Lecture 2: Vector and Matrix Operations

Filtering, Recycling, Vectorization

Ingmar Sturm
UCSB
2024-06-25

Special thanks to Robin Liu for select course content used with permission.

Three Important R Techniques

Filtering, Recycling, and Vectorization

Filtering

Recall that we can subset a vector by a logical vector

```
x <- 1:10
subs <- rep(c(F, T), each = 5)
```

Quick quiz: what is the result of `x[subs]`?

Often we want to select values from `x` that satisfy some property. The above can be returned as follows:

```
x[x > 5]
```

```
## [1]  6  7  8  9 10
```

01:00

Filtering

How did this work?

```
x
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10
```

```
x > 5
```

```
##  [1] FALSE FALSE FALSE FALSE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE
```

```
x[x > 5]
```

```
## [1]  6  7  8  9 10
```

We *filtered* `x` based on its values.

Filtering

My gas bill in dollars for every month in order in 2023 is given:

46, 33, 39, 37, 46, 30, 48, 32, 49, 35, 30, and 48.

Create a vector called `gas_bill`.

1. Verify that all months are accounted for (12 entries total).
2. Find the number of months where the gas bill exceeded $40. How does this work?
3. Find the percentage of months where the gas bill exceeded $40.
4. Find the total of how much I paid in gas over all months where the gas bill exceeded $40.

05:00

Recycling

In most cases, two vectors must have the same length in order to be operated on:

```
c(2, 4) + c(6, 3)
```

```
## [1] 8 7
```

For many operations between two vectors, the elements in the shorter vector are *recycled* so that the lengths of the two vectors match.

```
1:4 + c(2, 3)
```

```
## [1] 3 5 5 7
```

```
1:4 + c(2, 3, 2, 3)
```

```
## [1] 3 5 5 7
```

Recycling

A warning may appear when...

```
1:4 + 1:3
```

```
## Warning in 1:4 + 1:3: longer object length is not a multiple of shorter object
## length
```

```
## [1] 2 4 6 5
```

```
1:4 + c(1, 2, 3, 1)
```

```
## [1] 2 4 6 5
```

You are warned against possible mistakes.

Recycling

My gas bill in dollars for every month in order in 2023 is given:

46, 33, 39, 37, 46, 30, 48, 32, 49, 35, 30, and 48.

SoCal Gas overcharged me $3 on odd numbered months (January, March, …) and undercharged me $7 on even numbered months (February, April, …).

1. Update `gas_bill` to reflect my true gas costs in 2023
2. What is the difference between what I paid and what I actually owe?

05:00

Recycling

Adding a vector to a number.

```
x <- 1:10
x + 6
```

```
##  [1]  7  8  9 10 11 12 13 14 15 16
```

Did recycling occur?

Vectorization

A function f is *vectorized* if applying f to a vector $\vec{x} = (x_1, x_2, \ldots, x_n)$ results in a vector of f applied to each element of $\vec{x}$.

$f(\vec{x}) = (f(x_1), f(x_2), \ldots, f(x_n))$.

- Vector in, vector out
- Vectorization is *extremely fast*
- Later we will discuss loops. You should always look for a vectorized solution before reaching for a loop in R.

Many operations in R are already vectorized

```
(x <- (5:10)^2)
```

```
## [1]  25  36  49  64  81 100
```

```
log(x)
```

```
## [1] 3.218876 3.583519 3.891820 4.158883 4.394449 4.605170
```

Vectorization

1. Find the sum of the square roots of every integer from 1 to 1000.
2. Find the product of the natural log of every integer from 100 to 200.
3. Find the sum of the integers between 1 and 100 whose square is between 300 and 500.

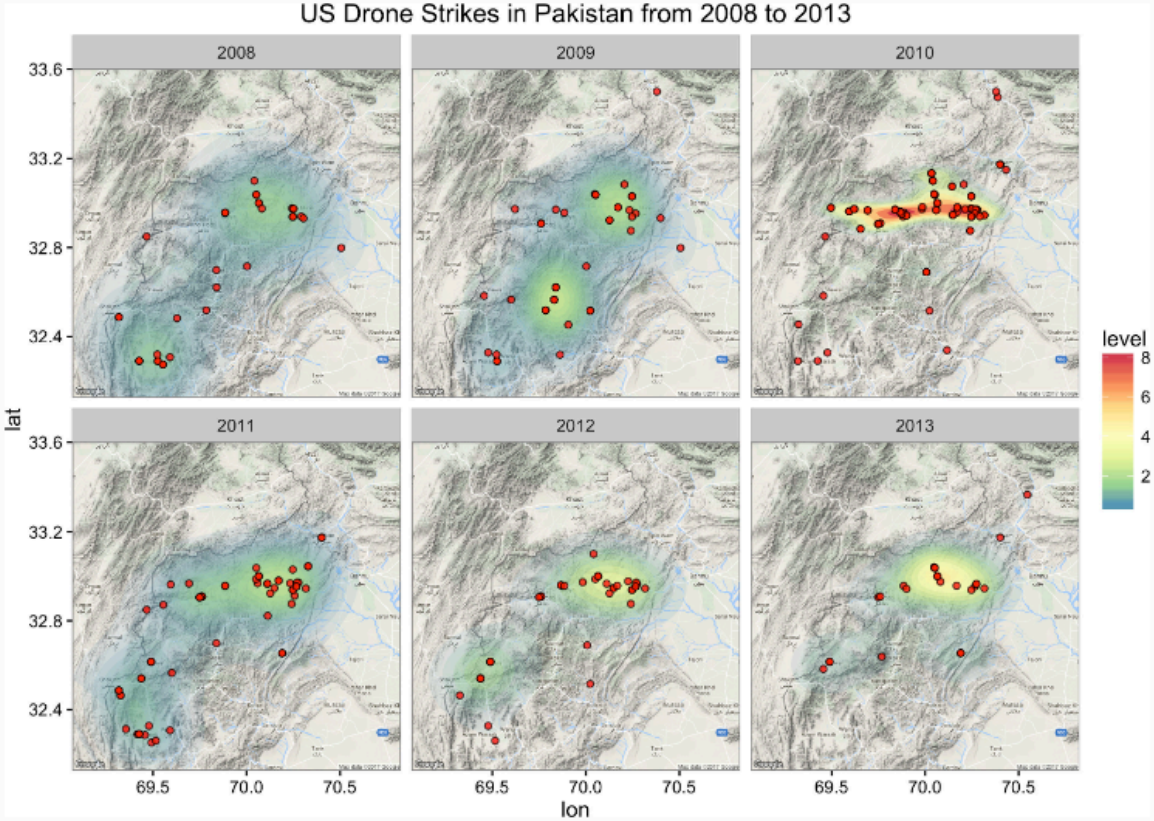
10:00

Matrices, Arrays, and Lists

Data with dimensionality

Data naturally have dimensions

Often times, a vector (1-D) is not enough.



https://trucvietle.me/r/tutorial/2017/01/18/spatial-heat-map-plotting-using-r.html

Matrices

Create a matrix in several ways

```
(x <- seq(-5, 5, 2))
```

```
## [1] -5 -3 -1  1  3  5
```

```
x_mx <- matrix(x, nrow=2, ncol=3)
x_mx
```

```
##      [,1] [,2] [,3]
## [1,]   -5   -1    3
## [2,]   -3    1    5
```

```
rbind(c(-5, -1, 3), c(-3, 1, 5))
```

```
##      [,1] [,2] [,3]
## [1,]   -5   -1    3
## [2,]   -3    1    5
```

```
cbind(c(-5, -3), c(-1, 1), c(3, 5))
```

```
##      [,1] [,2] [,3]
## [1,]   -5   -1    3
## [2,]   -3    1    5
```

Matrices

A matrix is just an atomic vector...

except endowed with extra information: the *dimension*

```
(x <- 1:4)
```

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

```
dim(x)
```

```
## NULL
```

```
is.matrix(x)
```

```
## [1] FALSE
```

```
dim(x) <- c(2, 2)
x
```

```
##      [,1] [,2]
## [1,]    1    3
## [2,]    2    4
```

```
dim(x)
```

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

```
is.matrix(x)
```

```
## [1] TRUE
```

Matrices

Because matrices are just vectors; recycling, filtering, and vectorization rules apply to them.

Subsetting a matrix is different: an index is provided to each dimension.

```
(x <- matrix(1:4, nrow=2, ncol=2))
```

```
##      [,1] [,2]
## [1,]    1    3
## [2,]    2    4
```

```
x[1, 1]
```

```
## [1] 1
```

```
x[2, 1]
```

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

Omitting an index returns all items in that dimension

```
x[, 2]
```

```
## [1] 3 4
```

```
x[1, ]
```

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

Matrices

```
(x <- matrix(1:9, nrow=3, ncol=3))
```

```
##      [,1] [,2] [,3]
## [1,]    1    4    7
## [2,]    2    5    8
## [3,]    3    6    9
```

What is the result of `x[1:2, -3]`?

01:00

Matrices

Create a 4×5 matrix `mx` of integers 1 through 17 (inclusive). Print the matrix.

1. Did filtering, recycling, or vectorization occur?
2. Subset `mx` to create the following matrix:

```
##      [,1] [,2] [,3]
## [1,]    1    5   17
## [2,]    4    8    3
```

1. Set all entries of `mx` greater than 10 to zero. Did filtering, recycling, or vectorization occur?

05:00

Matrices

Like vectors, rows and columns of a matrix can be named.

```
scores <- matrix(c(7, 6, 9, 8, 8, 9, 10, 5, 7), nrow = 3, ncol = 3)
colnames(scores) <- c("Anna", "Joe", "Carl")
rownames(scores) <- c("midterm1", "midterm2", "final")
scores
```

```
##          Anna Joe Carl
## midterm1    7   8   10
## midterm2    6   8    5
## final       9   9    7
```

```
scores["midterm2", -3]
```

```
## Anna  Joe
##    6    8
```

What is the output?

```
scores[scores[,3] > 5, ]
```

00:30

Image manipulation demo

```
install.packages("pixmap")
library(pixmap)
```

```
mtrush1 <- read.pnm("mtrush1.pgm")
str(mtrush1)
```

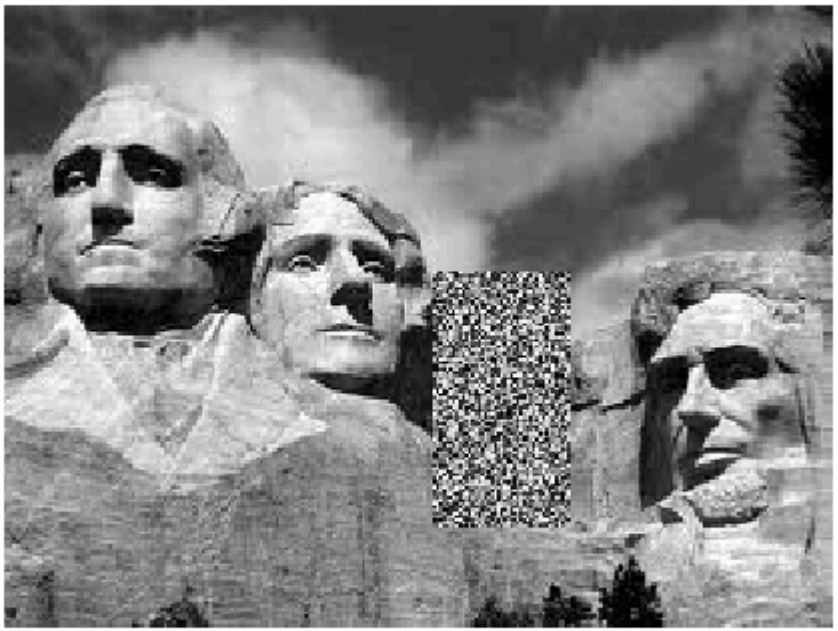
```
## Formal class 'pixmapGrey' [package "pixmap"] with 6 slots
##   ..@ grey    : num [1:194, 1:259] 0.278 0.263 0.239 0.212 0.192 ...
##   ..@ channels: chr "grey"
##   ..@ size    : int [1:2] 194 259
##   ..@ cellres : num [1:2] 1 1
##   ..@ bbox    : num [1:4] 0 0 259 194
##   ..@ bbcent  : logi FALSE
```

Image manipulation demo

`mtrush1@grey` is a matrix containing pixel intensities.

What is the intensity of the pixel at row 70, column 120?

Blur out Teddy Roosevelt's face at $84{:}\,163 \times 135{:}\,177$



Arrays

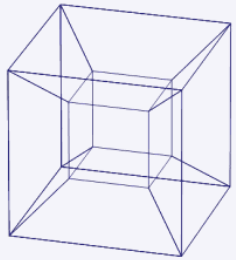
Higher (>2) dimensional vectors are called *arrays*

```
rubiks <- array(1:27, c(3, 3, 3))
dim(rubiks)
```

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

```
hypercube <- array(1:16, c(2, 2, 2, 2))
dim(hypercube)
```

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



Arrays

```
rubiks
```

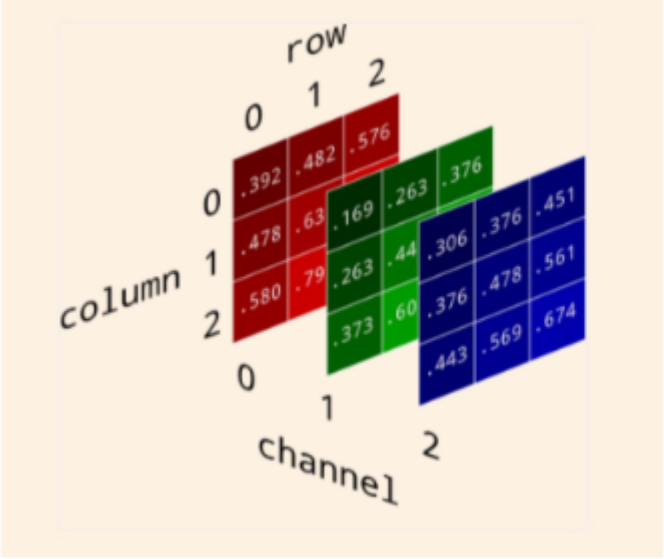
```
## , , 1
##
##      [,1] [,2] [,3]
## [1,]    1    4    7
## [2,]    2    5    8
## [3,]    3    6    9
##
## , , 2
##
##      [,1] [,2] [,3]
## [1,]   10   13   16
## [2,]   11   14   17
## [3,]   12   15   18
##
## , , 3
##
##      [,1] [,2] [,3]
## [1,]   19   22   25
```

```
hypercube
```

```
## , , 1, 1
##
##      [,1] [,2]
## [1,]    1    3
## [2,]    2    4
##
## , , 2, 1
##
##      [,1] [,2]
## [1,]    5    7
## [2,]    6    8
##
## , , 1, 2
##
##      [,1] [,2]
## [1,]    9   11
## [2,]   10   12
##
```

Application of Arrays

Color images are often represented as a 3D array

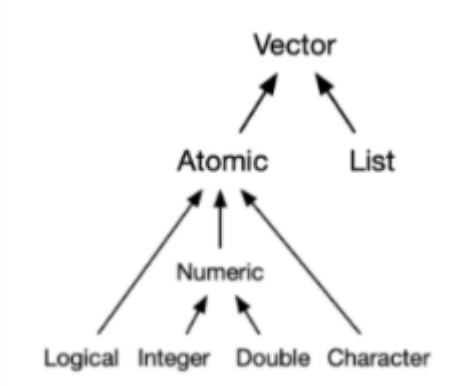


Lists

A **list** is also a vector, but is not atomic.

Usually "vector" means "atomic vector". Recall that atomic vectors can only hold one data type. Lists are also vectors, but can hold many different data types.

For the rest of the class, I will say "vector" to mean "atomic vector" and "list" to mean "list". But in the R literature, lists are vectors.



Lists

```
x <- list(first = 1:10, second = "cat", third = c(T, F))
```

Single brackets `[ ]` return a list while double brackets `[[ ]]` return the element.

```
x[1]
```

```
## $first
##  [1]  1  2  3  4  5  6  7  8  9 10
```

```
x[[1]]
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10
```

Getting an element by name:

```
x$first
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10
```

Lists

Is it possible to create a vector of vectors?

```
c(1:10, 66, 3:7)
```

Checking the structure

- Is `x` an atomic vector? `is.atomic(x)`
- Is `x` a list? `is.list(x)`
- Is `x` a matrix? `is.matrix(x)`
- Is `x` an array? `is.array(x)`

But beware:

```
x <- rbind(1:2, 3:4)
is.atomic(x)
```

```
## [1] TRUE
```

```
is.matrix(x)
```

```
## [1] TRUE
```

```
is.array(x)
```

```
## [1] TRUE
```

Sometimes better to check the *class*

```
class(x)
```

```
## [1] "matrix" "array"
```

AVOID is.vector

`is.vector` does not do what you think it does.

`is.vector` checks for a basic, atomic vector *without additional attributes* (except names), not just any one-dimensional structure.

Summary

- Practice filtering, recycling, and vectorization.
- Practice subsetting matrices.