Areal Data: Rasters

HES 505 Fall 2024: Session 8

Carolyn Koehn

Announcements

- R server fix: run `install.packages('sf')` in the Console
- Fall Graduate Proposal Showcase is tomorrow, 3-5pm, in the Student Union Building, Simplot AC Ballroom!
- Any questions that came up during the homework?

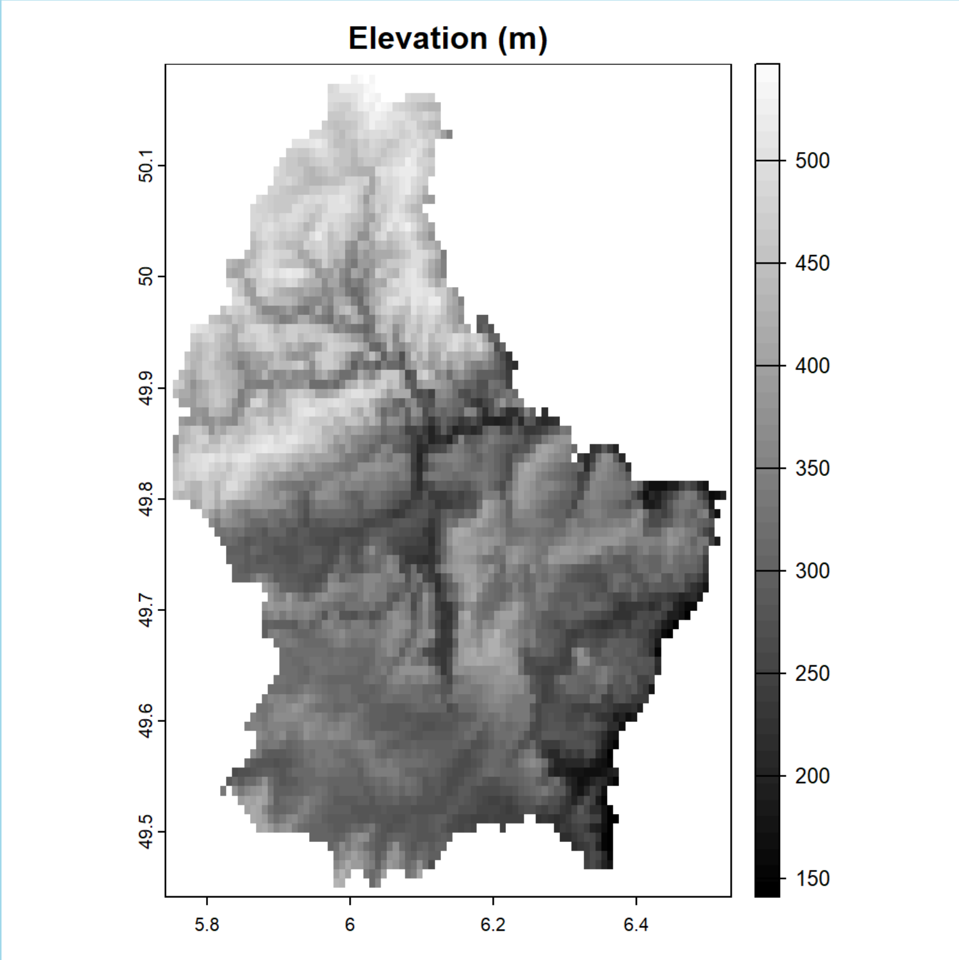
Today's Plan

By the end of today, you should be able to:

- Access the elements that define a raster
- Build rasters from scratch using matrix operations and `terra`
- Evaluate logical conditions with raster data
- Calculate different measures of raster data

Revisiting the Raster Data Model

- **Vector data** describe the "exact" locations of features on a landscape (including a Cartesian landscape)
- **Raster data** represent spatially continuous phenomena (`NA` is possible)
- Depict the alignment of data on a regular lattice (often a square)
  - Operations mimic those for `matrix` objects in `R`
- Geometry is implicit; the spatial extent and number of rows and columns define the cell size



Rasters with `terra`

- syntax is different for `terra` compared to `sf`
- Representation in `Environment` is also different
- Can break pipes, **Be Explicit**

Rasters by Construction

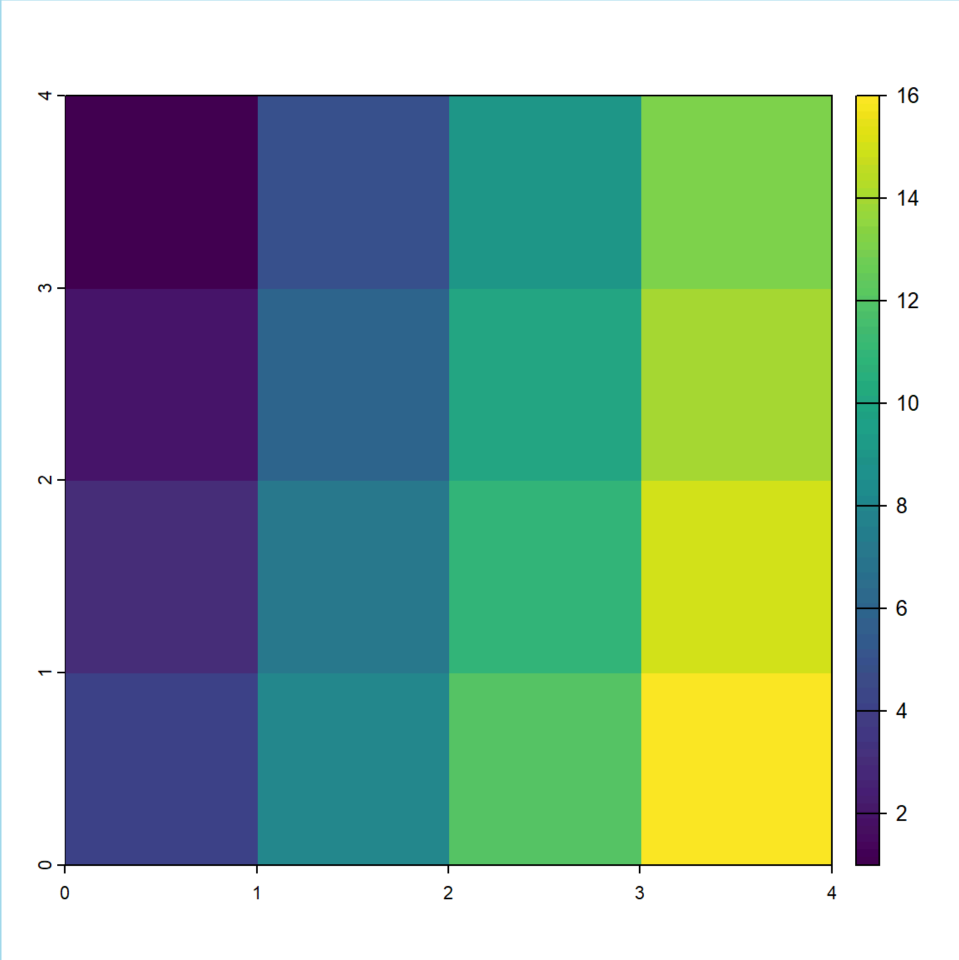
Rasters by Construction

```r
mtx <- matrix(1:16, nrow=4
mtx
```

```
     [,1] [,2] [,3] [,4]
[1,]    1    5    9   13
[2,]    2    6   10   14
[3,]    3    7   11   15
[4,]    4    8   12   16
```

```r
rstr <- terra::rast(mtx)
rstr
```

```
class       : SpatRaster
dimensions  : 4, 4, 1  (nrow, ncol,
nlyr)
resolution  : 1, 1  (x, y)
extent      : 0, 4, 0, 4  (xmin,
xmax, ymin, ymax)
coord. ref. :
source(s)   : memory
name        : lyr.1
min value   :     1
max value   :    16
```



Rasters by Construction: Origin

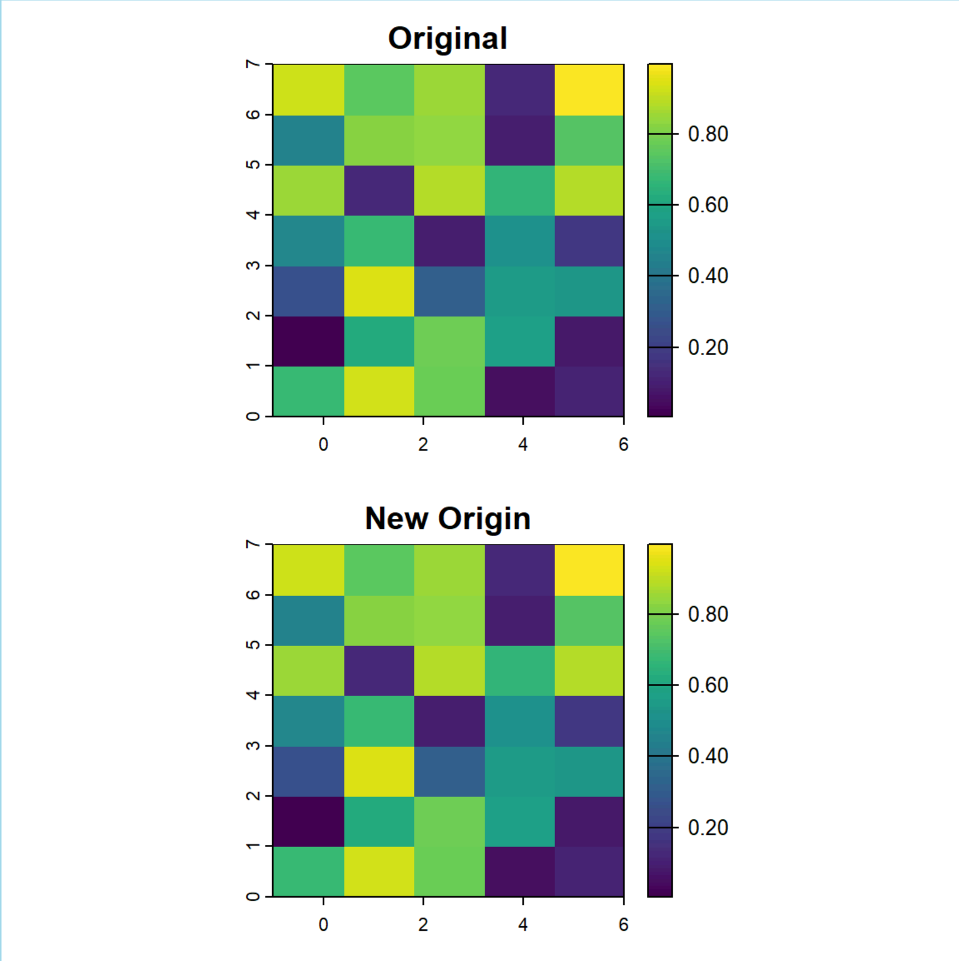
- Origin defines the location of the intersection of the x and y axes

- Ideally, the origin is the cell “corner” closest to `c(0, 0)`

```
r <- rast(xmin=-4, xmax =
r[] <- runif(ncell(r))
origin(r)
```

```
[1] 0.05 0.00
```

```
r2 <- r
origin(r2) <- c(2,2)
```



Rasters by Construction: Resolution

- Geometry is implicit; the spatial extent and number of rows and columns define the cell size
- **Resolution** (`res`) defines the length and width of an individual pixel

```
r <- rast(xmin=-4, xmax =
              ncols=10)
res(r)
```

```
[1] 1.35 1.00
```

```
r <- rast(xmin=-4, xmax =
              res=c(0.5,0.5))
ncol(r)
```

```
[1] 27
```

```
r2 <- rast(xmin=-4, xmax =
              ncols=10)
res(r2)
```

```
[1] 0.9 1.0
```

```
r2 <- rast(xmin=-4, xmax =
              res=c(5,5))
ncol(r2)
```

```
[1] 3
```

Predicates and measures in `terra`

Extending predicates

- **Predicates**: evaluate a logical statement asserting that a property is `TRUE`
- `terra` does not follow the same hierarchy as `sf` so a little trickier

Unary predicates in `terra`

- Can tell us qualities of a raster dataset
- Many similar operations for `SpatVector` class (note use of `.`)

predicate	asks...
`is.lonlat`	Does the object have a longitude/latitude CRS?
`inMemory`	is the object stored in memory?
`is.factor`	Are there categorical layers?
`hasValues`	Do the cells have values?

Unary predicates in terra

- `global`: tests if the raster covers all longitudes (from -180 to 180 degrees) such that the extreme columns are in fact adjacent

```r
r <- rast()
is.lonlat(r)
```

```
[1] TRUE
```

```r
is.lonlat(r, global=TRUE)
```

```
[1] TRUE
```

- `perhaps`: If TRUE and the crs is unknown, the method returns TRUE if the coordinates are plausible for longitude/latitude

```r
crs(r) <- ""
is.lonlat(r)
```

```
[1] NA
```

```r
is.lonlat(r, perhaps=TRUE, warn=FALSE)
```

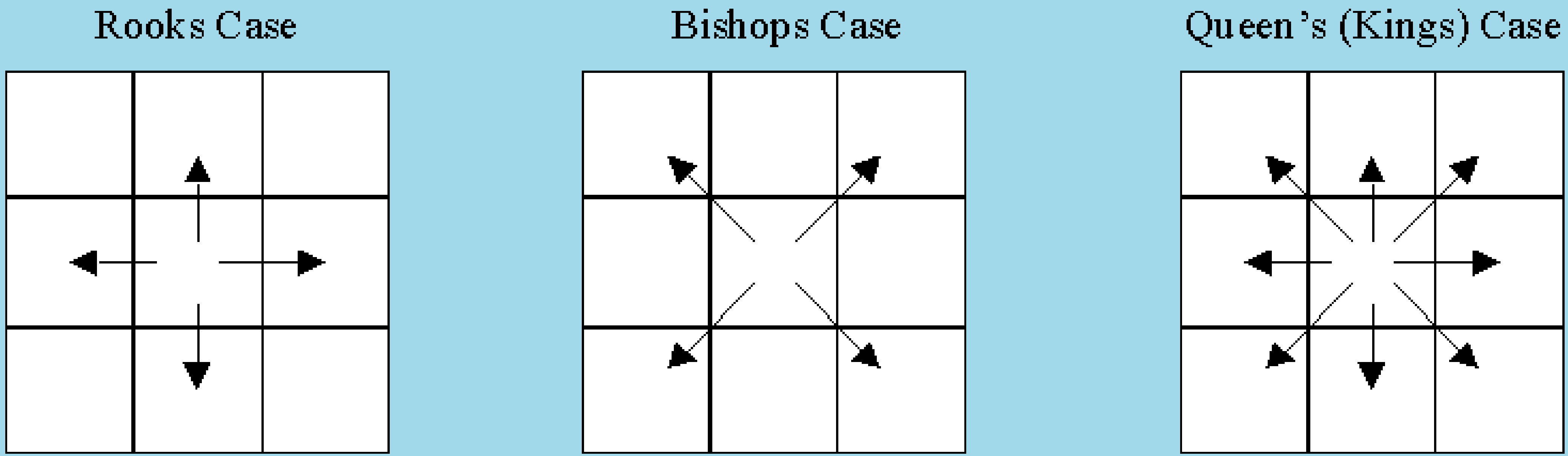
```
[1] TRUE
```

```r
crs(r) <- "+proj=lcc +lat_1=48 +lat_2=33 +
is.lonlat(r)
```

```
[1] FALSE
```

Binary predicates in `terra`

- Take exactly 2 inputs, return 1 matrix of cell locs where value is `TRUE`
- `adjacent`: identifies cells adajcent to a set of raster cells



Unary measures in `terra`

- Slightly more flexible than `sf`
- One result for each layer in a stack

measure	returns
`cellSize`	area of individual cells
`expanse`	summed area of all cells
`values`	returns all cell values
`ncol`	number of columns
`nrow`	number of rows
`ncell`	number of cells
`res`	resolution
`ext`	minimum and maximum of x and y coords
`origin`	the orgin of a `SpatRaster`
`crs`	the coordinate reference system
`cats`	categories of a categorical raster

Binary measures in `terra`

- Returns a matrix or `SpatRaster` describing the measure

measure	returns
`distance`	shortest distance to non-NA or vector object
`gridDistance`	shortest distance through adjacent grid cells
`costDist`	Shortest distance considering cell-varying friction
`direction`	azimuth to cells that are not `NA`

Practice

1. Create an empty raster (a raster with no cell values) with 5 rows, 5 columns, and a resolution of 2 units with a total of 25 cells. You may need to use the `rast` helpfile and check the argument descriptions.
2. Fill the raster's cells with random values (hint: remember how we reassigned the origin? Use another unary measure). Use a unary predicate to test if the raster has values. Plot your raster.
3. Print the default origin of your raster.
4. Reassign the values of cells adjacent to cell 12 to `NA` and plot. Try different cases if time allows.
5. With some cells set to `NA` (from step 4), create and plot a raster showing the distance to the nearest non-`NA` cell.

Extra Practice

1. Run the examples for `costDist` and `gridDistance` to see how those functions can be used.
2. Load the `wildfire_hazard_agg.tif` data from the `assignment03` folder. Use the data as the input for the `distance` function and plot the result. How might this be useful in your research?

Today's Objectives

By the end of today, you should be able to:

- Access the elements that define a raster
- Build rasters from scratch using matrix operations and `terra`
- Evaluate logical conditions with raster data
- Calculate different measures of raster data