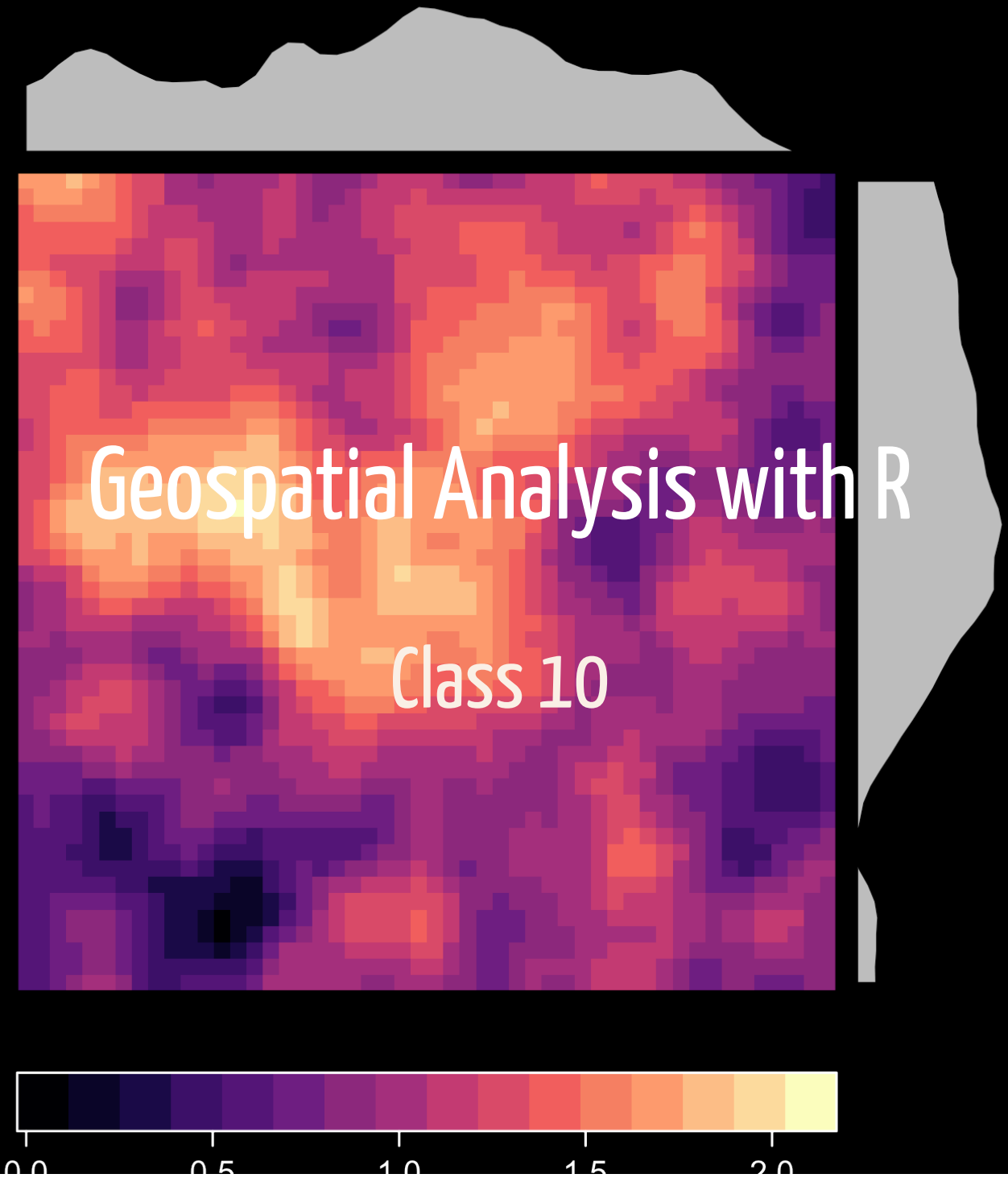
Geospatial Analysis with R

Class 10



Today

- A little R/Rmarkdown syntax
- Control structures (*apply)

Create your own data

- Create three matrices, `m1`, `m2`, `m3`
  - `m1`:
    - Random number seed: `set.seed(100)`
    - `V1` = 1:20
    - `V2` 20 random numbers from 1:100
    - `V3` 20 numbers from `rnorm`, mean = 500, sd = 100)
  - `m2`: Same variables, but `set.seed(200)`
  - `m3`: Same variables, but `set.seed(300)`
- Create vector `v`: 20 random draws from `LETTERS[1:5]`; `set.seed(1)`
- Create 2 `data.frame`s: `dat1` (`m1` & `v`), `dat2` (`m2` & `v`)
  - Name the column holding `v` "GRP"
- Create list `l` combining `m1`, `m2`, `m3`, `dat1`, `dat2`, `v`
  - Name the list elements the same as the object name.

```r
set.seed(100)
m1 <- cbind(V1 = 1:20, V2 = sample(1:100, size = 20, replace = TRUE),
            V3 = rnorm(n = 20, mean = 500, sd = 100))
set.seed(200)
m2 <- cbind(V1 = 1:20, V2 = sample(1:100, size = 20, replace = TRUE),
            V3 = rnorm(n = 20, mean = 500, sd = 100))
set.seed(300)
m3 <- cbind(V1 = 1:20, V2 = sample(1:100, size = 20, replace = TRUE),
            V3 = rnorm(n = 20, mean = 500, sd = 100))
set.seed(1)
v <- sample(LETTERS[1:5], size = 20, replace = TRUE)
dat1 <- data.frame(m1, "GRP" = v)
dat2 <- data.frame(m2, "GRP" = v)
l <- list(m1, m2, m3, dat1, dat2, v)
names(l) <- c("m1", "m2", "m3", "dat1", "dat2", "v")
```

Control structures

- Branching
  - Not usually vectorized
  - `ifelse` for vectorized branching
  - often done within looping
- Looping
  - `for`, `while`
  - `*apply`
    - A special form of looping
    - Intended for *applying* a function to data
    - Returns results of loop directly into object

lapply

- Apply function to vector, return list

```
l2 <- l[c("m1", "m2", "m3")]
lapply(l2, mean)
```

```
## $m1
## [1] 187.5092
##
## $m2
## [1] 183.919
##
## $m3
## [1] 197.5562
```

sapply

- Apply function to vector, return simplest possible output

```
l2 <- l[c("m1", "m2", "m3")]
lapply(l2, mean)
```

```
## $m1
## [1] 187.5092
##
## $m2
## [1] 183.919
##
## $m3
## [1] 197.5562
```

apply

- Apply function to margins of array or matrix, return vector, array, or list

```
apply(l2[[1]], MARGIN = 1, FUN = mean)
```

```
##  [1] 180.3295 179.2091 179.6122 194.6613 188.1127 184.0228 183.3715
##  [8] 199.0285 157.5395 253.0099 176.7303 225.8020 189.3987 210.4468
## [15] 170.1874 179.7183 155.3259 192.3648 146.4090 204.9025
```

```
apply(l2[[1]], MARGIN = 2, FUN = mean)
```

```
##       V1       V2       V3
##  10.5000  46.8000 505.2275
```

*apply with anonymous functions

- Use anonymous functions to pass an iterator
- For more complex operations inside `({})`

```
# Simple
o <- lapply(1:2, function(x) l2[[x]])
o
```

```
## [[1]]
##       V1 V2       V3
##  [1,]  1 31 508.9886
##  [2,]  2 26 509.6274
##  [3,]  3 56 479.8366
##  [4,]  4  6 573.9840
##  [5,]  5 47 512.3380
##  [6,]  6 49 497.0683
##  [7,]  7 82 461.1146
##  [8,]  8 38 551.0856
##  [9,]  9 55 408.6186
## [10,] 10 18 731.0297
## [11,] 11 63 456.1910
## [12,] 12 89 576.4061
## [13,] 13 29 526.1961
## [14,] 14 40 577.3405
## [15,] 15 77 418.5621
## [16,] 16 67 456.1549
## [17,] 17 21 427.9778
## [18,] 18 36 523.0945
```

```r
# More complex
o2 <- lapply(1:5, function(x) {
  l2[[1]][x] - l2[[2]][x]
})
o2
```

```
## [[1]]
## [1] 0
##
## [[2]]
## [1] 0
##
## [[3]]
## [1] 0
##
## [[4]]
## [1] 0
##
## [[5]]
## [1] 0
```

Create data using `lapply`

- Let's recreate our matrix examples

Create data using `lapply`

- Let's recreate our matrix examples

```
seeds <- c(100, 200, 300)  # or 100 * 1:3
l3 <- lapply(seeds, function(x) {
  set.seed(x)
  m <- cbind(V1 = 1:20,
             V2 = sample(1:100, size = 20, replace = TRUE),
             V3 = rnorm(n = 20, mean = 500, sd = 100))
})
names(l3) <- paste0("m", 1:3)
```

Check the values

- Let's check them now against original values

```
m1 == l3$m1
m2 == l3$m2
all(m1 == l3$m1)
all(m2 == l3$m2)
```

- Etc, but we could do this check with a looping function!

Check using `lapply`

- We know that `l[1:3]` contains `m1`, `m2`, `m3`, and so does `l3`, so

```
lapply(1:3, function(x) all(l[[x]] == l3[[x]]))
```

- More compact

```
sapply(1:3, function(x) all(l[[x]] == l3[[x]]))
```

Looping practice

- Write a `for` loop that iterates through the vector 1:10 and prints the iterator `i` multiplied by 10

Looping practice

- Write a `for` loop that iterates through the vector 1:10 and prints the iterator `i` multiplied by 10
- Do the same, but instead of print `i * 10`, catch the result in a predefined empty list `o`

Looping practice

- Write a `for` loop that iterates through the vector 1:10 and prints the iterator `i` multiplied by 10
- Do the same, but instead of print `i * 10`, catch the result in a predefined empty list `o`
- Do the same as above, but use an `lapply` that assigns output to `o`

Looping practice

- Write a `for` loop that iterates through the vector 1:10 and prints the iterator `i` multiplied by 10
- Do the same, but instead of print `i * 10`, catch the result in a predefined empty list `o`
- Do the same as above, but use an `lapply` that assigns output to `o`
- Do the same as above, but use `sapply` instead of `lapply`

Looping practice

- Write a `for` loop that iterates through the vector 1:10 and prints the iterator `i` multiplied by 10
- Do the same, but instead of print `i * 10`, catch the result in a predefined empty list `o`
- Do the same as above, but use an `lapply` that assigns output to `o`
- Do the same as above, but use `sapply` instead of `lapply`
- Let's use `sapply` to find which elements of `l` are `matrix`

Looping practice

- Write a `for` loop that iterates through the vector 1:10 and prints the iterator `i` multiplied by 10
- Do the same, but instead of print `i * 10`, catch the result in a predefined empty list `o`
- Do the same as above, but use an `lapply` that assigns output to `o`
- Do the same as above, but use `sapply` instead of `lapply`
- Let's use `sapply` to find which elements of `l` are `matrix`
- Let's use `lapply` to calculate the `colMeans` of matrices and `data.frame`s in `l`
- Use `lapply` to calculate and output `mean` and `sd` of each matrix in `l`

Looping practice

- Write a `for` loop that iterates through the vector 1:10 and prints the iterator `i` multiplied by 10
- Do the same, but instead of print `i * 10`, catch the result in a predefined empty list `o`
- Do the same as above, but use an `lapply` that assigns output to `o`
- Do the same as above, but use `sapply` instead of `lapply`
- Let's use `sapply` to find which elements of `l` are `matrix`
- Let's use `lapply` to calculate the `colMeans` of matrices and `data.frame`s in `l`
- Use `lapply` to calculate and output `mean` and `sd` of each matrix in `l`
- Do the same, but use conditional to test whether each element of `l` is `matrix`, then calculate `mean`, `sd`.