Early warning systems

DEFENSIVE R PROGRAMMING

Dr. Colin Gillespie
Jumping Rivers

Early warning systems

- Avoid problems where possible
- Handle issues as they arise in a sensible way

As an example, using the shortcuts `T` / `F` for `TRUE` / `FALSE`

Problem 1: TRUE and FALSE

- TRUE and FALSE are special values
- We can't override them

```r
TRUE <- 5
```

```
# Error in TRUE <- 5 : invalid (do_set) left-hand side to assignment
```

Problem 2: TRUE and FALSE

Suppose we are working out an F-statistic. It would be natural to have

```
# df is the F-density function
(F <- df(1, 9, 67))
```

```
[1] 0.7798
```

But R treats positive numbers as `TRUE` , so

```
if(F) message("Yer aff yer heid!")
```

```
Yer aff yer heid!
```

`F` is now treated as `TRUE` !

- Get in the habit of using `TRUE` and `FALSE`
  - not `T` and `F`
    - If you testing for `TRUE` , use `isTRUE()`

```
isTRUE(T)
```

```
[1] TRUE
```

```
isTRUE(2)
```

```
[1] FALSE
```

```
T <- 10
isTRUE(T)
```

```
[1] FALSE
```

Let's have a little practice

DEFENSIVE R PROGRAMMING

The message() function

DEFENSIVE R PROGRAMMING

Dr. Colin Gillespie
Jumping Rivers

The message() function

- Signals to the user the state of a process
- This isn't an error - it's just helpful information
- For example, suppose you're running cross-validation, then output could be

```
CV 1 of 10 complete
CV 2 of 10 complete
CV 3 of 10 complete
```

We can turn it off with `suppressMessages()`

```r
noisy = function(a, b) {
   message("I'm doing stuff")
   a + b
}
noisy(1, 2)
```

```
I'm doing stuff
# [1] 3
```

```r
suppressMessages(noisy(1, 2))
```

```
# [1] 3
```

Telling packages to be quiet

- Occasionally, packages can be a bit noisy
- Sometimes loading **ggplot2**, it presents a message
- Don't worry, we can tell it to be quiet

```
suppressPackageStartupMessages(library("ggplot2"))
```

Using message()

The `message()` function is helpful for letting

- you
- and other users

know what's happening.

It's very handy for long running processes

Let's do some work!

DEFENSIVE R PROGRAMMING

The warning() function

DEFENSIVE R PROGRAMMING

Dr. Colin Gillespie
Jumping Rivers

The warning message

The `warning()` function

```r
warning("You have been warned!")
```

```
# Warning message:
# You have been warned!
```

- signals that something may have gone wrong
- R continues (unlike an error)
- "Warning message:" is (pre) appended

Suppress Warnings

Similar to messages, you can suppress warnings via

- `suppressWarnings()``

This is almost never a good idea

- Fix the underlying problem!

When should you use a warning?

A good use of warning

Suppose we're performing regression on

```
d = data.frame(y = 1:4, x1 = 1:4)
d$x2 = d$x1 + 1
```

So `x2 = x1 + 1`

When we fit a multiple linear regression model

```
m = lm(y ~ x1 + x2, data = d)
summary(m)
```

```
# Some output removed
# Warning message:
# In summary.lm(m) : essentially perfect fit: summary may be unreliable
```

Your turn

DEFENSIVE R PROGRAMMING

Stop (right now)

DEFENSIVE R PROGRAMMING

Dr. Colin Gillespie
Jumping Rivers

I saw the sign

Sometimes things are just broken

- We need to raise an error

For example:

```
1 + "stuff"
```

```
Error in 1 + "stuff": non-numeric argument to binary operator
```

Stop right now thank you very much

To raise an error, we use the `stop()` function

```r
stop("A Euro 1996 error has occurred")
```

```
# Error: A Euro 1996 error has occurred
```

```r
conf_int <- function(mean, std_dev) {
   if(std_dev <= 0)
       stop("Standard deviation must be positive")

   c(mean - 1.96 * std_dev, mean - 1.96 * std_dev)
}
```

Catch em while you can

- You can't suppress (or ignore) errors
  - The definition of an error is that R can't continue
  - Instead, we catch errors

The try() function

The `try()` function acts a bit like `suppress()`

```
res <- try("Scotland" + "World cup", silent = TRUE)
```

It tries to execute something, if it doesn't work, it moves on

The try() idiom

```r
res <- try("Scotland" + "World cup", silent = TRUE)
res
```

```
[1] 'Error in "Scotland" + "World cup": non-numeric argument to binary operator'
attr(,"class")
[1] "try-error"
attr(,"condition")
<simpleError in "Scotland" + "World cup":
                            non-numeric argument to binary operator>
```

The try() idiom

```r
result <- try("Scotland" + "World cup", silent = TRUE)
class(result)
```

```
[1] "try-error"
```

```r
if(class(result) == "try-error")
  ## Do something useful
```

Let's practice

DEFENSIVE R PROGRAMMING