Reflective Assessment Example Lesson Plan

Matthew J. Lavin

Assignment Summary

This reflective assignment is designed to help you develop your understanding of reproducibility by identifying whether specific reproducibility or replicability practices are present in a block of R code. In class, you will work with one or more students to download the code and data for the assignment, run various code files using RStudio, and reflect on what happens. After each step of the assignment, you and your teammates will report to the rest of the class and discuss how your results compare with other groups'.

Included Files

	File/ Folder	**Description**
1.	activity.R	snippets of R code with errors deliberately inserted
2.	working_code.R	snippets of R code with no errors (but room for improvement)
3.	refactored.rmd	refactored code to demonstrate potential areas for revision (in code notebook / markdown-based format)
4.	data/cbus_listings.csv	airbnb listings data (scraped for a Denison assignment)
5.	data/cbus_landmarks.csv	Columbus landmark and location data to merge with listings
6.	utils/dv.R	modified Vincenty Distance function to support use with a mutate function

Detailed Instructions

Part I: Setup

1. Open RStudio
2. Click "File > New Project"
3. Click "Version Control"
4. Click "git"
5. Under "Repository URL," enter the Github URL for the code repository
6. Click "Create Project"

Part II: Discuss with Your Team

1. Open `activity.R` and try to run each code cell. If you get an error, try to resolve it, or comment out the line of code that errors and keep going until you can't get any further.

2. Take turns with your partner saying what you think each line of code does (rubber ducking). Try to be as precise as you can.
3. Once you have finished resolving or commenting out each code chunk, turn to your teammates and discuss what happened.
4. Decide on 1-2 questions or takeaways you want to share with the class (where you got stuck, error fixes, debugging strategies you tried, etc.). Make a plan for reporting to the class.
5. Share your experiences with your classmates and listen to their results. Take notes on any key questions or takeaways that seem to come up more than once.

Part III: Compare and Contrast

1. Open the file `working_code.R` and see if you can run it without errors. (Any errors you encounter here should be solvable.)
2. Discuss the differences you see between `activity.R`, and `working_code.R`.
3. Decide on 1-2 questions or takeaways you want to share with the class (key differences, areas for additional improvement, etc.). Make a plan for reporting to the class.
4. Share your experiences with your classmates and listen to their results. Take notes on any key questions or takeaways that seem to come up more than once.

Part IV: Literate Programming

1. Open the file `refactored.Rmd` in RStudio and try knitting it to HTML.
2. Discuss with your teammates how the code is now organized. Discuss formatting, organization, etc.

3. Decide on 1-2 questions or takeaways you want to share with the class (key ideas, areas for additional improvement, etc.). Make a plan for reporting to the class.
4. Share your experiences with your classmates and listen to their results. Take notes on any key questions or takeaways that seem to come up more than once.

Instructor Notes

I. Setup

This directions above are written assuming that the instructor will create a public Github repository for the code and data files for the assignment. Part I, no. 5 can be revised to include a link to the repository, or directions can be rewritten to direct students to Blackboard, Canvas, Google Drive, OSF, etc.

II. Results

Before using this assignment, an instructor should download the code and data files and try to run `activity.R`, `working_code.R`, and `refactored.Rmd`. As the assignment directions suggest, `activity.R` will error in several ways.

1. Any package dependencies must be installed
2. Two absolute paths are coded into the script

3. Installing the Imap package fails, as it is no longer supported by R
4. If you remove or comment out `library(Imap)` from the code, line 34 beginning `distances <- ...` will error
5. The alt code on line 37 can be made to work if you change it to reference the correct column names, but it calculates Haversine distance, not Vincenty ellipsoid distance

III. Dependencies, Prior Knowledge

1. The `ggplot` and `dplyr` packages are required, but installing the `tidyverse` to match the `library()` call is recommended
2. Transforming RMD to HTML requires `knitr` (alternatively, you can use an R script or convert to Quarto doc)
3. `working_code.R` requires the `geosphere` package, and
4. Prior knowledge of the `tidyverse` is probably not required, but the lesson as written assumes prior coding/R knowledge

IV. Sequencing

The lesson can build on or introduce concepts like:

1. searching for documentation and other published material about code (re:Imap package error)
2. debugging to better understand code design, paths, and file/folder organization
3. using rubber ducking to increase understanding of code
4. comparing buggy code, first draft working code, and refactored code
5. using literate programming methods to bolster reproducibility
6. highlighting level of reproducibility such as errors, dependency issues, and approximate substitutions like Vincenty vs. Haversine
7. framing reproducibility as ongoing practice

V. Modifying This Assignment for Your Class

This assignment should be adaptable to various alternate contexts (levels of instruction, programming languages, etc.). The Haversine vs. Vincenty aspect of the assignment, in particular, could be substituted for for any pair of measures or aggregations where there's a similarity but not an equivalence, and if the differences are related to subject of the source, so much the better. Additional time could be spent on concepts including but limited to the working directory, relative paths, folder organization, hidden files, the .rproj file's role, and how tools like the 'here' and 'renv' packages are used.