Defensive Programming

Nick Eubank
CSDI, Vanderbilt University

January 26, 2018

Goals

Goals

1. Introduce principles of *Defensive Programming*

Goals

1. Introduce principles of *Defensive Programming*
2. Learn four specific “best practices”
    - Use tests
    - Don’t duplicate information
    - Don’t transcribe, export
    - Use good style

Defensive programming

Defensive programming

Philosophy of writing code

Defensive programming

Philosophy of writing code motivated by the simple proposition:

Defensive programming

Philosophy of writing code motivated by the simple proposition:

People are bad at writing code

Defensive programming

Philosophy of writing code motivated by the simple proposition:

People are bad at writing code

If we want to avoid errors,

Defensive programming

Philosophy of writing code motivated by the simple proposition:

People are bad at writing code

If we want to avoid errors, not enough to “just be careful.”

Defensive programming

Philosophy of writing code motivated by the simple proposition:

People are bad at writing code

If we want to avoid errors, not enough to "just be careful."

$\Rightarrow$ Need strategies take take our fallibility into account

Defensive programming

Set of best practices designed to:

Defensive programming

Set of best practices designed to:

1. Minimize opportunities for errors to enter code

Defensive programming

Set of best practices designed to:

1. Minimize opportunities for errors to enter code
2. Maximize the probability that *when* we commit errors, we catch them quickly

Do I need defensive programming?

Do I need defensive programming?

YES.

Do I need defensive programming?

"To Err is Human"

Do I need defensive programming?

"To Err is Human"

- *Among professional programmers*, average error rate is 10 - 50 bugs per 1,000 lines of delivered code

  Steve McConnell, 1993

"Bugs" $\not\Rightarrow$ syntax errors

Do I need defensive programming?

QJPS Replication Review:

- Before publication, test whether replication packages run and generate results in the paper.

Do I need defensive programming?

QJPS Replication Review:

- Before publication, test whether replication packages run and generate results in the paper.

From 2012 - 2016:

Do I need defensive programming?

QJPS Replication Review:

- Before publication, test whether replication packages run and generate results in the paper.

From 2012 - 2016:

- 4 packages passed without modifications

Do I need defensive programming?

QJPS Replication Review:

- Before publication, test whether replication packages run and generate results in the paper.

From 2012 - 2016:

- 4 packages passed without modifications
- *58%* of packages generated results that were different from those in the paper.

Do I need defensive programming?

Do I need defensive programming?

1. If a correlation exists between sophistication of analysis and likelihood of errors, it is if anything positive.
    - Senior, junior, fancy, basic: all had problems!

Do I need defensive programming?

1. If a correlation exists between sophistication of analysis and likelihood of errors, it is if anything positive.
    - Senior, junior, fancy, basic: all had problems!
2. Even if you trust yourself, do you trust your coauthors?

Do I need defensive programming?

1. If a correlation exists between sophistication of analysis and likelihood of errors, it is if anything positive.
    - Senior, junior, fancy, basic: all had problems!
2. Even if you trust yourself, do you trust your coauthors?
    - (Do you trust the version of you that wrote that code at 3am?))

Do I need defensive programming?

1. If a correlation exists between sophistication of analysis and likelihood of errors, it is if anything positive.
    - Senior, junior, fancy, basic: all had problems!
2. Even if you trust yourself, do you trust your coauthors?
    - (Do you trust the version of you that wrote that code at 3am?))
3. Do you trust the people who made the dataset you're using?

Do I need defensive programming?

1. If a correlation exists between sophistication of analysis and likelihood of errors, it is if anything positive.
    - Senior, junior, fancy, basic: all had problems!
2. Even if you trust yourself, do you trust your coauthors?
    - (Do you trust the version of you that wrote that code at 3am?))
3. Do you trust the people who made the dataset you're using?
    - If you estimate the share of a population that's female, and someone left a `7` in the `female` variable, if you don't catch it, that means your answer is *wrong*.

Defensive programming

Set of best practices designed to:

1. Minimize opportunities for errors to enter code
2. Maximize the probability that *when* we commit errors, we catch them quickly

Four Skills

Write tests

Don’t duplicate information

Don’t transcribe, export

Use good style

Four Skills

Write tests

Don’t duplicate information

Don’t transcribe, export

Use good style

Tests

Lines of code that assert something about the data

- Evaluate to `True` or `False`

Tests

Lines of code that assert something about the data

- Evaluate to `True` or `False`

e.g.

```
df = read.csv('state_populations.csv')
stopifnot( nrow(df) == 50 )
```

But I “test” informally...

Value of tests:

But I “test” informally...

Value of tests:

1. Explicit form of checking data is doing what you expect

But I “test” informally...

Value of tests:

1. Explicit form of checking data is doing what you expect
2. Unlike just looking at result interactively, executes every time you run your code

But I “test” informally...

Value of tests:

1. Explicit form of checking data is doing what you expect
2. Unlike just looking at result interactively, executes every time you run your code
    - If you or co-author change upstream data or code and introduce a mistake, tests will catch.

Writing tests: R

Is age always positive?

This will pass (do nothing):

```r
age = c(42, 20, 31, 18)
# Make sure age is positive:
stopifnot( age > 0 )
```

Writing tests: R

Is age always positive?

This will pass (do nothing):

```
age = c(42, 20, 31, 18)
# Make sure age is positive:
stopifnot( age > 0 )
```

But if, for example, “missing” was coded as -99, this would throw an error:

```
age = c(42, 20, 31, -99)
stopifnot( age > 0 )
```

Writing tests: R

For vectors, `stopifnot` checks if ALL values are TRUE.

This will fail:

```
# Are all values True?
v = c(1, 2, 3)
stopifnot( v == 2 )
```

This will pass:

```
stopifnot( v > 0 )
```

Writing tests: R

If you want to see if test holds for at least *some* observations, use `any`.

```
# Are there at least some values that are 2?
v = c(1, 2, 3)
stopifnot( any(v == 2) )
```

This will pass.

Writing tests

Can combine with functions:

```
stopifnot( length(VECTOR) == 100 )
```

Writing tests: Stata

Is age always positive?

```
assert age > 0
```

Writing tests: Stata

Is age always positive?

```
assert age > 0
```

50 States in data?

```
count
assert r(N) == 50
```

How to write tests

Go to:

Tests: Exercise 1

Setup:

Tests: Exercise 1

Setup:

1. Download `exercises` folder

Tests: Exercise 1

Setup:

1. Download `exercises` folder
2. Open `starter_code.R` or `starter_code.do`, set the working directory to the `exercises` folder.

Exercises:

Tests: Exercise 1

Setup:

1. Download `exercises` folder
2. Open `starter_code.R` or `starter_code.do`, set the working directory to the `exercises` folder.

Exercises:

1. The World Development Indicator (WDI) data has duplicate entries. Write a test that fails because there shouldn't be duplicates.

Tests: Exercise 1

Setup:

1. Download `exercises` folder
2. Open `starter_code.R` or `starter_code.do`, set the working directory to the `exercises` folder.

Exercises:

1. The World Development Indicator (WDI) data has duplicate entries. Write a test that fails because there shouldn't be duplicates.
2. The countries in Polity should be a perfect subset of the countries in the WDI dataset, but they are not. Write a test that fails because of this.

Tests: When to write them

Tests: When to write them

- **After merges** No where are problems with data made more clear then in a merge. ALWAYS add tests after a merge!

Tests: When to write them

- **After merges** No where are problems with data made more clear then in a merge. ALWAYS add tests after a merge!
- **After complicated manipulations** If you had to think about it, you should test to do it.

Tests: When to write them

- **After merges** No where are problems with data made more clear then in a merge. ALWAYS add tests after a merge!
- **After complicated manipulations** If you had to think about it, you should test to do it.
- **Before dropping observations**

Tests: When to write them

- **After merges** No where are problems with data made more clear then in a merge. ALWAYS add tests after a merge!
- **After complicated manipulations** If you had to think about it, you should test to do it.
- **Before dropping observations**

Most of use check things interactively to make sure we did it right. A good rule of thumb is that when you catch yourself checking something interactively, stop and write it as a test.

Four Skills

Write tests

Don’t duplicate information

Don’t transcribe, export

Use good style

Don’t duplicate information

Don’t duplicate information

- If information is represented in many places, when you make changes you have to find all those places.

Don’t duplicate information

- If information is represented in many places, when you make changes you have to find all those places.
- If information is represented once, and everything else points back to that representation, one change will *always* change everything.

Don’t duplicate information

Duplicated information:

```
df$var1 <- gsub("armadillo", "Mr. Armadillo",
                df$var1)
df$var2 <- gsub("armadillo", "Mr. Armadillo",
                df$var2)

...
[Other manipulations]
...

df$var7 <- gsub("armadillo", "Mr. Armadillo",
                df$var7)
```

Don’t duplicate information

One representation:

```
pre_change_name <- "armadillo"
replacement_name <- "Mr. Armadillo"

df$var1 <- gsub(pre_change_name,
                replacement_name, df$var1)
df$var2 <- gsub(pre_change_name,
                replacement_name, df$var2)
df$var7 <- gsub(pre_change_name,
                replacement_name, df$var7)
```

Don’t duplicate information

One representation:

```
pre_change_name <- "armadillo"
replacement_name <- "Dr. Armadillo"

df$var1 <- gsub(pre_change_name,
                replacement_name, df$var1)
df$var2 <- gsub(pre_change_name,
                replacement_name, df$var2)
df$var7 <- gsub(pre_change_name,
                replacement_name, df$var7)
```

Don’t duplicate information

```
replace var1 = "Mr. Armadillo" ///
               if var1 == "armadillo"
replace var2 = "Mr. Armadillo" ///
               if var2 == "armadillo"
replace var7 = "Mr. Armadillo" ///
               if var3 == "armadillo"
```

Don’t duplicate information

```
local pre_change_name = "armadillo"
local replacement_name = "Mr. Armadillo"
replace var1 = "`replacement_name'" ///
               if var1 == "`pre_change_name'"
replace var2 = "`replacement_name'" ///
               if var2 == "`pre_change_name'"
replace var7 = "`replacement_name'" ///
               if var3 == "`pre_change_name'"
```

Don’t duplicate information

```
local pre_change_name = "armadillo"
local replacement_name = "Mr. Armadillo"
replace var1 = "`replacement_name'" ///
               if var1 == "`pre_change_name'"
replace var2 = "`replacement_name'" ///
               if var2 == "`pre_change_name'"
replace var7 = "`replacement_name'" ///
               if var3 == "`pre_change_name'"
```

(Or write as loop)

Don’t Duplicate: Exercise 2

Exercises:

- All the regressions in the code you have use the same base specification. Consolidate the representation of that base specification.
- Now add `population` as a control to all you regressions. You should only have to add it once!

Four Skills

Write tests

Don’t duplicate information

Don’t transcribe, export

Use good style

Don’t transcribe results!

Number one reason papers don’t match real results at QJPS.

R:

- `stargazer`
- Tutorial: `http://jakeruss.com/cheatsheets/stargazer.html`
- Custom: `http://stanford.edu/~ejdemyr/r-tutorials/tables-in-r/`

Stata:

- Summary: `http://www.nickeubank.com/exporting-results-stata-latex/`

Use for numbers in your text as well!

Don’t transcribe results: Exercise 3

Oh man, our tests found all these problems. Ugh, why did we put all those old results in by hand?! Now we have to copy them again. Or... we could make the automatically updating!

1) Export the regression table at the end of `starter_code.R` using stargazer.

2) Open our latex analysis file (`start_latex.tex`).

3) Import it into your latex document using the `input{}` command.

Don’t transcribe results: Exercise 3

Make tables:

```
stargazer(list(model1, model2, model3, model4),
          title="TITLE HERE",
          type="latex",
          out='FILE_NAME.tex' )
```

```
eststo clear
eststo: reg polity gdp_per_cap
eststo: reg polity gdp_per_cap under5_mortality
esttab using your_file_name.tex, replace ///
       title("My regressions!")
```

Latex Import:

```
input{your_file_name.tex}
```

Don’t transcribe: Exercise 2 (Part 2)

Now, in the text, we say that households have an average size of 8, but we know that’s wrong. Can you export the average size of the household from R and import it into LaTeX? Hint: Here’s how you write a number to a file as text:

```
x = 1/3
x_as_string = format(x, digits=2)
write(x_as_string, "my_file.tex")
```

Four Skills

Write tests

Don’t duplicate information

Don’t transcribe, export

Use good style

Style

Style isn't just about aesthetics; it's about making code readable so errors are easy to see.

- Whitespace is free; use it.
- Use informative variable names (not x, y)
- COMMENT

Style

Bad:

```
df=read.csv('file_90823409.csv')
df$var07=df$var07+11
df=df[df$var07>65]
lm("pid~var09")
```

Style

```
# Load voter survey data from 2007
voters = read.csv('file_90823409.csv')

# var07 is age in 2007. Want age today (2018)
voters$age = voters$var07 + 11

# Let's see how income predicts voterid for
# people over 65 today, so subset to
# people over 65 today and regress.

voters = voters[df$age>65]
voters$income = df$var09

lm("pid ~ income")
```

Thanks!

Resources: `www.nickeubank.com/replication`