Command Execution & Programming Intro

Revolution Analytics

1. Intro to Vector Calculation
2. Control Flow
3. Missing, indefinite, and infinite values

Overview

In this session you establish a foundation for R, by getting to know about:

- Calculations on vectors
- Control Flow in R

Outline

1. Intro to Vector Calculation
2. Control Flow
3. Missing, indefinite, and infinite values

Vectors

Already explored what a vector is.

Key question: How do we operate upon them?

How do we process vectors?

Goal Compute a mean-centered variable for `hp` in the mtcars dataset

Traditional Approach: Loops

$$\bar{X} = \frac{1}{N}\sum_{i}^{N} X$$

```
hp.sum <- 0
hp.n <- nrow(mtcars)
for (i in 1:length(mtcars$hp)) {
  hp.sum <- hp.sum + mtcars$hp[i]
}
hp.mean <- hp.sum/hp.n
mtcars$hp.cent <- NA
for (i in 1:length(mtcars$hp)) {
  mtcars$hp.cent[i] <- mtcars$hp[i] - hp.mean
}
```

R approach: Vector calculation

```r
mtcars$hp.cent2 <- mtcars$hp - mean(mtcars$hp)
head(mtcars[c("hp", "hp.cent", "hp.cent2")])
```

```
##                    hp hp.cent hp.cent2
## Mazda RX4         110  -36.69   -36.69
## Mazda RX4 Wag     110  -36.69   -36.69
## Datsun 710         93  -53.69   -53.69
## Hornet 4 Drive    110  -36.69   -36.69
## Hornet Sportabout 175   28.31    28.31
## Valiant           105  -41.69   -41.69
```

Vector calculation

An easy, efficient way to make large calculations in R.

We can do vector operations on columns of `cars`.

Example: Let's find the acceleration that passengers in each car face.

$$a = \frac{(\Delta v)^2}{2d}$$

Example

Calculate the denominator first

```
distance <- cars$dist
```

Instead of multiplying each element by 2 in a loop, multiply the whole vector by 2

```
denominator <- 2 * distance
denominator
```

```
##  [1]   4  20   8  44  32  20  36  52  68  34  56  28  40  48  56  52  68
## [18]  68  92  52  72 120 160  40  52 108  64  80  64  80 100  84 112 152
## [35] 168  72  92 136  64  96 104 112 128 132 108 140 184 186 240 170
```

Example (continued)

Calculate numerator values separately

```
numerator <- -(cars$speed^2)
```

Add both of these values as new columns to cars…

```
cars$numerator <- numerator
cars$denominator <- denominator
```

…and combine them to form a new column

Example (continued)

```
cars$acceleration <- cars$numerator/cars$denominator
```

We can easily do this in one line:

```
cars$acceleration <- -(cars$speed^2)/(2 * cars$dist)
```

REVOLUTION ANALYTICS

Briefly back to dataframes

Use NULL to remove those unnecessary columns:

```
cars$numerator <- NULL
cars$denominator <- NULL
```

In general, NULL can be used to eliminate elements of a list object.

Exercise

Your turn. Use the `mtcars` data set to compute each car's:

- hp-to-wt ratio
- hp per cyl
- Delete the `vs` variable from the dataset

Interim Summary

Vectorized code is easier to read (and debug) and generally faster!

Outline

Control Flow

In the last section, we talked about the `for` loop and how it can be used to execute a command multiple times. This is an example of a concept known as *Control Flow*

Control Flow Specification of the order in which commands are executed

```
help(Control)
```

Control Flow: Additional Contructs

- Looping until some contraint is met `while()`
- Conditional Behavior / Branch (`if() else`)
- Execute a distant set of commands (`subroutine` or `function` execution)
- Others

Additional looping mechanics

Sometimes we don't know how many iterations we want to create.

We may want to run an operation until some criterion is met

`while()` is a good choice.

```
mynum <- 8
while (mynum > 1) {
  print(mynum)
  mynum <- mynum - 1
}
```

Break endlessly repeating loops by pressing 'esc'

Note re: explicit loops

The downside of loops

Loops in R can be “memory hogs” and very slow to run

Loops in R can be difficult to read and track the control flow.

Loops can often be avoided by using:

- Vectorized calculations
- `apply()` family of functions

Control Flow: conditional branching

In addition to performing some operations until a criterion is met, we may want to do some operations under one circumstance, and another operation in others.

if-else statements and while-loops execute by evaluating statements that are either `TRUE` or `FALSE` (i.e. logical values like we saw above)

‘if-else’ statements have a simple structure:

```
If x do task 1...
... else ...
... If y do task 2
```

Logical Operators

Operator	Meaning
==	Equality (equal to)
<	Less than
>	Greater than
<=	Less than or equal
>=	Greater than or equal
!	Negation (logical NOT)
|	Logical OR
&	Logical AND
!=	Inequality (not equal to)

If-else example

```r
if (sample(c(FALSE, TRUE), 1)) {
  print("heads")
} else {
  print("tails")
}
```

Logical operators and vectors

- We previously saw how +, -, *, ˆ, and / can be applied to vectors
- We've seen how we can use Logical operators to evaluate single comparisons

What happens when we want to make *multiple* logical comparisons?

Example

Goal We want to identify which cars are efficient (high `mpg`)

```
mtcars$mpg > median(mtcars$mpg)
```

```
##  [1]  TRUE  TRUE  TRUE  TRUE FALSE FALSE FALSE  TRUE  TRUE FALSE FALSE
## [12] FALSE FALSE FALSE FALSE FALSE FALSE  TRUE  TRUE  TRUE  TRUE FALSE
## [23] FALSE FALSE FALSE  TRUE  TRUE  TRUE FALSE  TRUE FALSE  TRUE
```

Produces a `logical` vector: a vector of `TRUE`/`FALSE` values

Exercise

Your turn:

- Create a logical vector indicating whether a given car has a `hp` over 200

What can we do with logical vectors?

- subset datasets
- assignment
- Control Flow

Subsetting with `logical` vectors

We can use logical operators (and vectors) to identify which values of acceleration are less than -3:

```
cars$acceleration < -3
```

```
##  [1]  TRUE FALSE  TRUE FALSE FALSE  TRUE FALSE FALSE FALSE  TRUE FALSE
## [12]  TRUE  TRUE FALSE FALSE  TRUE FALSE FALSE FALSE  TRUE FALSE FALSE
## [23] FALSE  TRUE  TRUE FALSE  TRUE  TRUE  TRUE  TRUE FALSE  TRUE FALSE
## [34] FALSE FALSE  TRUE  TRUE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
## [45]  TRUE  TRUE  TRUE  TRUE FALSE  TRUE
```

And we can use the resulting logical vector to extract or modify a subset of the data:

```
length(cars$acceleration[cars$acceleration < -3])
```

```
## [1] 28
```

Assignment to subsets of a vector

We found all the cars with acceleration < -3.

What if we thought those were anomalous, and we wanted to replace all values with -3?

```
cars$acceleration[cars$acceleration < -3] <- -3
```

Exercise: Subsetting with logical vectors

Your turn:

- Create a logical vector that is TRUE if dist is greater than 40, and FALSE otherwise.
- Extract the subset of `cars` data corresponding to *only* the cars that have a dist greater than 40.
- Replace all distance values greater than 40 with a value of 40

Vectorized Logic

Situation: You want to label cars in `mtcars` with high mpg “efficient” and cars with low mpg “gas-guzzlers”

Can you do this easily with the if (cond) ifstatements else elsestatements ?

Why not?

```
if (mtcars$mpg > median(mtcars$mpg)) print("efficient") else print("gas-guzzler")
```

```
## [1] "efficient"
```

Vectorized `ifelse()` function

```
ifelse(condition,ifvalue,elsevalue)
```

```
mtcars2 <- mtcars
mtcars2$efficiency <- ifelse(mtcars$mpg > median(mtcars$mpg), "efficient",
  "gas-guzzler")
mtcars2$efficiency
```

```
##  [1] "efficient"   "efficient"   "efficient"   "efficient"   "gas-guzzler"
##  [6] "gas-guzzler" "gas-guzzler" "efficient"   "efficient"   "gas-guzzler"
## [11] "gas-guzzler" "gas-guzzler" "gas-guzzler" "gas-guzzler" "gas-guzzler"
## [16] "gas-guzzler" "gas-guzzler" "efficient"   "efficient"   "efficient"
## [21] "efficient"   "gas-guzzler" "gas-guzzler" "gas-guzzler" "gas-guzzler"
## [26] "efficient"   "efficient"   "efficient"   "gas-guzzler" "efficient"
## [31] "gas-guzzler" "efficient"
```

Similar syntax to ifelse() function in excel

Exercise:

- In mtcars, create a new variable called “type”, and label high hp cars as “highhp” and slow cars as “lowhp”

Summary: if / else and logical vectors

- single-comparison implementation and logical operators
- Logical vectors
- subsetting
- assignment
- vector implementation of `ifelse()`

Outline

NA (not available)

- Missing Values
- test with `is.na()`
- Used as placeholder for initialization

```
x <- 1:3
x[4]
```

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

NaN (not a number)

- 0/0 is NaN
- test with is.nan()
- Beware: is.na(NaN) is true

```
0/0
```

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

Inf and -Inf

- The result of a any number divided by zero.
- R knows how to deal with `Inf` and doesn't throw an error

```
1/0
```

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

Working with NA values

You can explicitly insert NA in your objects

```
x <- c(1, 2, 3, NA, 4, 5)
```

`is.na(x)` returns a vector of logical values. This will be `TRUE` wherever the element is `NA`

```
is.na(x)
```

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

```
x[is.na(x)] <- 3.5
x
```

```
## [1] 1.0 2.0 3.0 3.5 4.0 5.0
```

Lab: Subsetting with logical vectors

Consider the vectors:

```
V1 <- c(1, 2, 3, 4, NA)
V2 <- c(1, NA, 4, 3, 9)
```

- Calculate new vectors `W1` and `W2` such that they contain only the elements that are not `NA` for both `V1` and ‘V2.
- Multiply each element of W1 by its corresponding element in W2

Module review questions

- What is a 'vector calculation' and why is it useful?
- What is the difference between <-, =, != and ==
- What is a 'logical vector,' and how can it be used to subset/modify an object?
- What are the three types of `NA` values, and how do we test for missing values?

Thank you

Revolution Analytics is the leading commercial provider of software and support for the popular open source R statistics language.

www.revolutionanalytics.com
1.855.GET.REVO
Twitter: @RevolutionR