Lazy Evaluation

Data Analysis with R and Python

Deepayan Sarkar

Lazy Evaluation

- Consider the following function

```
choose1 <- function(u, a, b) {
    if (u < 0.5) a else b
}
```

Lazy Evaluation

- Chooses and returns one of two arguments depending on a third argument

Lazy Evaluation

- Chooses and returns one of two arguments depending on a third argument
- Could be used for treatment randomization

```
choose1(rbinom(1, size = 1, prob = 0.3), "placebo", "treatment")
```

```
[1] "placebo"
```

```
choose1(rbinom(1, size = 1, prob = 0.3), "placebo", "treatment")
```

```
[1] "placebo"
```

```
choose1(rbinom(1, size = 1, prob = 0.3), "placebo", "treatment")
```

```
[1] "placebo"
```

```
choose1(rbinom(1, size = 1, prob = 0.3), "placebo", "treatment")
```

```
[1] "treatment"
```

Lazy Evaluation

- How does this function work?

Lazy Evaluation

- How does this function work?
- Maybe
  - The function is called with some arguments
  - R matches and evaluates the arguments `u`, `a`, and `b`
  - Body of function executed with these values

Lazy Evaluation

- How does this function work?
- Maybe
  - The function is called with some arguments
  - R matches and evaluates the arguments `u`, `a`, and `b`
  - Body of function executed with these values
- Does this sound reasonable?
- How can we check? Code demo
- What happens in Python?

Non-standard Evaluation

- Suppose we have the following problem:
  - Find all subsets of size 2 from a set of size 5

Non-standard Evaluation

- Easy to create all *ordered* pairs using the `expand.grid()` function

```
g <- expand.grid(a = 1:5, b = 1:5)
g
```

```
   a b
1  1 1
2  2 1
3  3 1
4  4 1
5  5 1
6  1 2
7  2 2
8  3 2
9  4 2
10 5 2
11 1 3
12 2 3
13 3 3
14 4 3
```

Non-standard Evaluation

- But we want to retain only the $\binom{5}{2}$ distinct combinations of size 2
- Also not difficult:

```
g$b > g$a            # find the rows for which $a > b$
```

```
 [1] FALSE FALSE FALSE FALSE FALSE  TRUE FALSE FALSE FALSE FALSE  TRUE  TRUE FALSE FALSE
[15] FALSE  TRUE  TRUE  TRUE FALSE FALSE  TRUE  TRUE  TRUE  TRUE FALSE
```

Non-standard Evaluation

- But we want to retain only the $\binom{5}{2}$ distinct combinations of size 2
- Also not difficult:

```
g$b > g$a            # find the rows for which $a > b$
```

```
 [1] FALSE FALSE FALSE FALSE FALSE  TRUE FALSE FALSE FALSE FALSE  TRUE  TRUE FALSE FALSE
[15] FALSE  TRUE  TRUE  TRUE FALSE FALSE  TRUE  TRUE  TRUE  TRUE FALSE
```

```
g[g$b > g$a, ]       # Then use this as a row index
```

```
   a b
6  1 2
11 1 3
12 2 3
16 1 4
17 2 4
18 3 4
21 1 5
22 2 5
23 3 5
24 4 5
```

Non-standard Evaluation

- As noted before, referring to `g` multiple times not ideal
- We can avoid doing this using the `eval()` function

Non-standard Evaluation

- We can replace

```
g$b > g$a
```

```
 [1] FALSE FALSE FALSE FALSE FALSE  TRUE FALSE FALSE FALSE FALSE  TRUE  TRUE FALSE FALSE
[15] FALSE  TRUE  TRUE  TRUE FALSE FALSE  TRUE  TRUE  TRUE  TRUE FALSE
```

- by

```
eval(quote(b > a), g)
```

```
 [1] FALSE FALSE FALSE FALSE FALSE  TRUE FALSE FALSE FALSE FALSE  TRUE  TRUE FALSE FALSE
[15] FALSE  TRUE  TRUE  TRUE FALSE FALSE  TRUE  TRUE  TRUE  TRUE FALSE
```

Non-standard Evaluation

- We can replace

```
g$b > g$a
```

```
 [1] FALSE FALSE FALSE FALSE FALSE  TRUE FALSE FALSE FALSE FALSE  TRUE  TRUE FALSE FALSE
[15] FALSE  TRUE  TRUE  TRUE FALSE FALSE  TRUE  TRUE  TRUE  TRUE FALSE
```

- by

```
eval(quote(b > a), g)
```

```
 [1] FALSE FALSE FALSE FALSE FALSE  TRUE FALSE FALSE FALSE FALSE  TRUE  TRUE FALSE FALSE
[15] FALSE  TRUE  TRUE  TRUE FALSE FALSE  TRUE  TRUE  TRUE  TRUE FALSE
```

- A general function that makes this even simpler is the `with()` function

```
with(g, b > a)
```

```
 [1] FALSE FALSE FALSE FALSE FALSE  TRUE FALSE FALSE FALSE FALSE  TRUE  TRUE FALSE FALSE
[15] FALSE  TRUE  TRUE  TRUE FALSE FALSE  TRUE  TRUE  TRUE  TRUE FALSE
```

Non-standard Evaluation

- We might guess that this is done using `substitute()`

Non-standard Evaluation

- We might guess that this is done using `substitute()`
- We can verify this by looking at the `with` function by typing its name

```
with
```

```
function (data, expr, ...)
UseMethod("with")
<bytecode: 0x7f8779759448>
<environment: namespace:base>
```

Non-standard Evaluation

- Actually this just tells us that `with()` is a *generic* function
- The *method* we want is the default method

```
getS3method("with", "default")
```

```
function (data, expr, ...)
eval(substitute(expr), data, enclos = parent.frame())
<bytecode: 0x7f8779758aa8>
<environment: namespace:base>
```

Non-standard Evaluation

- All of this can now be combined to get all the $\binom{5}{2}$ distinct 2-subsets

```
g[with(g, b > a), ]
```

```
   a b
6  1 2
11 1 3
12 2 3
16 1 4
17 2 4
18 3 4
21 1 5
22 2 5
23 3 5
24 4 5
```

Non-standard Evaluation

- This is a very common use case: obtaining a subset a dataset
- There is an even simpler way using the `subset()` function

```
subset(g, b > a)
```

```
   a b
6  1 2
11 1 3
12 2 3
16 1 4
17 2 4
18 3 4
21 1 5
22 2 5
23 3 5
24 4 5
```

Non-standard Evaluation

- A popular add-on package **dplyr** has a similar function called `filter()`

```
dplyr::filter(g, b > a)
```

```
   a b
1  1 2
2  1 3
3  2 3
4  1 4
5  2 4
6  3 4
7  1 5
8  2 5
9  3 5
10 4 5
```

Non-standard Evaluation

- A popular add-on package **dplyr** has a similar function called `filter()`

```
dplyr::filter(g, b > a)
```

```
   a b
1  1 2
2  1 3
3  2 3
4  1 4
5  2 4
6  3 4
7  1 5
8  2 5
9  3 5
10 4 5
```

- Here `dplyr::filter` is the *namespace* notation `package::function`
- Allows us to use the function without attaching the whole package
- Similar to the `.` accessor in Python

Non-standard Evaluation

- The **dplyr** package is very useful for routine data manipulation
- We will discuss it again later
- The idea of quoted expressions and *non-standard evaluation* is essential in many other contexts