STAT 33A Lecture – April 27

Topics:

- Code Style & Organization
- Preventing Bugs
  - Defensive Programming
  - R Pitfalls
- Debugging
- What Now?

Code Style & Organization

Why does code style matter?

A clear, consistent style makes it easier to:

- Spot and fix bugs
- Revisit or reuse code
- Share code with others
  - Do you need online help?
  - Do you work on a team?
- Write code (fewer decisions to make)

Detailed guide at style.tidyverse.org

Use Spaces

Put 1 space:

- After commas
  - `sum(3, 4)`
- Before and after infix operators
  - `a + b`
- Before parenthesis in `if`, `for`, `while`, and `return`
  - `if (condition)`
  - `for (i in 1:10)`

Use Indentation

Keywords (like `if` and `function`) begin a block of code.

- Put opening braces `{` on the same line as the keyword.
- Indent each line inside by 2 spaces.
- Put closing braces `}` on their own line, or before `else`.

Okay to skip curly braces `{ }` if there's only 1 line inside.

Write Paragraphs

Break down problems into **paragraphs**:

- A single step in the larger computation
- Usually less than 10 lines of code

Put a blank line between paragraphs.

Turn repeated paragraphs into functions.

Use Comments

Most comments should explain **why** rather than what or how.

Also use comments to:

- Plan your strategy before writing the code
- Summarize the purpose of a paragraph of code
- Document how to use the code
    - See the roxygen2 package

Use Meaningful Names

Choose concise and descriptive names.

- Nouns for variable names
  - Exception: `i`, `j`, `k` in loops
- Verbs for function names
  - Exception: mathematical functions such as `mean()`

Choose a naming style and stick to it:

- `lowerCamelCase`
- `snake_case`

Preventing Bugs

Defensive Programming

Pay attention to warnings!

Test each line, paragraph, and function you write.

In functions, add code to check assumptions about inputs:

- `is()` to check data type
- `length()` and `dim()` to check dimensions
- `stop()` to print an error message and stop evaluation

R Pitfalls

R (and programming) can be counterintuitive.

See The R Inferno [link], a humorous catalog of common R pitfalls.

Debugging

Debugging

The process of confirming, step-by-step, that what you believe the code does is actually what the code does.

How?

- Work forward through the code from the beginning.
- Work backward from the source of an error.

R has built-in functions to help!

What Now?

What Now?

- **DATA 8**: For everyone. Basic statistics skills. A chance to learn Python.
- **DATA 100**: If you want a data science career. Round out your skill set: databases, visualizations, modern statistical methods.
- **STAT 133**: If you want to know more about (how statisticians use) R.

You should *not* take STAT 33B — too much overlap with this class.