# Solving and submitting problem sets

Each problem set in this course will ask you to address three types of questions:

- **The gimmes**: The first few questions of each problem set ask you to do things like name your R Markdown file and load the tidyverse.
- **Straight from the course material**: Most questions will directly reflect the material as it was provided.
- **Integrated questions**: Some questions ask that you integrate across the material that was provided and/or test your conceptual understanding of the material. Do avoid asking yourself "*where did we do ...?*", as this can lead to a major time sink when you're being asked to apply functions in a new way. Taking careful notes during the lectures can help.

**Don't code your way through a problem**! Data wrangling is basically the process of solving a series of puzzles and it's a time sink to solve these puzzles by jumping straight to the code. Instead, *plan ahead* … a whiteboard and/or pencil and scrap paper are a coders most important tools for problem solving.

## Recommended workflow

Invariably, we will reach point in the course in which you've now learned a lot of functions (or, if you've come in with prior tidyverse skills, we've at least *used* a lot of tidyverse functions). I understand that represents a ton of stuff to keep in your brain — don't dive right into the questions and try to think up the right functions to use as you go! Instead, here is my recommended workflow for solving problem sets:

1. **Functions you that may use**: For problem sets, start by looking at the "Functions that you may use …" list. If there are any functions that are unfamiliar to you, use `?[function name]` to learn about it. Then go to your code-along-code scripts and/or course tutorials to find how we've used the function in the past. If you're on a Mac, using Finder with the function name can usually take you to the script where the function was used (I'm not sure if this works well on Windows). I should note that this method will not work on the final project, because you may use *any* of the functions from this course to address the problems (a blessing *and* a curse)!
2. **Explore the data**: Before heading to the questions (other than data loading), explore the data set using functions like `str()` and `summary()`. Pay careful attention to the class and type of values present, the container used to hold those values (e.g., list or atomic vector) and whether there are any NA values. Note that, unless you are explicitly asked to do so, you will not include data exploration code in your submitted assignment.
3. **Visualize**: For each question, visualize what you want the resultant object to look like. Look for clues in the language of the question to determine the *class* of the resultant object (e.g., is it a single value or should you return a data frame?).
4. **Write out the primary steps** (*integrated questions*): If you're confronted with an *integrated question*, plan the primary steps that you need to do to achieve this result using pencil and paper or a whiteboard. I'm a visual-thinker, so I typically make diagrams that look like flow charts. You could just write the steps out too (but use plain language that a non-coder could understand).
5. **Append with the individual stages** (*integrated questions*): Modify your plan (again with pencil and paper or whiteboard) by breaking down any multi-step process into individual stages (still with a diagram or non-coder language). Be specific at this stage (for example "subset the rows of the data frame using a logical test").
6. **Append with the functions** (*integrated questions*): Once you're sure that your plan is complete, start to append it (still pencil and paper or whiteboard) by adding the functions that can be used to address the step in the process (from the assignment list or, for the final project, the functions we've used in

this course). Really give this step careful consideration — for example, if I'm asked to subset a data frame, I know I can choose from the select family of functions to subset a data frame by columns, the slice family to subset by row position, or the filter family to subset by logical condition. Look for clues in the language of the question to help guide you to the family of functions to draw from (e.g., Adding or modifying a variable suggests that you will be using a mutate or summarize).

7. **Code it**: Then (and only then) should you address a problem with code!
8. **Test it**: Make sure the output represents what you expected. If a multi-step process is not generating the expected output (or throwing an error), highlight sections of the code and run it to make sure that each component generates the desired output. If this process includes iteration or a custom function (e.g., an `i` for indexing, an `x` or `.x` for variables in functions), try assigning the index or variable name to your global environment and making sure that it works as expected outside of the loop or function. *Note: The worst thing you can do as this stage is to make changes to the whole code block in hopes of achieving a different outcome.*

**Note**: While the first step is specific to solving our problem sets, steps 2-8 are super useful when you are solving data wrangling problems in the real world!

## Other tips and tricks

**Keep a separate R script for answering problem sets**! It's a pain to code directly in R Markdown. Answer the questions in an R script and then copy-and-paste your answers into the R Markdown file.

I take off points for **code formatting and other minor issues**, but many of these points are easily avoidable if you:

- Always ensure that the naming convention used for your assignment is problem_set_[week]_[last name]_[first name].Rmd. For example, if I submitted a problem set for the first week, the name of my submitted document would be problem_set_1_Evans_Brian.Rmd.
- Don't forget to add your name in the YAML header of the document!
- Never, never, never use the function `setwd()`!
- Always use keyboard shortcuts for adding global assignment operators (`alt/option + dash`), comments (`Ctrl/Cmd + shift + c`), and pipes (`Ctrl/Cmd + shift + m`).
- Before submitting your problem set:
  - Select all content (Ctrl/Cmd + a) and fix any potential indentation errors (Ctrl/Cmd + i).
  - Double-check your code formatting with principles outlined in "Best practices …"/
  - Cross-check your functions (including operators) with those provide to you in the "Functions you may use …" section.
  - Clear your global environment and knit your problem set … if it works with no errors, you are ready to submit!

**A note on Stack Overflow**: Every coder I know (including me) uses the website Stack Overflow when they get stuck. It is an awesome resource where coding questions are answered by members of the Stack Overflow community. Despite its utility, **I strongly advise against using Stack Overflow** when completing weekly problem sets. First, there's simply a lot of bad advice out there – although the Stack Overflow community contains some heavy-hitters in the world of coding (e.g., Ben Bolker), anyone can submit their answer to questions. As a reader, you've got to be discerning out which/whose advise to take. Next, the world of R coding is changing at a rapid pace (especially as it relates to GIS) – a good answer two years ago may not represent a good answer today. Finally, Stack Overflow can be a major time sink – the course content and additional readings that have been provided to you can be used to address all questions for a given problem set.