Best Practices for Code Reviews

Goals:

1. For the presenter to receive feedback on their code

2. For all participants (regardless of skill level) to engage

Presenter:

1. Sign up

2. Select your code

a. That you are willing to receive and incorporate feedback on

b. That is digestible in an hour (<100 lines)

c. Use “vanilla” R to check for dependencies on a “fresh” environment

d. Prepare a small example dataset

3. Use slack to communicate the week in advance

a. Provide code chunk

b. Provide instructions for appropriate dependencies

c. Provide use case (where available/appropriate)

d. Specify goals & expectations

4. Check your attitude

a. Be humble

b. Be willing to incorporate feedback

c. Ask clarifying questions

5. Lead the discussion

a. Have an action plan

b. Balance expertise in groups

c. Focus the conversation

d. Attempt to purposefully engage early learners

e. Avoid letting (senior) individuals dominate

Participants:

1. Prepare

a. Review the code in advance

b. Google what you don’t understand

2. Check your attitude

a. Be humble

b. Remember that this isn’t a competition

3. Participate

a. Focus on the speaker’s goals

b. Avoid distractions and engage with the presenter

c. Allow others to contribute

d. Ask clarifying questions and note *why* you need the clarification

e. Give feedback & coding suggestions at a novice explanation level

f. Give compliments

Planning a Code Review: (Nick’s Learned Lessons)

* 1. Set specific goals for each code review -
     1. Try to be as explicit and clear as possible for what you are asking the participants to do
        1. I have tried to be (what felt to me) painfully clear but I still was falling short making it clear to some people.
        2. People will always tend to notice or ask about things that are not the focus of the exercise, being specific about the goal may help. Also setting up a simplified scenario can be helpful (although requires much more prep time)
  2. Practice exercise
     1. When in doubt try it out!
        1. Run it by someone prior to lab meeting to make sure your request/goal is understandable
     2. Running on different systems will inevitably run into issues
        1. Send out basic script ahead of time to try the dependencies/functions of focus
  3. Create a minimal working example
     1. Working directly with your own raw code can be challenging
        1. It is likely filled with code specific to you/your project
        2. It can take a lot of time to explain details and working of code
        3. Code can have multiple steps that are needed to run to get to script focus
        4. Using a whole script opens up the risk of getting off topic
  4. Use think/pair/share
     1. Pair to mix up experience
     2. Gives individuals time to think on their own, discuss their ideas and learn from each other and then present unique idea to all, instead of getting one answer to a question
     3. This allows pairing of different experiences and greater exchange of ideas
  5. Keep It Simple Silly (KISS)
     1. There is never enough time
        1. I typically have multiple activities planned, but rarely get through them all
        2. Modularize the activity so that each can be its own and if accomplished you are able to move onto the next
     2. One activity per code review is typically sufficient
        1. This will take up most of the time
           1. Intro - 10 min
           2. Think/pair - 30 min
           3. Share in group - 10 min
           4. Debrief - 5 min
        2. I have not been successful in trying to do more than that in different ways because it is going to be taking time away from one of the other steps
  6. Participants value (based on feedback)
     1. Group work
        1. Helpful to write code, see others approach or work through problem
        2. Smaller groups 2-3 to make sure everyone is able to talk
     2. Short exercises (not rushed)
     3. Help/guidance
        1. Clear explanation of the problem, code, and goal
        2. Hints or pointing towards resources for those new to specific task
     4. Tasks
        1. Short (not rushed)
        2. Simplified (so focus can be on goal and not figuring things out)
        3. If complex task, split into digestible parts