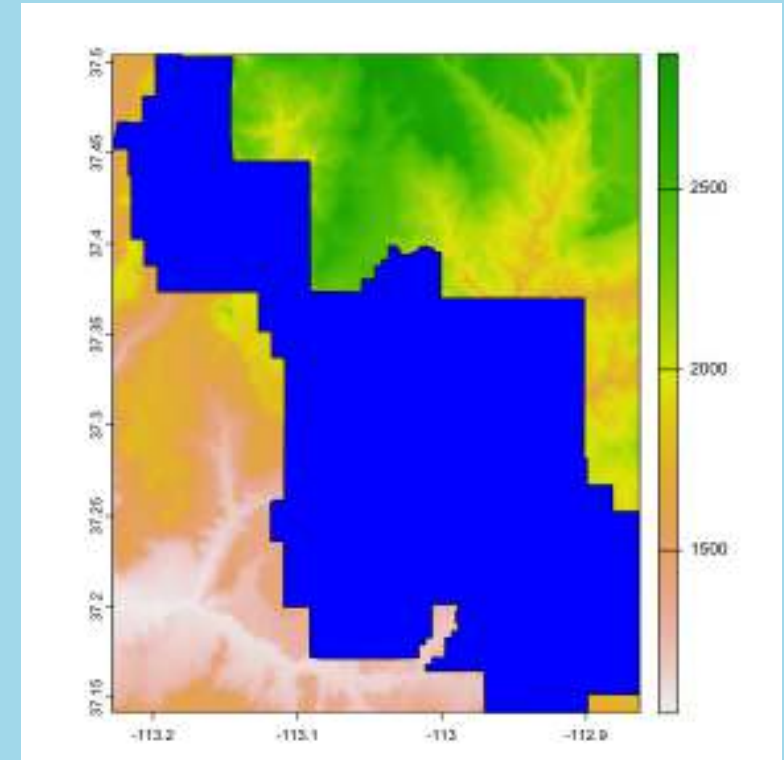
- Coordinate Reference System must be the same for both objects
- Crop is based on the (converted) `SpatExtent` of the 2nd object
- `snap` describes how `y` will be aligned to the raster
- Returns all data within the extent

Using `terra::crop()`

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

```
[1] TRUE
```

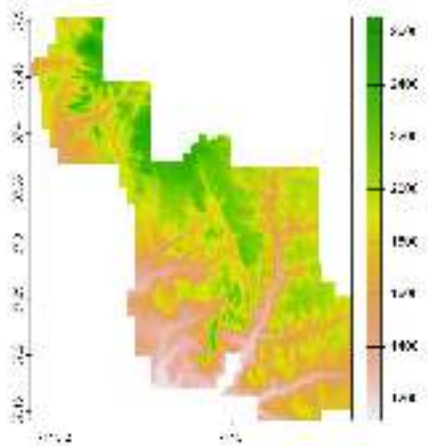
```
1 srtm.crop <- crop(x=srtm, y=zion, snap="near")
```



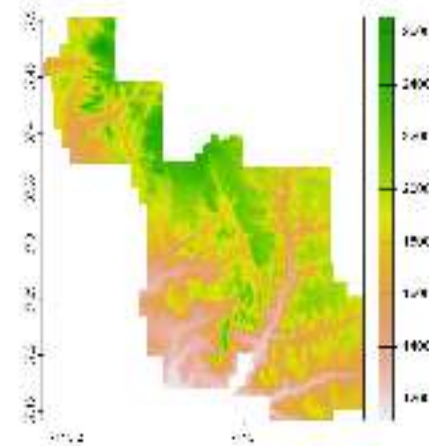
Using `mask()`

- 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)
```



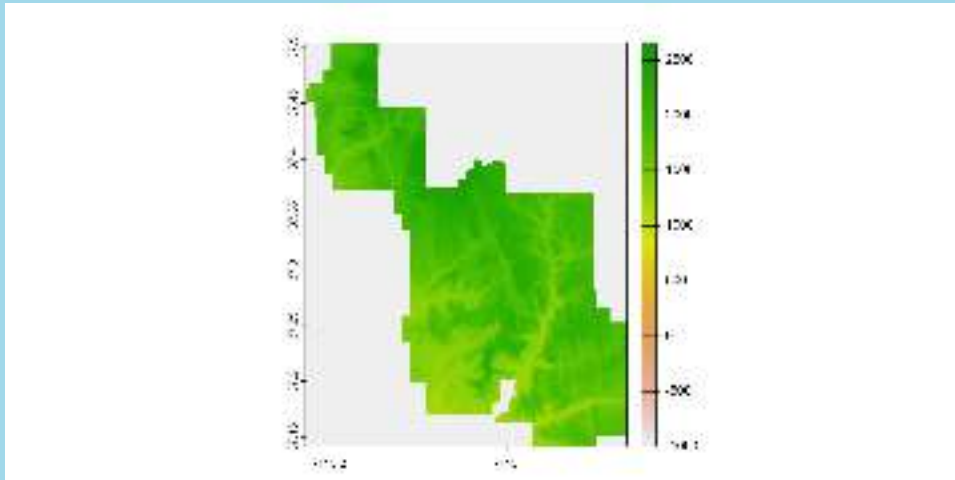
```
1 srtm.msk <- mask(srtm.crop, vect(z  
2 plot(srtm.msk)
```



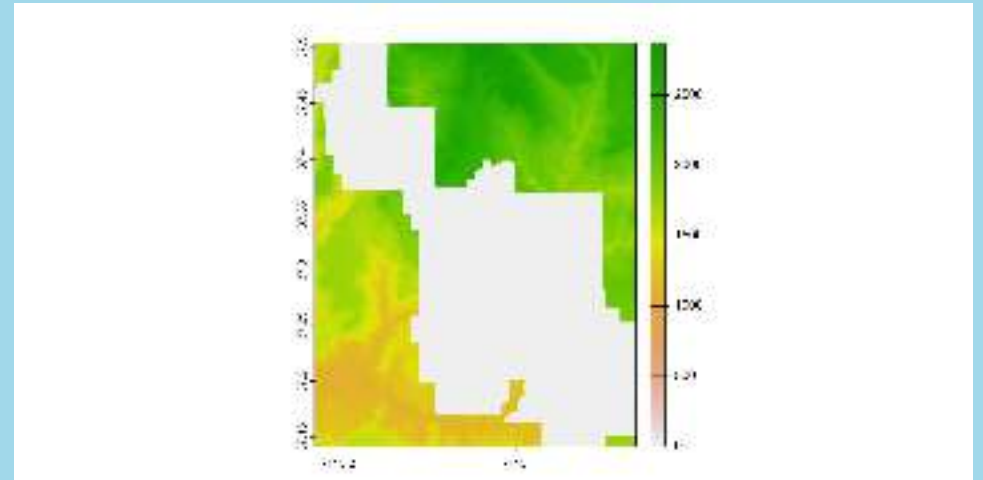
Using `mask()`

- 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()**

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

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

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

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

