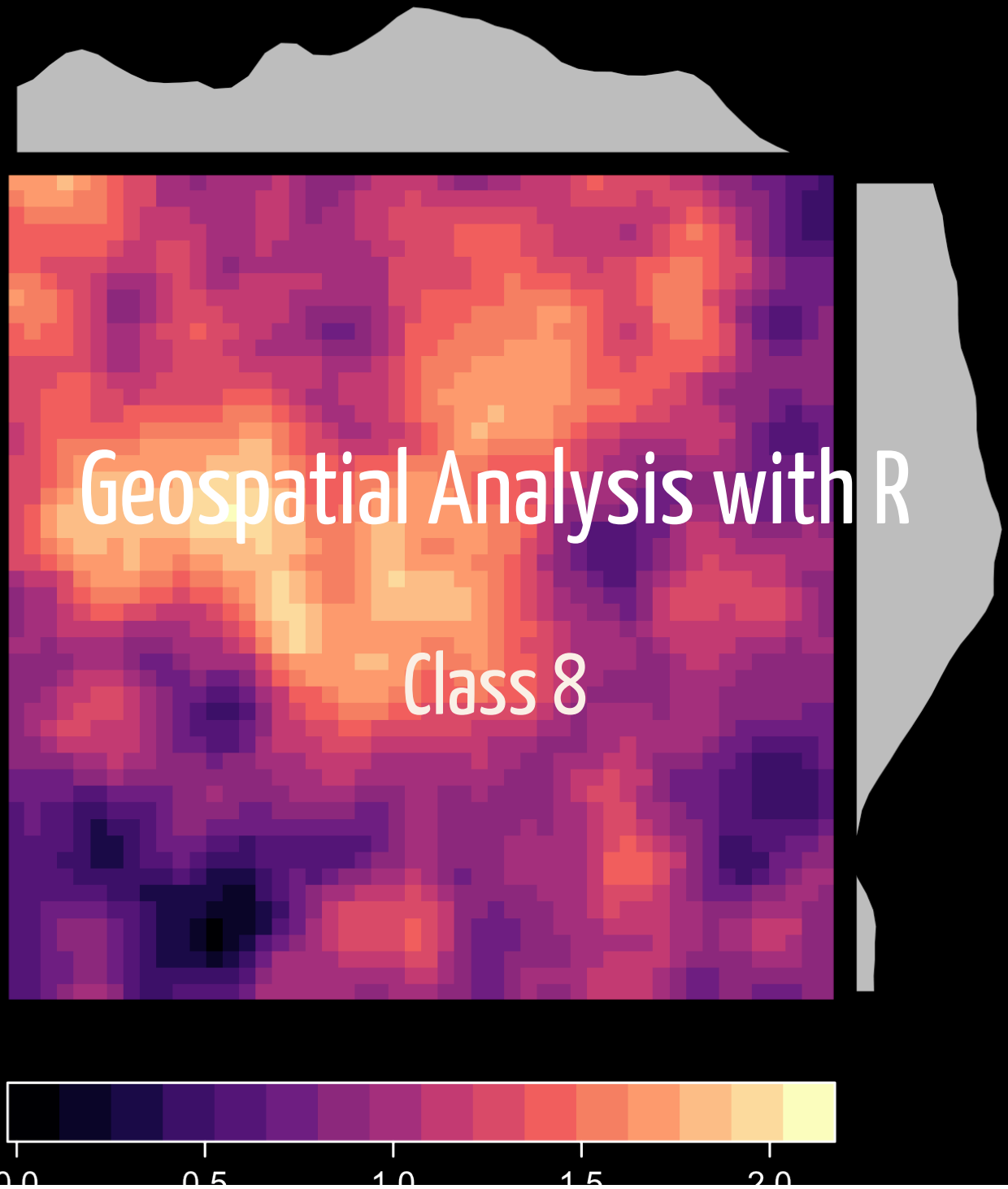
Geospatial Analysis with R

Class 8



Today

- More coding practice
  - Indexing
  - A little bit of summarizing
  - Control structures

Create your own data

- Create the following:
  - `a`: a random vector of integers with 10 elements drawn from 1-20:
  - Use the `sample` function with `set.seed(10)`
  - Name the elements of `a` with a vector of names starting with "V1" and ending with "V10".
    - Use the `paste0` function to create those names.
    - Create the identical vector of names using the `paste` function.
  - `b`: Using `a` as an index to select from `letters`
  - `d`: Use `rnorm` with a mean = 100 and an sd of 20
  - Why did I skip `c`?
  - Create a list `l` from `a`, `b`, `d`.
  - Assign the names of the vectors in `l` to the `l`'s elements

```r
set.seed(10)
a <- sample(1:20, 10, replace = TRUE)
names(a) <- paste0("V1", 1:10)
names(a) <- paste("V1", 1:10, sep = "")
b <- letters[a]
d <- rnorm(n = 10, mean = 100, sd = 20)
l <- list("a" = a, "b" = b, "d" = d)
l
```

```
## $a
##  V11  V12  V13  V14  V15  V16  V17  V18  V19 V110
##   11    7    9   14    2    5    6    6   13    9
##
## $b
##  [1] "k" "g" "i" "n" "b" "e" "f" "f" "m" "i"
##
## $d
##  [1] 107.79589  75.83848  92.72648  67.46655  94.87043 122.03559 115.11563
##  [8]  95.23533 119.74889 114.82780
```

2-d structures

- Create the following:
  - `m`: a matrix with three integer columns named "V1", "V2", "V3"
  - Create each column first as its own vector, then combine
  - `V1` = 1:10
  - `V2` is a random sample between 1:100
  - `V3` is drawn from a random uniform distribution between 0 and 50 - Use `set.seed(50)`
  - Inspect the `str` and `class` of `m`
  - `dat`, a data.frame built from `V1`, `V2`, `V3`, and `V4`
  - `V4` is a random selection of the letters A-E

```
set.seed(50)
m <- cbind(V1 = 1:10,
           V2 = sample(1:100, size = 10, replace = TRUE),
           V3 = runif(n = 10, min = 0, max = 50))
str(m)
```

```
##  num [1:10, 1:3] 1 2 3 4 5 6 7 8 9 10 ...
##  - attr(*, "dimnames")=List of 2
##   ..$ : NULL
##   ..$ : chr [1:3] "V1" "V2" "V3"
```

```
dat <- data.frame(m, V4 = sample(letters[1:5], size = 10, replace = TRUE))
dat
```

```
##    V1 V2        V3 V4
## 1   1 71 19.527909  d
## 2   2 44 13.488290  a
## 3   3 21 32.044308  d
## 4   4 77  3.877977  e
## 5   5 52 13.864477  b
## 6   6  5 33.805022  d
## 7   7 70 41.768315  c
## 8   8 65 18.228113  b
## 9   9  5  3.706184  b
## 10 10 11  8.446420  b
```

1-d Indexing/subsetting/replacing

- Select the 1st, 2nd, and 10th elements from `a`
- Select the elements of `a` named V1, V2, V3 (use the names)
- Replace the second to last value of `a` with the word "sasquatch"
    - Use code to find the index value, not the actual integer value of the index
- Select from `b` the values "k", "n", "e"
- Identify the index position in `b` of values "k", "n", "e"
- Select the first 5 values of `d` and the last 5 values of `d` into two separate vectors and multiply them.
- Select from `d` all values > 100:
    - How many values are there?
- Select from `d` all values between 90 and 110, and replace them with 100
- Repeat steps 1, 3, 4, and 8 above, but do it by accessing `a`, `b`, and `d` from `l`

2-d Indexing/subsetting/replacing

- Select the first 10 values from `m`, using a single vector and no row or column information
- Use a single vector to select the last row, column value from `m`
- Replace the value selected in 2 above with -99
- Now select row 3, columns 1:2 from `m`, and replace them with their values multiplied by 10
- Do the same, but select the columns by their name, and reset the new values by dividing by 10
- Select from `dat` the values of V3, and square them. Do it using 1) index notation and column name using both 2) `[]`, and 3) `$`
- Subset the first two rows and columns of `dat` into a new data.frame `datss`.
- Replace `dat` rows 1:2, column 1:2 with the values -1:-4
- Reset the part of `dat` you just changed with the values in `datss`

Summarizing datasets

- Calculate the row and column sums of both `m` and `dat`.
- Calculate the overall means and sums of all values in each dataset
- From `dat`, use both the base `aggregate` function and `dplyr` function to calculate the group mean, using `V4` as the grouping variable.

Index into lists

- Select from `l`'s first element the 1st and 5th elements
- Do the same, using the element name and `[[]]` to get the first element of `l`
- Do the same, using `$` and the element name to get the first element
- Do the same, using `[]` and the element name to get the first element
- Select the last element of the last element of `l` (tricky without absolute indexing)

Control structures

Branching

```
a <- 5
if(a > 10) {
  print("Greater than 10!")
} else {
  print("Less than or equal to 10")
}
```

```
## [1] "Less than or equal to 10"
```

Looping

```
b <- 1:3
for(i in b) print(i)
```

```
## [1] 1
## [1] 2
## [1] 3
```

*apply

- A special form of looping
- Intended for *applying* a function to data

```
l2 <- l[c("a", "d")]
lapply(l2, mean)
```

```
## $a
## [1] 8.2
##
## $d
## [1] 100.5661
```

*apply

- Key uses:
  - Return results of loop directly into object
  - Use with anonymous functions to pass an iterator, often into more complex procedures

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

```
## [[1]]
##  V11  V12  V13  V14  V15  V16  V17  V18  V19 V110
##   11    7    9   14    2    5    6    6   13    9
##
## [[2]]
##  [1] 107.79589  75.83848  92.72648  67.46655  94.87043 122.03559 115.11563
##  [8]  95.23533 119.74889 114.82780
```

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

```
## [[1]]
##       V11
## -96.79589
##
## [[2]]
##       V12
## -68.83848
##
## [[3]]
##       V13
## -83.72648
##
## [[4]]
##       V14
## -53.46655
##
## [[5]]
##       V15
## -92.87043
```

- 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`