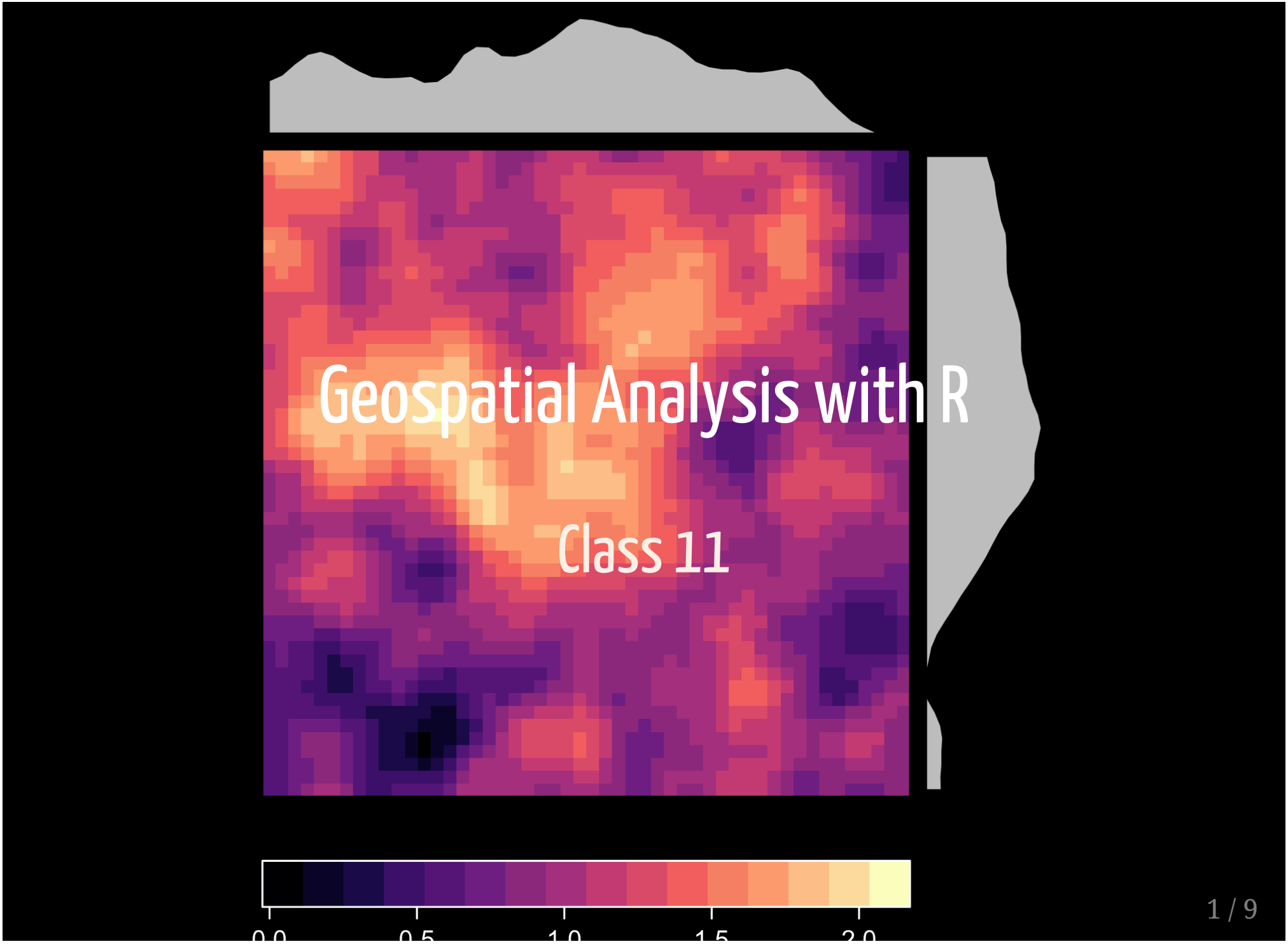
Geospatial Analysis with R
Class 11
0.0
0.5
1.0
1.5
2.0

Today

- Team-based practicals:
  - More control structures (*apply)
  - Data preparation

Practical 1

- *set up:* create a named list `l` made up of three matrices (`m1`, `m2`, `m3`)
  - matrix dimensions: 10 X 3, variables v1:v3: 1:10, `rnorm(10, 100, 20)`, `sample(1:100, 10, replace = TRUE)`; seeds of 1, 2, and 3
- *Write a `for` that*:
  - Iterate over `1:10` and `print`s iterator `i` times 10
  - Iterates over the rows of `l$m1`, printing sum of each row
  - Iterate over each element of `l`, printing first row of each element
- *Write an `lapply` that*:
  - Iterates over `1:10` and returns iterator `i` times 10
  - Applies the `sum` function to each element of `l`
  - Applies the `rowSums` function to each element of `l`
  - Applies the `sum` function to the first row of each element of `l`
- *Repeat previous steps with `sapply`*

```r
set.seed(1)
m1 <- cbind(V1 = 1:10, v2 = rnorm(n = 10, mean = 100, sd = 10),
            v3 = sample(1:100, size = 10, replace = TRUE))
set.seed(2)
m2 <- cbind(V1 = 1:10, v2 = rnorm(n = 10, mean = 100, sd = 10),
            v3 = sample(1:100, size = 10, replace = TRUE))
set.seed(3)
m3 <- cbind(V1 = 1:10, v2 = rnorm(n = 10, mean = 100, sd = 10),
            v3 = sample(1:100, size = 10, replace = TRUE))

for(i in 1:10) print(i * 10)
for(i in 1:nrow(l$m1)) print(sum(l$m1[i, ]))
for(i in 1:length(l)) print(l[[i]][1, ])

lapply(1:10, function(x) x * 10)
lapply(l, sum)
lapply(l, rowSums)
lapply(1:length(l), function(x) sum(l[[x]][1, ]))

sapply(1:10, function(x) x * 10)
sapply(l, sum)
sapply(l, rowSums)
sapply(1:length(l), function(x) sum(l[[x]][1, ]))
```

Extras

- Use an `lapply` to create `l`.
- Use `lapply` to calculate and output `mean` and `sd` of each matrix in `l`. Now do `sapply`
- Add a `data.frame` `d` to `l`. `d` is `m1` and `v4` random sample of a-e
- Use lapply with a conditional to test whether elements of `l` are a matrix. If they are calculate `mean`, `sd`. If they are not (i.e. it's a `data.frame`), then calculate `mean` and `sd` of appropriate columns

```r
seeds <- c(1, 2, 3)  # or 100 * 1:3
l <- lapply(seeds, function(x) {
  set.seed(x)
  m <- cbind(v1 = 1:10, v2 = rnorm(n = 10, mean = 100, sd = 10),
             v3 = sample(1:100, size = 10, replace = TRUE))
})
names(l) <- paste0("m", 1:3)

lapply(l, function(x) c("mu" = mean(x), "sd" = sd(x)))
sapply(l, function(x) c("mu" = mean(x), "sd" = sd(x)))

l$d <- data.frame(l$m1, v4 = sample(letters[1:5], 10, replace = TRUE))

lapply(l, function(x) {
  if(is.matrix(x)) {
    c("mu" = mean(x), "sd" = sd(x))
  } else {
    c("mu" = mean(unlist(x[, 1:3])), "sd" = sd(unlist(x[, 1:3])))
  }
})
```

Practical 2

- Use `readr::read_csv` to read dummy_dataset.csv into `tb_df`
- Determine the unique (distinct) values in the *group* and *element* columns
- Spread `tb_df` so that "Price" and "Element" have their own columns
- Do the same as above, but exclude the *group* variable
- Redo the spread that includes *group*, and then arrange by *group*
- Redo the spread that includes *group*, and then arrange by *group* and by *year*, with *year* in decending order
- Calculate a new column that describes the weight:price ratio

Extras

- Redo the spread that includes *group*, and then arrange by *group* and by *year*, with *year* in decending order, select out the values of group *a*, and calculate the weight:price ratio just for those

```r
tb_df <- readr::read_csv("~/Desktop/dummy_dataset.csv")
tb_df %>% distinct(group, element)
# tb_df %>% distinct(group)
# tb_df %>% distinct(element)
tb_df %>% spread(element, value)
tb_df %>% select(-group) %>% spread(element, value)
tb_df %>% spread(element, value) %>% arrange(group)
tb_df %>% spread(element, value) %>% arrange(group, desc(year))
tb_df %>% spread(element, value) %>% mutate(wt_price = Weight / Price)
# extra
tb_df %>% spread(element, value) %>% arrange(group, desc(year)) %>%
  filter(group == "a") %>% mutate(wt_price = Weight / Price)
```