The main changes/suggestions are summarized below, and primarily thinking about code stability as it's used on new parks or subsequent years and interpretability for new users of the code. These are mostly things I’ve learned the hard way.

1. Added params in the top of the RMD (called the YAML) to make the report even more generalized by park, and less likely to miss code changes required for a different park or year.
2. Switched the confidence interval calculation from assuming a normal distribution to bootstrapping the actual data.
3. Changed some of the bar charts that used year as a factor to line plots where year was treated as a continuous variable on the x-axis.
4. Switched from the magrittr pipe (%>%), which requires dplyr to be loaded, to the native R pipe (|>). There are a few places where the |> doesn't work like the %>% (primarily when joins are included in dplyr chains). I revised the code to handle the |>.
5. Modified code where I could see things not behaving as intended with new data or that could be difficult for a new user to interpret.
   1. I broke up dplyr chains with more than 3-4 pipes, especially if there's dplyr and ggplot code, more than one group\_by/summarize step or multiple joins within a chain. Long dplyr chains are more likely to fail, harder to check your work, and harder for a new user to follow what the code is doing. As someone who is frequently revisiting code I wrote years ago, I find I always have to break up the long dplyr chains to remember what they’re doing. So now I just write them shorter to start with.
   2. Moved a few tasks from dplyr to base R code for stability for functions in dplyr that haven’t been super stable in the past.
6. If the lower confidence interval is < 0 for a metric that can't be negative (eg Basal Area), it's an indication that the normal distribution assumed to generate the CIs is not correct. Instead of changing negative lowerCIs to 0, I recommend bootstrapping the data to get CIs. This works best when there are >= 10 sites, but is okay for ~ 7 plots. In my RMD, I calculated CIs using the bootstrap approach.

In the comments below, line numbers refer to the RMD named Overstory\_Regen\_Canopy\_2.2.Rmd. If I’m referring to the RMD I revised, I’ll state that.

Questions/overall comments:

* How are the csvs that the RMD reads in generated? Is this a query in the database?
  + A long-term goal would be for every step of this process to be in code, including compiling the data used to make the tables and figures.
  + I wasn’t sure, for example, how much QC had been done on the datasets that were compiled. If everything occurs in code, then you can include QC as part of the compilation, and it’s all documented in code
  + If you wanted to generalize on datasets, which I recommend, consider using a standard naming convention for the csvs (eg HOSPCanopy.csv, HOSPOverstory.csv, HOSPRegeneration.csv). Once (if) the process is all in code, this won't matter.
* I reorganized the code in my rmd, so the things you have to modify for new parks/years are in the first code chunk at the top. It also helps show how params are used to generalize code.
* Using parameters in the YAML at the top of the RMD is a much easier way to generalize by park. For example, I'm guessing that lines 67-74 are things you have to update for each park and/or year. Setting those as params (see my example) in the YAML is a lot easier to find/remember to change them. It also allows you to iterate, so that you could run the RMD for every park simultaneously, by iterating on the params.
  + Using params in the YAML also means you don't have to catch everywhere you need to switch a park code, year, etc. For example, Line 13 says LIBO, whereas the csvs are HOSP. See line 99 in my rmd on how I use params to set the title.
* Consider changing the magrittr pipe operator (%>%) to the base R pipe (|>). The %>% pipe requires dplyr to be loaded. The native |> pipe is also supposed to be better optimized for performance. There are only a few cases where the change isn't seamless.
  + You can change your default pipe to |> in Global Options > Code > Use native pipe operator, |>. Once checked, the Ctrl+Shift+M keystroke will use the native pipe.
* It's not wrong to use // for file paths, but I'm seeing cases of \\ and // being used. You only need either \\ or /. Inconsistent uses might be confusing to less R savvy users. In my code, I only use the /.
* Using size to change default linewidth in ggplot was deprecated. Code should change size to linewidth for those cases (eg Line 138).

Comments by line (from original rmd)

* Lines 29-41: The tidyverse package bundle includes ggplot2, dplyr and stringer, so they don't need to be installed separately or loaded separately. I tend to load only the packages I need, so it's clearer what the code dependencies are.
* Line 62: Looks like you're changing Year to a factor, so the X-axis of figures are evenly spaced. I don’t recommend forcing evenly spaced axes, so dropped the code that sets year as a factor.
* Line 65: n is a function in dplyr too, so I'd avoid using it as a variable name to avoid confusion. I changed n to nsites in my code.
* Lines 67, 68, 73, 74 could be accomplished in fewer lines, if you’re interested
  + Year <- unique(na.omit(Overstory$Year2))
  + Current <- max(Year)
  + First <- min(Year)
* Line 76: Nice job creating your own theme! That saves a lot of time coding. Only suggestion is to edit the name from something other than theme (like theme\_veg), because theme is already a function in ggplot2 and could cause confusion for other coders. See the example in my code- called theme\_veg().
* Lines 94 and 103 are naming the same object (BA\_Site\_Class) with the 2nd one including a left\_join with the same named BA\_Site\_Class. If the code fails in the 2nd round, it may continue to run based on the first time it's named, which can make troubleshooting where the issue happened harder. These datasets aren't huge, so there's no need to try to limit the number of objects you're creating. I recommend unique names for each step in the code, and numbering in a way that helps you understand order. I also tend to protect the name of the original data frame that I read in. That way, if I want to start over while coding/troubleshooting a dataset, I don’t have to go back to the top of the file and read it in again.
  + I also simplified this code chunk to make it easier to follow and more stable. One simplification was setting the Class and Site variables in supplement as characters in the top code chunk, so this only had to happen once and didn't happen in the dplyr pipes.
* Line 100: If you don't like the chatty console when you use summarize, you can add: .groups = 'drop' to the function (examples throughout my code).
* Line 106: This is the only place where the native pipe (|>) doesn't work the same as %>%. The native pipe can't handle (.). But I also think it's better to specify the column you want to replace NA as 0 with, so that it's a conscious choice, rather than a blanket apply of 0 to NAs. Usually NAs are something you want to check why they're happening first, in case it's an error or issue in the data, before replacing with 0. Shelby likely did this while coding, but new data sets will be assumed to get the blanket change from NA to 0. I took the replacement of NA to 0 out of the pipe, so you can look at the data before you replace NA with 0.
* Line 119: It's really subtle that BA\_Site is the new variable summing BA\_site with a lowercase s. I try to be more obvious in naming differences, so it’s easy for users of the code to see that difference. In general I minimize capitalizing in code, unless there's a good reason to capitalize (like labels in figures or table headings), to avoid missing capitalization. It also means faster coding, because you don't have to hit shift as often. I took out capitalization from my code for that reason.
* Lines 134 - 139 (and others): It looks like these bar charts are why the code turns year into a character, so the bars (eg YearBA) are evenly spaced on the x-axis that have differing time intervals. While that makes for a more visually appealing figure, it's deceptive and I wouldn't recommend it. I find line plots look better when there are differing lengths of sampling intervals, so my code makes line plots.
* Line 148 (and others): If the lower CI is < 0 for a metric that can't be negative (eg Basal Area), it's an indication that the normal distribution assumed to generate the CIs is not correct. Instead of changing negative lowerCIs to 0, I recommend bootstrapping the data to get CIs. This works best when there are >= 10 sites, but is okay for ~ 7 plots. In my RMD, I calculated CIs using the bootstrap approach.
* Lines 228-234: This code could be simplified by naming the column in the sum. For example:

Density <- Overstory |>

filter(Condition == 'L', SpeciesCode != 'SNAG') |>

mutate(Year = as.character(Year2)) |>

group\_by(Year, Site, Class) |>

summarize(Count = n()) |>

* Line 336: I'm not 100% sure why you added this line. It makes the Y axis range wider than if it wasn't there, but I'm not sure why that's needed.
* Line 358-363: It's usually safer to drop NAs before doing a summarize (or drop them in the summarize) than to do it on the last line. The way it's coded now, any species with an NA record, but that has multiple non-NAs will be dropped. If instead, you do summarize(BA\_site = sum(BA\_ha, na.rm = T)), rows with NAs will be dropped, but species with any non-NA will return a sum.
* Line 367: I've been burned in the past when including math within a sum. I moved the /n outside of the sum() in my code: summarize(MeanBA = sum(BA\_site)/n
* Lines 371-384: This is a LONG dplyr chain. To make it easier to interepret/troubleshoot for the next user, I recommend splitting this at the join. I try to keep dplyr chains to 4 or fewer lines, not that it's a hard rule.
* Line 395: I've been burned using n() this way (like if there's an NA in the data that should not be included). I instead turn it into a logical check that's summed to get a count (e.g. sum(!is.na(DBH)), which counts the number of TRUE cases, with each TRUE case counting as 1.
* Line 420: occurrence is spelled wrong.
* Line 423-433: A couple of thoughts on this code chunk
  + The base R pipe gets cranky with the write.csv() being part of the pipe chain. I'd split it out.
  + This is another long pipe that includes a join. Better to break up the steps a bit.
  + I prefer to sum on a column that will have non-zero values to determine occurrence, instead of length of the unique sites within a species group, in case there are NAs in there that would otherwise get counted.
  + Whenever you're summarizing data and you have to use the distinct() function to get the results you're looking for, there may be a better way to summarize that doesn't require removing duplicates with distinct(). It's at least good practice to figure out why duplicates are still there after group\_by / summarize operations. There may be another round of group\_by / summarize needed. In the case of this code, using summarize instead of mutate would remove the duplicates instead of needing to use distinct().
* Lines 474 - 484:
  + I prefer to make changes with NAs in base R without dependencies, since they'll always work, and gives me a chance to look at the data as I'm making changes to it. At least in the HOSP dataset, there are no negative sums, for example.
  + Do you always know that 4 plots are sampled, or are there rare occasions when fewer than 4 plots are sampled?
  + Changed the CI calculation to bootstrapping instead of assuming normal distribution of canopy cover.
* Lines: 503-510: changing this to line plots because of the different sampling intervals
* Lines 531-543: changing this to bootstrapped CIs; I also broke this into multiple steps, so I could check my work easier.
* Lines 553- 562: While combining dplyr and ggplot code is possible, it's harder for new users to interpret and not the best practice. I recommend splitting these up into different steps.
* Line 558: there's an comma in that line
* Lines 586-600: Revised this to be more stable, not require distinct() and allow for change to be 0.
* Line 615: mutate\_if was superseded in dplyr by use of pick() or across(). It's unfortunate because I find mutate\_if, summarize\_all, etc. to be more intuitive than the new approach. I instead converted NAs to 0 using base R, in case dplyr changes their mind on how to do this again.
* Line 613-614: Instead of doing 2 pivots, where it's hard to follow what happened, you can select multiple names\_from columns.