Appalachian National Scenic Trail Forest Health Monitoring - Standard Operating Procedures

Northeast Temperate Network - National Park Service

2024-04-09

Contents

Chapter 1

Background

1.1 Forest Inventory and Analysis Program (FIA)

The U.S. Forest Service (USFS) administers the Forest Inventory and Analysis Program to acquire inventory data on U.S. forests. The program was launched following the McSweeney - McNary Forest Research Act of 1928 (P.L. 70-466) which led to the first forests being inventoried in 1930 (USFS 2012). Data collected by the program are valuable for a variety of purposes, including timber harvest and ecological health. According to the FIA website, "…FIA reports on status and trends in forest area and location; in the species, size, and health of trees; in total tree growth, mortality, and removals by harvest; in wood production and utilization rates by various products; and in forest land ownership…"FIA data are organized by state, but all state datasets conform to a consistent format. However, in order to use FIA data for the Appalachian National Scenic Trail (APPA), data must be re-aggregated to the APPA area of interest.

This document describes how to use the `rFIA` R package (Stanke et al. 2020) to download USFS FIA data and estimate forest attributes described in the APPA Forest Health Protocol (Dieffenbach 2018). The FIA program conducts annual (panel) inventories within each state. For the APPA region, this is most often a series of 5 annual, spatially unbiased inventories within each sampling cycle. This panel structure allows the FIA program to improve the precision of status and change estimates by leveraging previously collected data within an inventory cycle (e.g., estimate for 2015 may include data from annual inventories conducted from 2010-2015).

This document illustrates how to use the functions in `rFIA` to:

1. Access FIA data

2. Subset the FIA data frame to the APPA HUC10 Shell
3. Produce the following estimates, typically at the plot- and ecoregion-scale:
    - **Live tree abundance**
        - TPA, BAA, biomass, and carbon by species
    - **Species diversity of live trees**
        - Shannon's diversity, evenness, and richness
    - **Tree vital rates**
        - Annual diameter, basal area, and biomass growth by species
    - **Forest demographic rates**
        - Annual recruitment and mortality rates by species
    - **Regeneration abundance**
        - TPA of seedlings and saplings by species and size-class
    - **Snag abundance**
        - TPA, BAA, biomass, carbon, relative fraction
    - **Down woody debris abundance**
        - Volume, biomass, and carbon by fuel class
    - **Invasive Plant abundance**
        - % cover by species
    - **Stand structural stage distributions**
        - % area in pole, mature, and late stage forest

This document does not describe the FIA database structure or demonstrate all of the functionality of the `rFIA` package functions. Readers should consult the documentation within the `rFIA` package for specific function details. Some additional rFIA references include:

- rFIA github page https://github.com/hunter-stanke/rFIA
- rFIA package documentation on CRAN: https://rdrr.io/cran/rFIA/
- rFIA tutorials page: https://rfia.netlify.app/

1.2 SOP Usage

This SOP was written in *R* using `Rmarkdown` and the associated `bookdown` package. Each "Chapter" of this SOP (e.g. 1-Background, 2-Download, 3-Clip, etc.) is a separate `.Rmd` file that can be opened and executed to create the actual files and tables described within a chapter. The most recent, working copies of `.Rmd` files can be found within the (APPAForest github repo) . Because the SOP and the executable code are one and the same, when the R code within the .Rmd is updated, so is the SOP. Furthermore, when the supporting text describing the procedures are updated via the SOP text, the intent of the code should remain clear to the code editor. There are other advantages to the SOP and R code originated from the same source. Namely, this approach saves time (one location to update), reduces the chance of errors, and helps avoid the tendency of an SOP to become asynchronous and out of date with intended procedures.

1.2.1 Annotations

Several annotations are used throughout the SOP. Examples follow:

These messages highlight broken code that requires a fix.

These messages highlight a choice made that may need further examination.

Blue code chunks like this highlight user input options to change functionality:

```
# User input needed in these code chunks
example_variable <- TRUE
```

1.2.2 Run vs. Render

To produce the desired APPA .csv files needed for the forest health report, you must *Run* the individual .Rmd files (see "Run .Rmd files" section). However, to create the SOP document, you will need to *Render* all the .Rmd files together into one document (see "Render SOP" section).

1.2.3 Run .Rmd files

The .Rmd files presented here need to be run in sequential order to produce the APPA FIA data files. Each .Rmd file (except for index.Rmd) has externally saved output (see table 1.1) with the subsequent file utilizing the output from the previous file. The reason the .Rmd is split into different files (instead of one file) is because `01-download.Rmd` and `02-clip.Rmd` are time intensive (i.e. computer processor intensive) steps and the user will not want to run these steps more than once. The last .Rmd file, `03-make.Rmd`, selects the appropriate dataset details for use in the APPA report and will likely need periodic updates/edits. Separating `03-make.Rmd` from the first two steps allows the dataset details to be adjusted without having to redo the initial, time-intensive download and clip steps.

index.Rmd - This chapter, **1 Background**, is (and must be) named `index.Rmd` and contains important header information (not displayed in the final rendered SOP document) which tells R the desired parameters for how the document should be rendered (see "Render SOP" section). All other `.Rmd` files are named according to their corresponding chapter names in this SOP. Note that `index.html` will be the homepage if the SOP document is rendered as a HTML file.

To generate the APPA FIA data files, open a .Rmd file (starting with `01-download.Rmd`) and select `Run All` button from Rstudio or press `Ctrl + Alt + R`.

Table 1.1: The .Rmd files and the corresponding file dataset output.

.Rmd	Output
index.Rmd	None
download.Rmd	.csv files of FIA data for all APPA states (multiple FIA files per state)
clip.Rmd	one .csv file of all FIA plot data within APPA region
make.Rmd	.csv files containing data for each FIA forest measurement type in APPA region.

1.2.4 Render SOP

If edits have been made to one or more of the .Rmd files, the SOP can be rendered as a new document (.pdf, .html, word doc).

You can render the HTML version of this SOP in R by installing the 'bookdown' package, opening the bookdown project file (located within the github repo. For example APPA_SOP.proj), then:

1. Find the **Build** pane in the RStudio IDE, and
2. Click on **Build Book**, then select your output format, or select "All formats" if you'd like to use multiple formats from the same book source files.

Or build the book from the R console:

```
bookdown::render_book()
```

To render to a PDF as a `bookdown::pdf_book`, you'll need to install XeLaTeX. You are recommended to install TinyTeX (which includes XeLaTeX): https://yihui.org/tinytex/.

Run and Render? - Note that when rendering the document the R code itself will not run because `eval = FALSE` in the code chunk at the beginning of the index.Rmd file.

```
knitr::opts_chunk$set(
  eval = FALSE
)
```

If `eval = TRUE` all R code would run in addition to rendering the SOP document. This is not recommended as processes in the initial .Rmd files (downloading and clipping) are time-intensive and best run separately (see "Run .Rmd files" section).

Preview SOP - R studio and bookdown offer different options to preview what the rendered document will look like in real time (updates as you edit) without having to render the entire document. You can start the live preview through the Rstudio interface by selecting the "visual" mode instead of "source" mode. Another option is to start the live preview directly from the R console:

```
bookdown::serve_book()
```

Chapter 2

Download FIA

Note that as of *03 Mar 2024* the `rFIA` R package has not been updated for approximately 1 year. Therefore, some functionality has been lost because the package is not being actively maintained. For example, downloading FIA data for multiple states using the `getFIA()` function currently results in an error due to recent updates to the online FIA database (FIA DataMart).

Currently, there are two suggested work-arounds: 1) Download the outdated package and manually fix the source code, or 2) Download the rFIA package from an alternative repo with some fixes already applied.

2.1 Install rFIA package

2.1.1 Option 1: Install rFIA from offical repo, then manually fix package:

```r
remotes::install_github("hunter-stanke/rFIA")
library(rFIA)

# Then, edit the source code of the package using the changes outlined here:
# https://github.com/hunter-stanke/rFIA/pull/46/files
```

2.1.2 Option 2: Install rFIA from alternative repo:

Issue with multi-state download request fixed in this version below. But note that this copy is only supplied as a temporary solution and is also not being actively updated or maintained.

```
remotes::install_github("jakegross808/rFIA")
library(rFIA)
```

2.2 Setup parallel processing

Setting up parallel processing in R may help with downloading extensive amounts of data from the FIA datamart:

```
## How many physical cores do you have?
parallel::detectCores(logical = FALSE)

## How many cores do you want to use?

## If you still want to use your computer for anything else during the computations
## (e.g., checking your E-mails or writing a Word document),
## you should reserve at least one core for those remaining tasks.
cores <- parallel::detectCores(logical = FALSE)-2 # set to use all physical cores minu
```

2.3 Download FIA data for each state intersecting APPA

The `rFIA` function `getFIA()`(https://rdrr.io/cran/rFIA/man/getFIA.html) downloads State FIA Data from the FIA Datamart.

2.3.1 Test Run

Use the small “test dataset” below to see if the function is working properly before downloading the very large Appalachian trail dataset. Test dataset consists of a small amount of FIA data from US territories, American Samoa and Guam.

```
## Small subset of FIA dataset to test 'getFIA()' before big download
test <- c('AS', 'GU') # smallest datasets in 2023

## If dir = NULL tables will not be saved on disk and only loaded into R environment
test_FIA_db_object <- getFIA(states = test, dir = NULL, nCores = cores)

str(test_FIA_db_object)
```

2.3.2 Save location

Choose where local FIA database tables will be saved:

```r
## default location:
save_default <- './download_FIA/'
## Alternatively, enter specific custom location:
save_custom <- 'C:/Users/JJGross/Documents/R_projects/FIA_data/allStates'

## Specify `save_location <- save_custom` or `save_location <- save_default`
save_location <- save_custom

## Create directory if it doesn't already exist
if (!dir.exists(save_location)) {dir.create(save_location)}
```

2.3.3 Download all FIA data from all 13 APPA states:

This is a large amount of data and depending on connection, may take more than an hour to complete. Note that `load = FALSE` saves the dataset to hard drive location instead of loading directly into R session. The dataset saved to the hard drive will be loaded into R using steps outlined in the next chapter. “Clip”.

Argument `common = TRUE` only imports the most commonly used FIA tables, including all those required for rFIA functions.

```r
## APPA States
at_states <- c('CT', 'GA', 'ME', 'MD', 'MA',
               'NH', 'NJ', 'NY', 'NC', 'PA',
               'TN', 'VT', 'VA')

## Download the data:
getFIA(states = at_states, dir = save_location,
       nCores = cores, common = TRUE, load = FALSE)
```

Chapter 3

Clip by Ecoregion

3.1 Location to save output

```r
## default location:
save_default <- './clip_FIA/'
## Alternatively, enter specific custom location:
save_custom <- 'C:/Users/JJGross/Documents/R_projects/FIA_data/clip_FIA/'

## Specify `save_location <- save_custom` or `save_location <- save_default`
save_location <- save_custom

## Create directory if it doesn't already exist
if (!dir.exists(save_location)) {dir.create(save_location)}
```

3.2 Load 'Ecoregions' spatial data

Load in a file containing hierarchical spatial categories, or levels, of polygons surrounding the Appalachian National Scenic Trail (more info on how this file was created can be found in APPA Forest Health FIA Narrative). The hierarchical spatial levels (from smallest to largest) include ecological subsections, ecological sections, and provinces (all collectively referred to as ecoregions here). The APPA Ecoregions dataset can be downloaded from IRMA: APPA_Ecoregions_USFS_HUC10_Shell_AEA

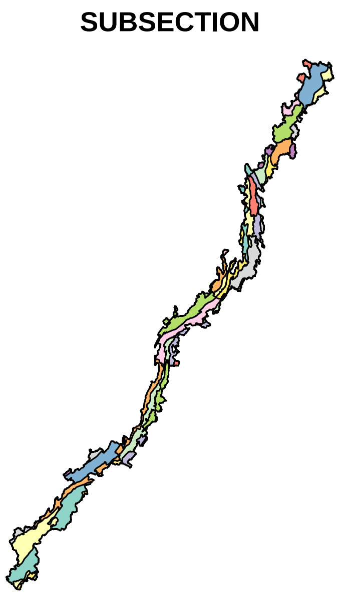
```r
load_eco_path <- "C:/Users/JJGross/Downloads/APPA_Ecoregions_USFS_HUC10_Shell_AEA"
```

Read the shapefile using `read_sf()` from the `sf` (simple features) package.

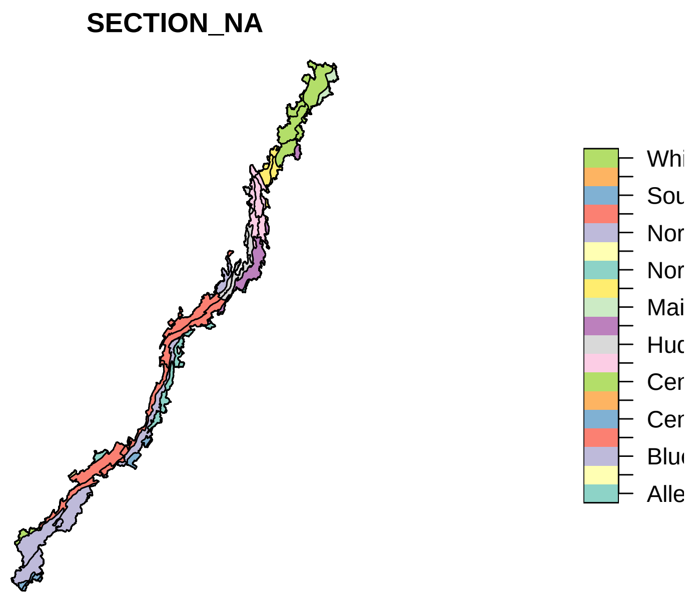
```
eco_shapefile <- sf::read_sf(dsn = load_eco_path,
                             layer = 'APPA_Ecoregions_USFS_HUC10_Shell_AEA')
```

3.2.1 Explore ecoregion levels:

```
plot(eco_shapefile["SUBSECTION"])
```



```
plot(eco_shapefile["SECTION_NA"])
```



```
## SUBSECTIONS:
n_distinct(eco_shapefile$SUBSECTION)
```

```
## [1] 50
```

```
levels(as.factor(eco_shapefile$SUBSECTION))
```

```
##  [1] "211Aa"  "211Ba"  "211Bb"  "211Da"  "211Ec"  "211Fc"  "211Fd"  "211Ia"
##  [9] "221Ae"  "221Al"  "221Am"  "221Ba"  "221Bb"  "221Bd"  "221Da"  "221Db"
## [17] "221Dc"  "221Dd"  "221De"  "221Ja"  "221Jb"  "221Jc"  "231Ab"  "231Ac"
## [25] "231Ad"  "231Ag"  "231Ib"  "M211Ab" "M211Ac" "M211Ad" "M211Ae" "M211Af"
## [33] "M211Ag" "M211Ba" "M211Bb" "M211Bc" "M211Ca" "M211Cb" "M211Cc" "M211Cd"
## [41] "M221Aa" "M221Ab" "M221Ac" "M221Ad" "M221Be" "M221Cb" "M221Da" "M221Db"
## [49] "M221Dc" "M221Dd"
```

```
## SUBSECTI_1:
levels(as.factor(eco_shapefile$SUBSECTI_1))
```

```
##  [1] "Aroostook Hills"
##  [2] "Berkshire-Vermont Upland"
##  [3] "Catskill Mountains"
##  [4] "Central Blue Ridge Mountains"
##  [5] "Central Maine Embayment"
##  [6] "Central Maine Foothills"
##  [7] "Champlain Glacial Lake and Marine Plains"
##  [8] "Connecticut Lakes"
##  [9] "Eastern Allegheny Plateau"
## [10] "Eastern Coal Fields"
## [11] "Gettysburg Piedmont Lowland"
## [12] "Great Valley of Virginia"
## [13] "Holston Valley"
## [14] "Hudson Highlands"
## [15] "Hudson Limestone Valley"
## [16] "Kittatinny-Shawangunk Ridges"
## [17] "Lower Foot Hills"
## [18] "Lynchburg Belt"
## [19] "Mahoosic Rangely Lakes"
## [20] "Maine-New Brunswick Lowlands"
## [21] "Maine Central Mountains"
## [22] "Metasedimentary Mountains"
## [23] "Newark"
## [24] "Northern Blue Ridge Mountains"
## [25] "Northern Great Valley"
## [26] "Northern Green Mountain"
## [27] "Northern Piedmont"
## [28] "Northern Ridge and Valley"
## [29] "Piedmont Ridge"
## [30] "Piedmont Upland"
## [31] "Pocono Plateau"
## [32] "Reading Prong"
## [33] "Ridge and Valley"
## [34] "Rolling Limestone Hills"
## [35] "Sandstone Hills"
## [36] "Schist Hills"
## [37] "Schist Plains"
## [38] "Sebago-Ossipee Hills and Plains"
## [39] "Southern Blue Ridge Mountains"
## [40] "Southern Green Mountain"
## [41] "Southern Piedmont"
## [42] "St. John Upland"
## [43] "Sunapee Uplands"
## [44] "Taconic Foothills"
## [45] "Taconic Mountains"
## [46] "Triassic Basins"
## [47] "Western Allegheny Mountain and Valley"
## [48] "Western Maine Foothills"
## [49] "White Mountains"
```

```
n_distinct(eco_shapefile$SUBSECTI_1)
```

```
## [1] 49
```

```
## Note that 'SUBSECTI_1' names are not unique - there are two different "Northern Piedmont"
eco_shapefile %>%
  filter(SUBSECTI_1 == "Northern Piedmont") %>%
  select(SUBSECTI_1, SUBSECTION, SECTION_NA, PROVINCE_N) %>%
  sf::st_drop_geometry()
```

```
## # A tibble: 2 x 4
##   SUBSECTI_1        SUBSECTION SECTION_NA                   PROVINCE_N
## * <chr>             <chr>      <chr>                        <chr>
## 1 Northern Piedmont M211Ba     New England Piedmont         Adirondack-New Eng~
## 2 Northern Piedmont 221De      Northern Appalachian Piedmont Eastern Broadleaf ~
```

```
eco_shapefile <- eco_shapefile %>%
  mutate(SUBSECTI_1 = case_when(
    SUBSECTI_1 == "Northern Piedmont" & SUBSECTION == "M211Ba" ~ "New Engl. Northern Piedmont",
    SUBSECTI_1 == "Northern Piedmont" & SUBSECTION == "221De" ~ "Appal. Northern Piedmont",
    .default = as.character(SUBSECTI_1)))
```

Note that `SUBSECTI_1` names are not unique. There are two different subsections both named “Northern Piedmont”. See current re-name solution above.

```
## Distinct SECTION_NA:
n_distinct(eco_shapefile$SECTION_NA)
```

```
## [1] 19
```

```
levels(as.factor(eco_shapefile$SECTION_NA))
```

```
##  [1] "Allegheny Mountains"
##  [2] "Aroostook Hills and Lowlands"
##  [3] "Blue Ridge Mountains"
##  [4] "Catskill Mountains"
##  [5] "Central Appalachian Piedmont"
##  [6] "Central Maine Coastal and Embayment"
##  [7] "Central Ridge and Valley"
##  [8] "Green-Taconic-Berkshire Mountains"
##  [9] "Hudson Valley"
## [10] "Lower New England"
## [11] "Maine - New Brunswick Foothills and Lowlands"
## [12] "New England Piedmont"
## [13] "Northern Appalachian Piedmont"
## [14] "Northern Cumberland Mountains"
## [15] "Northern Glaciated Allegheny Plateau"
## [16] "Northern Ridge and Valley"
## [17] "Southern Appalachian Piedmont"
## [18] "St. Lawrence and Champlain Valley"
## [19] "White Mountains"
```

```r
## Distinct PROVINCE_N:
n_distinct(eco_shapefile$PROVINCE_N)
```

```
## [1] 5
```

```r
levels(as.factor(eco_shapefile$PROVINCE_N))
```

```
## [1] "Adirondack-New England Mixed Forest--Coniferous Forest--Alpine Meadow"
## [2] "Central Appalachian Broadleaf Forest-Coniferous Forest-Meadow"
## [3] "Eastern Broadleaf Forest"
## [4] "Northeastern Mixed Forest"
## [5] "Southeastern Mixed Forest"
```

3.3 Read FIA data into R

Load the FIA data stored locally (From “Download Chapter) into R session.

```r
## load from default location (specified in 01-Download.Rmd chapter):
load_default <- './download_FIA/'
## Alternatively, enter specific custom location:
load_custom <- 'C:/Users/JJGross/Documents/R_projects/FIA_data/allStates'

load_location <- load_custom
#load_location <- load_default

## Change number of processing cores used, if desired
cores <- parallel::detectCores(logical = FALSE)-2
```

Read the FIA data stored locally (From “Download Chapter) using the Read-FIA() function.

```r
## Specify the 13 APPA states by state code (this is extra insurance that the wrong states are no
at_states <- c('CT', 'GA', 'ME', 'MD', 'MA',
               'NH', 'NJ', 'NY', 'NC', 'PA',
               'TN', 'VT', 'VA')

## Read FIA data
at_states_FIA <- readFIA(dir = load_location, states = at_states, nCores = cores)
```

3.4 Clip FIA plots by Ecoregion

clipFIA() performs space-time queries on Forest Inventory and Analysis Database (FIADB). `clipFIA` subsets the dataset to include only data associated with particular inventory years (i.e., most recent), and/or only data within a user-defined region.

Spatially, the `mask = eco` argument below selects only the `at_states_FIA` FIA plots which fall within the perimeter of the `eco` spatial extent (i.e. the outer boundary of the ecoregion polygons).

Temporally, the `mostRecent` argument returns only data for most recent inventory if set to `TRUE`

```r
## Restrict to most recent inventory
at_FIA_MR <- clipFIA(at_states_FIA, matchEval = TRUE, mostRecent = TRUE, mask = eco_shapefile)
saveRDS(at_FIA_MR, file = paste0(save_location, "at_FIA_MR.rds"))

## Access all inventories
at_FIA <- clipFIA(at_states_FIA, matchEval = TRUE, mostRecent = FALSE, mask = eco_shapefile)
saveRDS(at_FIA, file = paste0(save_location, "at_FIA.rds"))
```

Note the argument `matchEval = TRUE` in the above function. The clipFIA() documentation states that:

> "if `matchEval = TRUE`, `clipFIA()` returns a subset of data for which there are matching reporting years across states. Only useful if db contains multiple state subsets of the FIA database."

This seems appropriate for the Appalachian Trail dataset, but when compared (12 March 2024), both `matchEval = TRUE` and `matchEval = FALSE` appeared to result in the same output. This may have something to do with the method that is used (e.g. 'Annual'" ' vs 'TI'). Further evaluation is needed.

Note that currently matchEval argument is set to `matchEval = TRUE` for both `at_FIA` and `at_FIA_MR`. Confirm this is appropriate.

3.4.1 Ecosubsections

It is important to note the column `ECOSUBCD` which is part of the native FIA database and referenced in the FIA Database User Guide. The `ECOSUBCD` column can be included in the output of most `rFIA` functions with the argument `grpBy = c(ECOSUBCD)`. The FIA database guide states that subsection codes for the conterminous United States were developed as part of the "Ecological Subregions: Sections and Subsections for the Conterminous United States" publication (Cleland et al. 2007).

Conversely, the same ecoregional subsection code can be found within the `APPA_Ecoregions_USFS_HUC10_Shell_AEA` shapefile column `eco$SUBSECTION` along with additional related columns - for example, the full subsection name in the `SUBSECTI_1` column. These columns (and others added below) are appended to the FIA dataset during most rFIA functions by way of the `polys = eco`. For this reason the `polys = eco`argument is often more advantageous than the `grpBy = c(ECOSUBCD)` and the `polys = eco`argument is utilized throughout the make.Rmd chapter.

Note that this `polys = eco` argument is different than the `rFIA::clipFIA(mask = eco)` function/argument which only clips the plot data by the eco mask (outer spatial boundary) and does not append any data.

3.5 Supporting tables

3.5.1 Appalachian trail spatial centerline data

```
## Manually downloaded `Appalachian_National_Scenic_Trail.shp` from `APPAForesthealthR
centerline_shapefile_location <- "move_to_server/at_centerline/"

## read in shapefile of Appalachian trail center-line
 at_centerline <- sf::st_read(paste0(centerline_shapefile_location, "Appalachian_Natio
   mutate(region_3cl = case_when(
     Region == "New England" ~ "Northeast",
     Region == "Mid-Atlantic" ~ "Mid-Atlantic",
     Region %in% c("Southern", "Virginia") ~ "Southeast"
   ))

saveRDS(at_centerline, paste0(save_location, "at_centerline.rds"))
```

`Appalachian_National_Scenic_Trail.shp` currently saved within `APPAForesthealthReport` repo. In future move this to appropriate location (e.g. IRMA).

3.5.2 APPA FIA plots

Note that the `rFIA::tpa()` function was used below to calculate and construct tables for APPA FIA plot-level attributes (e.g. elevation, etc.). TPA was selected because it is a core metric of FIA methods and should be present in all field visited plots.

```r
## Get plot-level data for the most recent tpa dataset
tpa_plots_MR <- rFIA::tpa(at_FIA_MR,
                          byPlot = TRUE,
                          grpBy = c(STATECD, ECOSUBCD, ELEV),
                          returnSpatial = TRUE)

## Get plot-level data for full tpa dataset
tpa_plots <- rFIA::tpa(at_FIA,
                       byPlot = TRUE,
                       grpBy = c(STATECD, ECOSUBCD, ELEV),
                       returnSpatial = TRUE)

## Get only the unique plot locations (pltID)
## i.e. exclude the revisits
tpa_plots_by_year <- tpa_plots |>
  # HAVE TO DO BIND ROWS HERE BECAUSE `tpa_plots' SUPRISINGLY DOES NOT
  # CONTAIN ALL PLOTS FOUND IN `tpa_plots_MR'.
  # Error with rFIA discussed below
  dplyr::bind_rows(tpa_plots_MR) |>
  dplyr::group_by(PLT_CN) |> #PLT_CN is each unique event, while pltID is each unique plot locat
  distinct()

## Get only the unique plot locations (pltID)
## i.e. exclude the revisits
tpa_plots_distinct <- tpa_plots_by_year |>
  select(pltID, STATECD, ECOSUBCD, ELEV, geometry) %>%
  distinct()
```

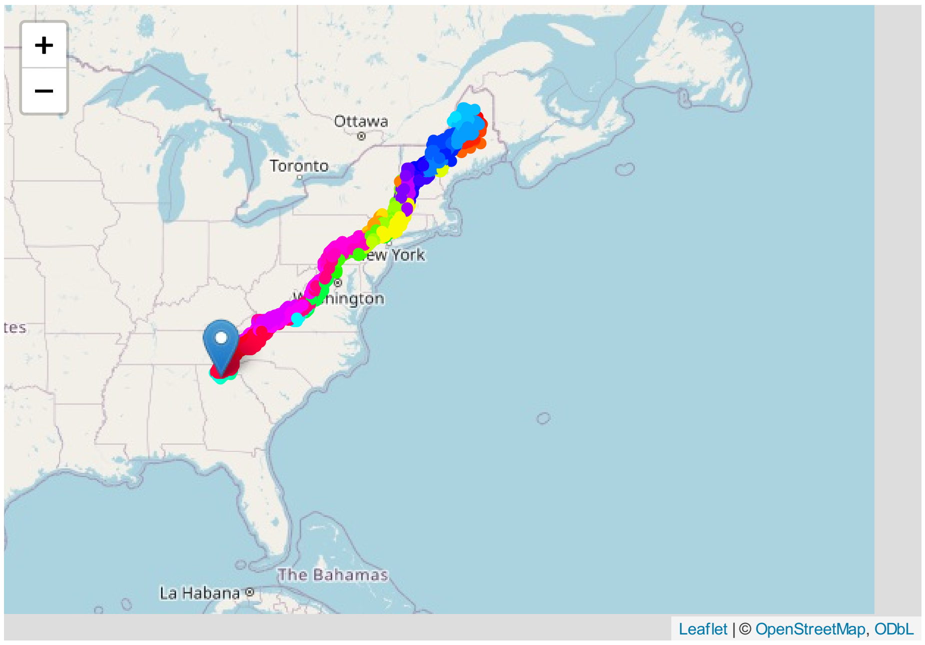
Display APPA FIA plots by eco-subsection and southern most plot used to calculated distance.

```r
## Select Southern-most plot
most_S_plot <- tpa_plots_distinct %>%
  filter(pltID == "5_13_85_5")

## ecosubsection colors
ecosubCol <- colorFactor(rainbow(51), tpa_plots_distinct$ECOSUBCD)
## leaflet map
```

```
tpa_plots_distinct %>%
  leaflet() %>%
  addTiles() %>%
  addCircleMarkers(
    color = ~ecosubCol(ECOSUBCD),
    stroke = FALSE, fillOpacity = 1,
    radius = 4) %>%
  addMarkers(data = most_S_plot)
```

```
## Google Chrome was not found. Try setting the `CHROMOTE_CHROME` environment variable
```

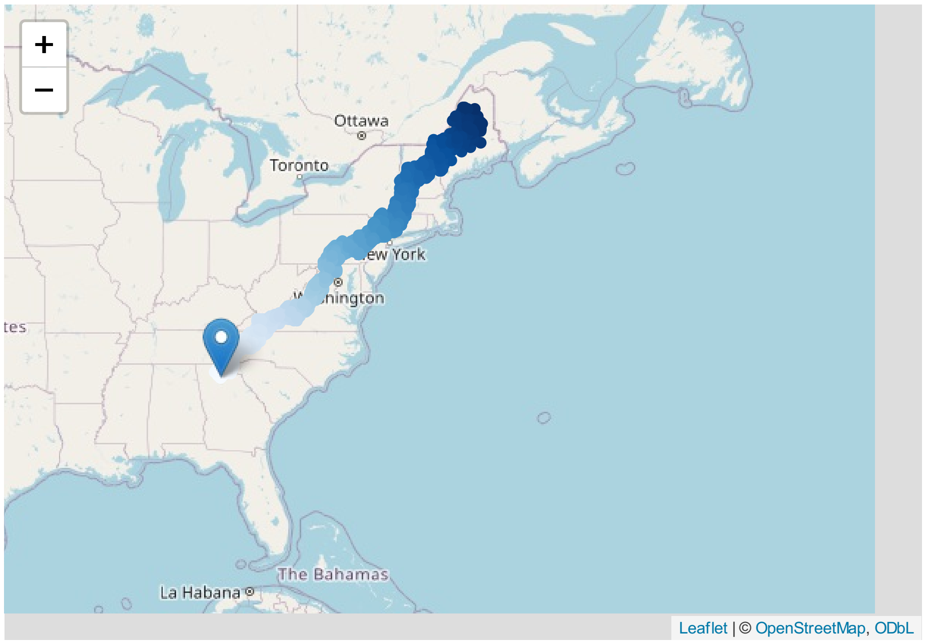


Calculate each plot's distance from southern point for use in displaying data as transect along axis.

```
tpa_plots_distance <- tpa_plots_distinct |>
  mutate(S_plot = most_S_plot$geometry) |> # add south point
  mutate(dist_unit = sf::st_distance(
    S_plot, geometry, by_element = TRUE)) |> # calculate distance
  mutate(distance_m = round(as.numeric(dist_unit))) |>
  select(-S_plot, -dist_unit)

## Create a continuous palette function
dist_pal <- colorNumeric(palette = "Blues", domain = tpa_plots_distance$distance_m)
## leaflet map
```

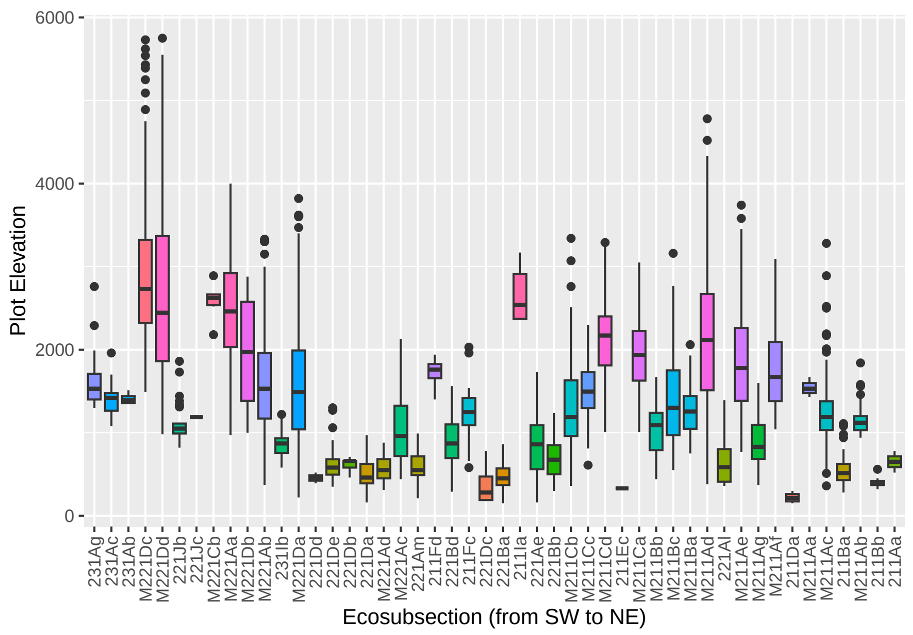
```
tpa_plots_distance %>%
  leaflet() %>%
  addTiles() %>%
  addCircleMarkers(
    color = ~dist_pal(distance_m),
    stroke = FALSE, fillOpacity = 1,
    radius = 4) %>%
  addMarkers(data = most_S_plot)
```



Demo usage of plot attributes

```
plot_attributes <- tpa_plots_distance

# Demo plot distance x elevation
plot_attributes |>
  group_by(ECOSUBCD)|>
  mutate(median_dist = median(distance_m)) |>
  mutate(median_elev = median(ELEV)) |>
  ggplot(aes(x= fct_reorder(ECOSUBCD, distance_m),
             y=ELEV,
             fill = as.factor(median_elev))) +
  geom_boxplot() +
  theme(axis.text.x = element_text(angle = 90, vjust = 0.5, hjust=1),
        legend.position = 'none') +
  xlab("Ecosubsection (from SW to NE)") + ylab("Plot Elevation")
```



3.5.2.1 Check Plot Counts

Perform some checks on records within tpa dataset - helps gauge dataset changes. Can also add quality assurance checks here:

```
## Check that entire dataset (at_plots) contains most recent dataset (at_plots_MR)
tpa_chk <- sf::st_drop_geometry(tpa_plots)
tpa_chk_MR <- sf::st_drop_geometry(tpa_plots_MR)

MRplots_not_in_full_tpa_dataset <- anti_join(tpa_chk_MR, tpa_chk, by = join_by(pltID))
if (nrow(MRplots_not_in_full_tpa_dataset) != 0) {
  warning("tpa_plots does not contain all plots found in tpa_plots_MR (most recent) da
}
```

```
## Warning: tpa_plots does not contain all plots found in tpa_plots_MR (most
## recent) dataset
```

```
## total events/visits
tpa_event_count <- nrow(tpa_chk)
tpa_event_count # 12,457
```

```
## [1] 12457
```

```
## total plot locations
tpa_plot_count <- length(unique(tpa_chk$pltID))
tpa_plot_count # 4,328
```

```
## [1] 4328
```

```
tpa_plots_summary <- tpa_chk |>
  group_by(pltID) |>
  summarise(visits = n()) |>
  group_by(visits) |>
  summarize(n_plots = n())

tpa_plots_summary
```

```
## # A tibble: 5 x 2
##   visits n_plots
##    <int>   <int>
## 1      1     571
## 2      2     781
## 3      3    1590
## 4      4    1376
## 5      5      10
```

```
## In 2022
#      1     571 (571 plots measured once)
#      2     781 (781 plots measured twice)
#      3    1590
#      4    1376
#      5      10

## total events/visits
tpa_event_count_MR <- nrow(tpa_chk_MR)
tpa_event_count_MR # 3,980
```

```
## [1] 3980
```

```
## total plot locations
tpa_plot_count_MR <- length(unique(tpa_chk_MR$pltID))
tpa_plot_count_MR # 3,980
```

```
## [1] 3980
```

```r
if (tpa_event_count_MR != tpa_plot_count_MR) {
  warning("some plots not unique in most recent (MR) dataset")
}
```

Possible error with rFIA: tpa_plots (full dataset) does not contain all plots found in tpa_plots_MR (most recent) dataset" see `MRplots_not_in_full_tpa_dataset` for table of missing plots.

3.5.3 Subsections

3.5.3.1 Check subsections

```r
tpa_ECOSUBCD_distinct <- tpa_plots_distinct |>
  sf::st_drop_geometry() |>
  distinct(ECOSUBCD)

## Eco shapefile:
eco <- eco_shapefile %>%
  full_join(tpa_ECOSUBCD_distinct, by = c("SUBSECTION" = "ECOSUBCD")) %>%
  sf::st_transform('+proj=longlat +datum=WGS84')

## return all rows from eco_shapefile without a match in FIA_plots
## These are subsections with no FIA plots:
dplyr::anti_join(eco, tpa_ECOSUBCD_distinct, by = join_by(SUBSECTION == ECOSUBCD))
```

```
## Simple feature collection with 3 features and 15 fields
## Geometry type: MULTIPOLYGON
## Dimension:     XY
## Bounding box:  xmin: -83.99575 ymin: 34.51386 xmax: -80.44353 ymax: 37.65403
## Geodetic CRS:  +proj=longlat +datum=WGS84
## # A tibble: 3 x 16
##      AREA PERIMETER ECOMAP_200 ECOMAP_201 PROVINCE SECTION SUBSECTION DOMAIN_NAM
##     <dbl>     <dbl>      <int>      <int> <chr>    <chr>   <chr>      <chr>
## 1  7.42e8   436121.       1232       1381 M221     M221B   M221Be     Humid Tem~
## 2  3.99e4  1265905.       1480       1590 221      221J    221Ja      Humid Tem~
## 3  2.10e6   415055.       1688       1746 231      231A    231Ad      Humid Tem~
## # i 8 more variables: DIVISION_N <chr>, PROVINCE_N <chr>, SECTION_NA <chr>,
## #   SUBSECTI_1 <chr>, ATIntersec <int>, Acres <dbl>, Hectares <dbl>,
## #   geometry <MULTIPOLYGON [°]>
```

```r
## return all rows from FIA_plots without a match in eco_shapefile
## These rows are locations with FIA plots but no subsection included in shapefile:
eco %>% filter(is.na(SECTION))
```

```
## Simple feature collection with 1 feature and 15 fields (with 1 geometry empty)
## Geometry type: MULTIPOLYGON
## Dimension:     XY
## Bounding box:  xmin: NA ymin: NA xmax: NA ymax: NA
## Geodetic CRS:  +proj=longlat +datum=WGS84
## # A tibble: 1 x 16
##    AREA PERIMETER ECOMAP_200 ECOMAP_201 PROVINCE SECTION SUBSECTION DOMAIN_NAM
## * <dbl>     <dbl>      <int>      <int> <chr>    <chr>   <chr>      <chr>
## 1    NA        NA         NA         NA <NA>     <NA>    M211Aa     <NA>
## # i 8 more variables: DIVISION_N <chr>, PROVINCE_N <chr>, SECTION_NA <chr>,
## #   SUBSECTI_1 <chr>, ATIntersec <int>, Acres <dbl>, Hectares <dbl>,
## #   geometry <MULTIPOLYGON [°]>
```

3.5.3.2 Missing eco$SUBSECTION info

Note that `eco` shapefile is missing information for SUBSECTION `M211Aa - International Boundary Plateau`. This needs to be fixed in original shapefile.

Temporary fix here:

```r
## missing info found by google search of "M211Aa"
# https://www.fs.usda.gov/research/publications/misc/73326-wo-gtr-76d-cleland2007.pdf
missing_subsection <- data.frame(
  PROVINCE = "M211",
  SECTION = "M211A",
  SUBSECTION = "M211Aa",
  SECTION_NA = "White Mountains",
  SUBSECTI_1 = "International Boundary Plateau")

eco <- eco |>
  filter(SUBSECTION != "M211Aa") |>
  bind_rows(missing_subsection)
```

3.5.3.3 Calculate n plots

Calculate plots per subsection and median elevation and distance for all and most recent (MR) datasets

```r
# calculate plots per subsection
plot_attributes_drop_geo <- plot_attributes %>%
  sf::st_drop_geometry()

n_plots_tpa_MR <- tpa_plots_MR |>
  sf::st_drop_geometry() |>
  left_join(plot_attributes_drop_geo,
            by = join_by(pltID, STATECD, ECOSUBCD, ELEV)) |>
  group_by(ECOSUBCD) |>
  summarize(n_plots_subset_MR = n(),
            median_elevation_MR = median(ELEV),
            median_distance_MR = median(distance_m))

n_plots_tpa <- tpa_plots |>
  sf::st_drop_geometry() |>
  left_join(plot_attributes_drop_geo,
            by = join_by(pltID, STATECD, ECOSUBCD, ELEV)) |>
  group_by(ECOSUBCD) |>
  summarize(n_plots_subset = n(),
            median_elevation = median(ELEV),
            median_distance = median(distance_m))
```

3.5.3.4 Add states and subsection abbreviations

Shorter subsection names with states they occur in.

```
## Read in state code names
state_codes <- read_csv("move_to_server/us-state-ansi-fips.csv", show_col_types = FALSE
  mutate(st = as.integer(st))

## Get all subsections in one dataset (subsections from FIA + eco shapefile)
## Have to join because these can be different if subsection missing from eco shapefil
## or FIA data not collected for a subsection
subsections_per_state <- plot_attributes_drop_geo |>
  select(STATECD, ECOSUBCD) |>
  distinct() |>
  right_join(eco, by = join_by(ECOSUBCD == SUBSECTION))

subsections_all <- subsections_per_state |>
  select(-STATECD) |>
  group_by(ECOSUBCD) |>
  distinct()

## Get list of which states a subsection occurs - will help with data viz interpretati
subsection_state_list <- subsections_per_state |>
  select(STATECD, ECOSUBCD) |>
  distinct() |>
  left_join(state_codes, by = join_by(STATECD == st)) |>
  select(-STATECD, -stname) |>
  arrange(stusps) |>
  pivot_wider(names_from = stusps, values_from = stusps) |>
  unite("states", CT:VT, remove = FALSE, na.rm = TRUE, sep = " ") |>
  mutate(states = paste0("(",states,")")) |>
  select(ECOSUBCD, states)

## Abbrivations to shorten subsection names
replace_list <- (c("Mountain" = "Mt",
                  "Central" = "C.",
                  "Northern" = "N.",
                  "Southern" = "S.",
                  "Eastern" = "E.",
                  "New Brunswick" = "NB",
                  "Lowlands" = "Lowl.",
                  "Lowland" = "Lowl.",
                  "Uplands" = "Upl.",
                  "Upland" = "Upl.",
                  "Plateau" = "Plat.",
                  "Piedmont" = "Pdmt",
                  "Blue" = "Bl.",
                  "Ridge" = "Rdg",
                  "Valley" = "Vly",
                  "Maine" = "ME",
                  "Connecticut" = "CT",
                  "Vermont" = "VT",
                  "Virginia" = "VA",
                  "Western" = "W.",
                  "Glacial Lake and Marine Plains" = "GL/MP",
                  "Hills and Plains" = "",
                  "Kittatinny-Shawangunk" = "Kitt-Shaw",
                  "Metasedimentary" = "Meta",
                  "Mahoosic Rangely Lakes" = "Mahoo/Rng Lks",
                  "Highlands" = "Highl.",
                  "Hudson" = "Hud.",
                  "Limestone" = "Limstn",
                  "Gettysburg" = "Getty",
                  "Sebago-Ossipee" = "Seb-Ossipee",
                  "Berkshire" = "Berk",
                  "Embayment" = "Embaymt",
                  "International" = "Int.",
                  "Plateau" = "Plat."
                  ))

subsections_abbr <- subsections_all |>
  left_join(subsection_state_list, by = join_by(ECOSUBCD)) |>
  mutate(SUBSECT_ABBR = str_replace_all(SUBSECTI_1, replace_list)) |>
  unite("SUBSECT_ABBR_ST", SUBSECT_ABBR:states, remove = FALSE, na.rm = TRUE, sep = "
```

3.5.4 Save subsection attributes

```
# append n plot counts for entire dataset and most recent
subsection_attributes <- subsections_abbr |>
  left_join(n_plots_tpa, by = join_by(ECOSUBCD)) |>
  left_join(n_plots_tpa_MR, by = join_by(ECOSUBCD))

saveRDS(subsection_attributes, file = paste0(save_location, "subsection_attributes.rds
```

3.5.5 Save centroids

```
section_centroids <- subsection_attributes |>
  sf::st_as_sf() |>
  group_by(SECTION_NA) |>
  summarise(geometry = sf::st_union(geometry)) |>
  sf::st_centroid() |>
  mutate(X = sf::st_coordinates(geometry)[,1],
         Y = sf::st_coordinates(geometry)[,2])
```

```
## Warning: st_centroid assumes attributes are constant over geometries
```

```
str(section_centroids)
```

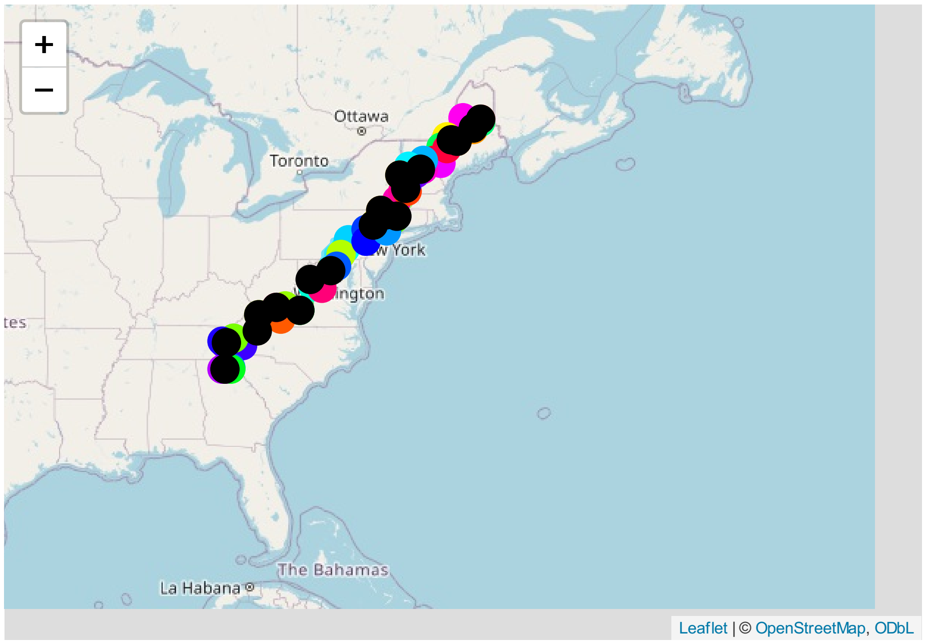
```
## sf [19 x 4] (S3: sf/tbl_df/tbl/data.frame)
##  $ SECTION_NA: chr [1:19] "Allegheny Mountains" "Aroostook Hills and Lowlands" "Blu
##  $ geometry  :sfc_POINT of length 19; first list element:  'XY' num [1:2] -80.8 37.
##  $ X         : num [1:19] -80.8 -68.6 -81.9 -74.5 -79.4 ...
##  $ Y         : num [1:19] 37.5 45.9 36.4 42 37.4 ...
##  - attr(*, "sf_column")= chr "geometry"
##  - attr(*, "agr")= Factor w/ 3 levels "constant","aggregate",..: NA NA NA
##   ..- attr(*, "names")= chr [1:3] "SECTION_NA" "X" "Y"
```

```
subsection_centroids <- subsection_attributes |>
  sf::st_as_sf() |>
  sf::st_centroid()
```

```
## Warning: st_centroid assumes attributes are constant over geometries
## leaflet map
ecosubCol <- colorFactor(rainbow(51), subsection_attributes$SUBSECT_ABBR)

subsection_attributes |>
  sf::st_as_sf() |>
  leaflet() |>
  addTiles() |>
  addPolygons(color = ~ecosubCol(SUBSECT_ABBR), label = ~SUBSECT_ABBR) |>
  addCircleMarkers(data = subsection_centroids,
                   color = ~ecosubCol(SUBSECT_ABBR),
                   stroke = FALSE, fillOpacity = 1,
                   radius = 10, label = ~SUBSECT_ABBR) |>
  addCircleMarkers(data = section_centroids,
                   color = "black",
                   stroke = FALSE, fillOpacity = 1,
                   radius = 10, label = ~SECTION_NA)
```

```
## Warning in validateCoords(lng, lat, funcName): Data contains 1 rows with either
## missing or invalid lat/lon values and will be ignored
```



```
saveRDS(section_centroids, file = paste0(save_location, "section_centroids.rds"))
saveRDS(subsection_centroids, file = paste0(save_location, "subsection_centroids.rds"))
```

A centroid file was in original repo - but not sure how it was created - took a stab here. Check map above to see if method was acceptable.

3.5.6 Save plot attributes

```
# create list of abbreviated subsection to append to plot attributes table
sub_attributes_for_plots <- subsection_attributes %>%
  select(ECOSUBCD, PROVINCE, SECTION, DOMAIN_NAM, DIVISION_N, PROVINCE_N,
         SECTION_NA, SUBSECTI_1,ATIntersec,
         SUBSECT_ABBR, SUBSECT_ABBR_ST, states)

## Plot attributes unique locations
plot_attributes_final <- plot_attributes |>
  right_join(sub_attributes_for_plots, by = join_by(ECOSUBCD)) |>
  select(-STATECD)

## Plot attributes by year
plot_attributes_drop_geom <- plot_attributes_final |>
  sf::st_drop_geometry()
plot_attributes_by_year_final <- tpa_plots_by_year |>
  select(YEAR, pltID) |>
  left_join(plot_attributes_drop_geom, by = join_by(pltID))

saveRDS(plot_attributes_final, file = paste0(save_location, "plot_attributes_locations
saveRDS(plot_attributes_by_year_final, file = paste0(save_location, "plot_attributes_by
```

3.5.6.1 join subsection attributes to eco

```
eco_final <- eco |>
  left_join(subsection_attributes)

saveRDS(eco_final, file = paste0(save_location, "eco.rds"))
```

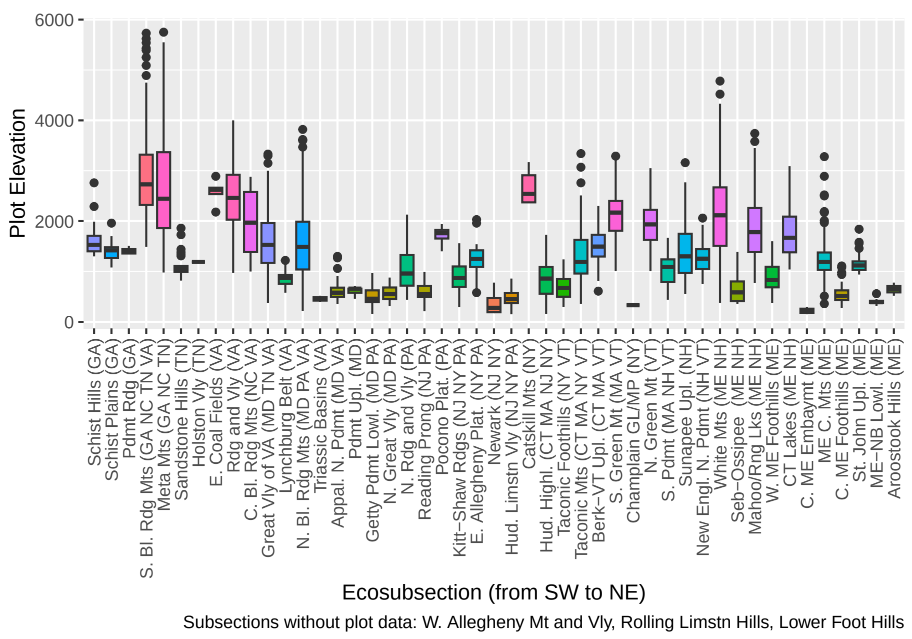
3.6 Check plot attributes final

Note the new Ecosubsection names and ability to detect subsections without FIA plots

```r
# Visualize plot distance x elevation with plot_attributes_final
plot_attributes_final_graph <- na.omit(plot_attributes_final)

plots_na <- plot_attributes_final |>
  filter(is.na(pltID)) |>
  pull(SUBSECT_ABBR)
plots_na <- paste("Subsections without plot data:", paste(plots_na, collapse = ", "))

plot_attributes_final_graph |>
  group_by(ECOSUBCD)|>
  mutate(median_dist = median(distance_m)) |>
  mutate(median_elev = median(ELEV)) |>
  ggplot(aes(x= fct_reorder(SUBSECT_ABBR_ST, distance_m),
             y=ELEV,
             fill = as.factor(median_elev))) +
  geom_boxplot() +
  theme(axis.text.x = element_text(angle = 90, vjust = 0.5, hjust=1),
        legend.position = 'none') +
  xlab("Ecosubsection (from SW to NE)") + ylab("Plot Elevation") +
  labs(caption = plots_na)
```



## Vizualize plots per year

```r
# Visualize number of plots per subsection per year
n_plots_per_year <- tpa_plots |>
  sf::st_drop_geometry() |>
  group_by(ECOSUBCD, YEAR) |>
  summarize(n_plots_per_year = n(), .groups = 'drop') |>
  right_join(subsection_attributes, by = join_by(ECOSUBCD))

n_plots_per_year_graph <- na.omit(n_plots_per_year)

n_plots_per_year_graph %>%
  mutate(highlight = case_when(n_plots_per_year == 1 ~ "1 plot",
                               n_plots_per_year == 2 ~ "2 plots",
                               .default = NA)) %>%
  ggplot(aes(x=forcats::fct_reorder(SUBSECT_ABBR_ST, median_distance),
             y=YEAR,
             size = n_plots_per_year,
             color = highlight)) +
  geom_count() +
  theme(axis.text.x = element_text(angle = 90, vjust = 0.5, hjust=1)) +
  scale_size_area() +
  scale_color_manual(values = c("red2", "cyan3", "grey50"),
                     breaks = c("1 plot", "2 plots")) +
  xlab("Ecosubsection (from SW to NE)") +
  labs(caption = plots_na)
```

Chapter 4

Make

4.1 Load data

Location where data was saved from previous chapter (clip)

```r
## default location:
load_location_default <- './clip_FIA/'
## custom location:
load_location_custom <- 'C:/Users/JJGross/Documents/R_projects/FIA_data/clip_FIA/'
#set variable
load_location <- load_location_custom
```

Load data and set as default argument variables

```r
## Load most recent (MR) Appalachian trail (at) dataset:
at_MR <- read_rds(file = paste0(load_location, "at_FIA_MR.rds"))
## Load entire Appalachian trail (at) dataset:
at <- read_rds(file = paste0(load_location, "at_FIA.rds"))
## Load eco
eco <- read_rds(file = paste0(load_location, "eco.rds"))

## Set default variables for rFIA function:
database_variable <- at
polys_variable <- eco
```

4.2 Save location

Choose where final tables will be saved:

```r
## default location:
save_default <- './summary_data/'
## Alternatively, enter specific custom location:
save_custom <- 'C:/Users/JJGross/Documents/R_projects/FIA_data/summary_data/'

save_location <- save_custom

## Create directory if it doesn't already exist
if (!dir.exists(save_location)) {dir.create(save_location)}
```

4.3 Metrics

Derive population estimates (at the plot- or the ecoregion-scale) for each forest health metric:

Live tree abundance

- tpa
- tpa_spp
- tpa_spp_sizecl
- biomass

Species diversity of live trees

- diversity

Tree vital rates

- growth

Forest demographic rates

- mortality

Regeneration abundance

- seedlings
- seedlings_spp
- saplings
- saplings_spp
- saplings_sizecl

Snags

- snag_abundance
- dead_and_live_tpa
- snag_volume

Down woody material

- downwoody

Invasive plant abundance

- invasive

Stand structural stage distributions

- ss

Each of the following subsections illustrates how to use the variety of functions in `rFIA` to calculate these metrics.

4.3.1 Argument Variables

Are plot-level records desired within each `SUBSECTION`? If so, run the entire `03-make.Rmd` with `by_plot_variable` set to `TRUE`. Otherwise, leave `by_plot_variable <- FALSE` to aggregate FIA plots at `SUBSECTION` level.

- `by_plot_variable <- FALSE` means tables are summarized at SUBSECTION-level
- `by_plot_variable <- TRUE` means tables are summarized at the FIA plot-level

```
by_plot_variable <- FALSE
```

- `most_recent <- FALSE` means all FIA plots are included in the data tables
- `most_recent <- TRUE` means only the most recent FIA plots are included

```
most_recent <- TRUE
```

The code chunk below automatically updates argument variables and folder save location based on above user input (blue boxes) to `by_plot_variable` and `most_recent`. Code chunk also specifies the number of processing cores to use during the execution of rFIA functions. See the rFIA webpage Tips for working with Big Data for more information.

```
## save tables to new folder if by_plot_variable is TRUE
if (by_plot_variable == TRUE) {
  # new save location folder
  ifelse(!dir.exists(file.path(save_location)), dir.create(file.path(save_location)), 
  save_location <- paste0(save_location, "by_plot_variable/")

}

## if 'most_recent == TRUE', than change database_variable to 'at_MR' database
## and create new folder to save MR tables to.
if (most_recent == TRUE) {
  database_variable <- at_MR
  # new save location folder
  ifelse(!dir.exists(file.path(save_location, "most_recent")),
         dir.create(file.path(save_location, "most_recent")), FALSE)
  save_location <- paste0(save_location, "most_recent/")
}

## default number of processing cores to utilize in rFIA function arguments below.
cores_variable <- parallel::detectCores()-2
```

4.3.2 Population estimate method

Functions in `rFIA` can be used to derive population estimates of forest data using 5 unique estimators (i.e. different methods for panel combination) using the `method` argument in each function. For the purposes of forest population estimation for APPA, only the `method = ANNUAL` argument is used (i.e. no panel combination). For more information on panel combinations see the rFIA Alternative design-based estimators page.

```
## Set method variable for rFIA function arguments here:
method_variable <- "ANNUAL"
```

Note that when using `method = "ANNUAL"` all `rFIA::` functions result with the following error message:

"Bad stratification, i.e., strata too small to compute variance of annual panels. If you are only interested in totals and/or ratio estimates, disregard this. However, if interested in variance (e.g., for confidence intervals) try using method ="TI"."

4.4 Live tree abundance

The rFIA::tpa() function Produces tree per acre (TPA) and basal area per acre (BAA) estimates from FIA data, along with population totals for each variable. Options to group estimates by species, size class, and other variables defined in the FIADB.

Argument `treeDomain = DIA \>= 5` results in estimates only utilizing tree diameters above 5 inches DBH.

4.4.1 tpa

```r
## Trees per Acre (species and size classes lumped)
tpa <- rFIA::tpa(
  db = database_variable,
  byPlot = by_plot_variable,
  polys = polys_variable,
  method = method_variable,
  nCores = cores_variable,
  returnSpatial = TRUE,
  treeType = "live",
  totals = TRUE,
  treeDomain = DIA >= 5
)
saveRDS(tpa, paste0(save_location, "tpa.rds"))
```

4.4.2 tpa_spp

```r
## Trees per Acre by species (size-classes lumped)
tpa_spp <- rFIA::tpa(
  db = database_variable,
  byPlot = by_plot_variable,
  polys = polys_variable,
  method = method_variable,
  nCores = cores_variable,
  returnSpatial = TRUE,
  treeType = "live",
  totals = TRUE,
  treeDomain = DIA >= 5,
  bySpecies = TRUE,
  bySizeClass = FALSE
)
saveRDS(tpa_spp, paste0(save_location, "tpa_spp.rds"))
```

4.4.3 tpa_spp_sizecl

```
## Trees per Acre (by species and size-class)
tpa_spp_sizecl <- rFIA::tpa(
  db = database_variable,
  byPlot = by_plot_variable,
  polys = polys_variable,
  method = method_variable,
  nCores = cores_variable,
  returnSpatial = TRUE,
  treeType = "live",
  totals = TRUE,
  treeDomain = DIA >= 5,
  bySpecies = TRUE,
  bySizeClass = TRUE
)
saveRDS(tpa_spp_sizecl, paste0(save_location, "tpa_spp_sizecl.rds"))
```

4.4.4 biomass

The rFIA::biomass() function produces estimates of volume (cu.ft./acre), biomass (tons/acre), and carbon (tons/acre) with options to group estimates by species, size class, and other variables defined in the FIADB.

```
## biomass (by species and size-class)
# biomass <- rFIA::biomass(db = database_variable,
#            byPlot = by_plot_variable,
#            polys = polys_variable,
#            method = method_variable,
#            nCores = cores_variable,
#            returnSpatial = TRUE,
#            treeType = "live",
#            totals = TRUE,
#            treeDomain = DIA >= 5,
#            bySpecies = TRUE,
#            bySizeClass = TRUE)
# saveRDS(biomass, paste0(save_location, "biomass.rds"))
```

Note that the `rFIA::biomass()` function currently produces the following error and needs attention from package developers.

Error in fcase(is.na(DIA), NA_real_, !is.na(DRYBIO_WDLD_SPP), DRYBIO_WDLD_SPP/(jTotal - : object 'DRYBIO_WDLD_SPP' not found

4.5 Species diversity of live trees

The rFIA::diversity() function produces estimates of diversity from FIA data. Returns Shannon's Index (H), Shannon's Equitability (Eh), and Richness (S) for alpha (mean/SE of stands), beta, and gamma diversity. Default behavior estimates species diversity, using TPA as a state variable and Species Code (SPCD) to groups of individuals.

4.5.1 diversity

```
## Diversity
diversity <- rFIA::diversity(
  db = database_variable,
  byPlot = by_plot_variable,
  polys = polys_variable,
  method = method_variable,
  nCores = cores_variable,
  returnSpatial = TRUE,
  treeType = "live",
  totals = TRUE,
  treeDomain = DIA >= 5
)
saveRDS(diversity, paste0(save_location, "diversity.rds"))
```

4.6 Tree growth rates

The rFIA::vitalRates() function computes estimates of average annual DBH (inches/ yr), basal area (sq. ft./ yr), biomass (short tons/ yr), and net volume (cu. ft./yr) growth rates for individual stems, along with average annual basal area and net volume growth per acre.

To calcuate by size class include argument `bySizeClass =TRUE`

To estimate net growth rates (include trees that have recruited or died in estimates), use `treeType = 'all'` (default)

To exclude stems that died or recruited into the population between plot measurements set `treeType = 'live'`

4.6.1 growth

```r
## Tree DBH growth
growth <- rFIA::vitalRates(
  db = database_variable,
  byPlot = by_plot_variable,
  polys = polys_variable,
  method = method_variable,
  nCores = cores_variable,
  returnSpatial = TRUE,
  treeType = "live",
  totals = TRUE,
  treeDomain = DIA >= 5,
  bySpecies = TRUE,
  variance = TRUE # note error message: "Bad stratification, i.e., strata too small to
)
saveRDS(growth, paste0(save_location, "growth.rds"))
```

4.7 Forest demographic rates

The rFIA::growMort() function estimates of annual growth, recruitment, natural mortality, and harvest rates, along with population estimates for each variable.

Recruitment events are defined as when a live stem which is less than 5 inches DBH at time 1, grows to or beyond 5 inches DBH by time 2. This does NOT include stems which grow beyond the 5-inch diameter criteria and are then subject to mortality prior to remeasurement. Natural mortality is defined as when a live stem is subject to non-harvest mortality between successive measurement periods. Finally, harvest is defined as when a live stem is cut and removed between successive measurements.

4.7.1 mortality

To estimate mortality per species:

```r
## mortality
mortality <- rFIA::growMort(
  db = database_variable,
  byPlot = by_plot_variable,
  polys = polys_variable,
  method = method_variable,
  nCores = cores_variable,
  returnSpatial = TRUE,
  treeType = "all", #default, includes all stems, live and dead
  totals = TRUE,
  treeDomain = DIA >= 5,
  bySpecies = TRUE,
  variance = TRUE # note error message: "Bad stratification, i.e., strata too small to compute va
)
saveRDS(mortality, paste0(save_location, "mortality.rds"))
```

Column `mortality$MORT_PERC` results in `Inf` for each row. Suspected error with `rFIA::growMort()` because none of the tables produced had any values in `MORT_PREC`

4.8 Regeneration abundance

The rFIA::seedlings() and rFIA::tpa() functions are used to estimate regeneration of seedlings and sapling, respectively. Note that querying saplings can be achieved within the `rFIA::tpa()` function by setting the tree domain argument `treeDomain = DIA < 5`. Stems below 1 inch DBH are not included in output of `rFIA::tpa()` because FIA seedlings (<1 inch DBH) are sampled differently and therefore queried using the distinct `rFIA::seedlings()` function.

4.8.1 seedlings

Seedling abundance (trees per acre) is computed using the distinct `rFIA::seedlings()` function, not the `rFIA::tpa()` function.

The FIA glossary defines seedlings as: "Live trees smaller than 1.0 inch (2.5 cm) d.b.h./d.r.c. that are at least 6 inches (15.2 cm) in height for softwoods and 12-inches (30.5 cm) in height for hardwoods."

```
## Trees per acre of seedlings (<1 inch DBH)
seedlings <- rFIA::seedling(
  db = database_variable,
  byPlot = by_plot_variable,
  polys = polys_variable,
  method = method_variable,
  nCores = cores_variable,
```

```r
  returnSpatial = TRUE,
  totals = TRUE
)
saveRDS(seedlings, paste0(save_location, "seedlings.rds"))
```

4.8.2 seedlings_spp

```r
## Trees per acre of seedlings by species
seedlings_spp <- rFIA::seedling(
  db = database_variable,
  byPlot = by_plot_variable,
  polys = polys_variable,
  method = method_variable,
  nCores = cores_variable,
  returnSpatial = TRUE,
  totals = TRUE,
  bySpecies = TRUE # by species
)
saveRDS(seedlings_spp, paste0(save_location, "seedlings_spp.rds"))
```

4.8.3 saplings

Saplings must use `rFIA::tpa()` function with `treeDomain = DIA < 5` argument To estimate sapling size class (1 to 4.9 inch DBH).

```r
## Trees per acre of saplings (trees greater than 1 inch DBH, and less than 5 inch DBH
saplings <- rFIA::tpa(
  db = database_variable,
  byPlot = by_plot_variable,
  polys = polys_variable,
  method = method_variable,
  nCores = cores_variable,
  returnSpatial = TRUE,
  treeType = "live",
  totals = TRUE,
  treeDomain = DIA < 5
)
saveRDS(saplings, paste0(save_location, "saplings.rds"))
```

4.8.4 saplings_spp

```
## Trees per acre of saplings by species
saplings_spp <- rFIA::tpa(
  db = database_variable,
  byPlot = by_plot_variable,
  polys = polys_variable,
  method = method_variable,
  nCores = cores_variable,
  returnSpatial = TRUE,
  treeType = "live",
  totals = TRUE,
  treeDomain = DIA < 5,
  bySpecies = TRUE, # by species
  bySizeClass = FALSE
)
saveRDS(saplings_spp, paste0(save_location, "saplings_spp.rds"))
```

4.8.5 saplings_sizecl

```
## Trees per acre of saplings by size class
saplings_sizecl <- rFIA::tpa(
  db = database_variable,
  byPlot = by_plot_variable,
  polys = polys_variable,
  method = method_variable,
  nCores = cores_variable,
  returnSpatial = TRUE,
  treeType = "live",
  totals = TRUE,
  treeDomain = DIA < 5,
  bySpecies = FALSE,
  bySizeClass = TRUE # by size class
)
saveRDS(saplings_sizecl, paste0(save_location, "saplings_sizecl.rds"))
```

4.9 Snags

rFIA::tpa() and rFIA::biomass() can be used to estimate snag abundance, volume, and percentage.

4.9.1 snag_abundance

`treeType = "dead"` `bySizeClass = TRUE`

```
## Snags per acre
snag_abundance <- rFIA::tpa(
  db = database_variable,
  byPlot = by_plot_variable,
  polys = polys_variable,
  method = method_variable,
  nCores = cores_variable,
  returnSpatial = TRUE,
  treeType = "dead",
  totals = TRUE,
  treeDomain = DIA >= 5,
  bySizeClass = TRUE
)

saveRDS(snag_abundance, paste0(save_location, "snag_abundance.rds"))
```

4.9.2 dead_and_live_tpa

`treeType = "all"` includes both live and dead trees in calculations. Use `grpBy = STATUSCD` to avoid lumping Live and Dead tree.

Table 4.1: STATUSCD (From The FIA Database User Guide)

STATUSCD	Description
0	No status - Tree is not presently in the sample (remeasurement plots only). Tree was incorrectly tallied at the previous inventory, currently not tallied due to definition or procedural change, or is not tallied because it is located on a nonsampled condition (e.g., hazardous or denied). RECONCILECD = 5-9 required for remeasured annual inventory data but not for periodic inventory data.
1	Live tree
2	Dead tree
3	Removed - Cut and removed by direct human activity related to harvesting, silviculture or land clearing. This tree is assumed to be utilized.

```
## dead and live trees - to calculate percent snags
dead_and_live_tpa <- rFIA::tpa(
  db = database_variable,
  byPlot = by_plot_variable,
  polys = polys_variable,
  method = method_variable,
  nCores = cores_variable,
  returnSpatial = TRUE,
  treeType = "all",
  grpBy = STATUSCD,
  totals = TRUE,
  treeDomain = DIA >= 5,
  bySizeClass = TRUE
)

dead_and_live_tpa <- dead_and_live_tpa |>
  filter(STATUSCD == 1 | STATUSCD == 2) |> #only interested in live and dead
  mutate(tree_status = case_when(STATUSCD == 1 ~ "Live",
                                 STATUSCD == 2 ~ "Dead",
                                 #anything else will result in NA
                                 TRUE ~ NA))

saveRDS(dead_and_live_tpa, paste0(save_location, "dead_and_live_tpa.rds"))
```

4.9.3 snag_volume

```
# ## Snag Volume
# snag_volume <- rFIA::biomass(
#   db = database_variable,
#   byPlot = by_plot_variable,
#   polys = polys_variable,
#   method = method_variable,
#   nCores = cores_variable,
#   returnSpatial = TRUE,
#   treeType = "dead",
#   totals = TRUE,
#   treeDomain = DIA >= 5
# )
# saveRDS(snag_volume, paste0(save_location, "snag_volume.rds"))
```

Note that the `rFIA::biomass()` function currently produces the following error and needs attention from package developers.

Error in fcase(is.na(DIA), NA_real_, !is.na(DRYBIO_WDLD_SPP), DRYBIO_WDLD_SPP/(jTotal - : object 'DRYBIO_WDLD_SPP' not found

4.10 Down woody material

rFIA::dwm() produces estimates of down woody material stocks. Estimates are returned by fuel class (duff, litter, 1HR, 10HR, 100HR, 1000HR, piles) for application in fuels management.

1HR fuels: small, fine woody debris **10HR fuels**: medium, fine woody debris **100HR fuels**: large, fine woody debris **1000HR fuels**: coarse woody debris and slash piles **duff**: O horizon; all unidentifiable organic material above mineral soil, beneath litter **litter**: identifiable plant material which is downed and smaller than 10HR fuel class (1HR class includes standing herbaceous material).

4.10.1 downwoody

```r
## Down woody material
downwoody <- rFIA::dwm(
  db = database_variable,
  byPlot = by_plot_variable,
  polys = polys_variable,
  method = method_variable,
  nCores = cores_variable,
  returnSpatial = TRUE,
  totals = TRUE
)
saveRDS(downwoody, paste0(save_location, "downwoody.rds"))
```

4.11 Invasive plant abundance

The rFIA::invasive() function produces estimates of the areal coverage (%) of invasive species and frequency of plots invasive species were detected in.

4.11.1 invasive

```r
## invasive
invasive <- rFIA::invasive(
  db = database_variable,
  byPlot = by_plot_variable,
  polys = polys_variable,
  method = method_variable,
  nCores = cores_variable,
  returnSpatial = TRUE,
  totals = TRUE,
  variance = TRUE # note error message: "Bad stratification, i.e., strata too small to compute va
)
saveRDS(invasive, paste0(save_location, "invasive.rds"))
```

4.12 Stand structural stage distributions

The rFIA::standStruct() function estimates forest structural stage distributions as percent forested land area in pole, mature, late, and mosaic stages and returns the stand structural stage distribution of an area of forest/timberland from FIA data.

- **Description:** Estimates of forest structural stage distributions as percent forested land area in pole, mature, late, and mosaic stages
  - Diameter Classes:
    - *Pole*: 11 - 25.9 cm
    - *Mature*: 26 - 45.9 cm
    - *Large*: 46+ cm
  - Structural Stage Classification:
    - *Pole Stage*: > 67% BA in pole and mature classes, with more BA in pole than mature.
    - *Mature Stage*: > 67% BA in pole and mature classes, with more BA in mature than pole OR > 67% BA in mature and large classes, with more BA in mature.
    - *Late-Successional Stage:*: > 67% BA in mature and large classes, with more in large.
    - *Mosiac:*: Any plot not meeting above criteria.

4.12.1 ss

```r
## stand structural stage
ss <- rFIA::standStruct(
  db = database_variable,
  byPlot = by_plot_variable,
  polys = polys_variable,
  method = method_variable,
  nCores = cores_variable,
  returnSpatial = TRUE,
  totals = TRUE
)
saveRDS(ss, paste0(save_location, "ss.rds"))
```

Chapter 5

References

Bechtold, W.A.; Patterson, P.L., eds. 2005. The Enhanced Forest Inventory and Analysis Program - National Sampling Design and Estimation Procedures. Gen. Tech. Rep. SRS - 80. Asheville, NC: U.S. Department of Agriculture, Forest Service, Southern Research Station. 85 p. https://www.srs.fs.usda.gov/pubs/gtr/gtr_srs080/gtr_srs080.pdf

Cleland, D.T.; Freeouf, J.A.; Keys, J.E.; Nowacki, G.J.; Carpenter, C.A.; and McNab, W.H. 2007. Ecological Subregions: Sections and Subsections for the conterminous United States. Gen. Tech. Report WO-76D [Map on CD-ROM] (A.M. Sloan, cartographer). Washington, DC: U.S. Department of Agriculture, Forest Service, presentation scale 1:3,500,000; colored. https://doi.org/10.2737/WO-GTR-76D

Dieffenbach, F. (2018). Appalachian National Scenic Trail forest health monitoring protocol, Natural Resource Report NPS/NETN/NRR—2018/1804. National Park Service, Fort Collins, Colorado. https://irma.nps.gov/DataStore/DownloadFile/610353

Stanke, H., Finley, A. O., Weed, A. S., Walters, B. F., & Domke, G. M. (2020). rFIA: An R package for estimation of forest attributes with the US Forest Inventory and Analysis database. Environmental Modelling & Software, 127, 104664. https://doi.org/10.1016/j.envsoft.2020.104664

Woodall, C.; Monleon, V.J., eds. 2007. Sampling Protocol, Estimation, and Analysis Procedures for the Down Woody Materials Indicator of the FIA Program. Gen. Tech. Rep. NRS - 22. Newtown Square, PA: U.S. Department of Agriculture, Forest Service, Northern Research Station. https://www.nrs.fs.fed.us/pubs/gtr/gtr_nrs22.pdf

FIA Database User Guide: https://www.fia.fs.fed.us/library/database-documentation/

Chapter 6

Revision History

Version numbers will be incremented by a whole number (e.g., Version 1.3 to 2.0) when a change is made that significantly affects requirements or procedures. Version numbers will be incremented by decimals (e.g., Version 1.06 to Version 1.07) when there are minor modifications that do not affect requirements or procedures included in the protocol. Add rows as needed for each change or set of changes tied to an updated version number.

Revision History Log

Version #	Date	Revised by	Changes	Justification
1.00	July 2017	Fred Dieffenbach	Initial Version	

Version #	Date	Revised by	Changes	Justification
2.00	May 2020	Aaron Weed	Updated and merged prior versions of SOPs 1 and 2 into this SOP to incorporate new R-based FIA data workflow utilizing ther `rFIA` package.	These changes were made because former workflow using MS Access downloads from FIA are no longer supported and because analysis workflow for estimating metrics was updated.

Version #	Date	Revised by	Changes	Justification
3.00	Feb 2024	Jacob Gross & Aaron Weed	Revised so SOP and code to download, clip, and summarize the data tables are one and the same. Edits for clarity.	Former SOP was was like a tutorial on how to use rFIA functions. It was separate from APPA forest health report and quickly became out of date due to evolving nature of the rFIA package and the APPA Forest Health Rmarkdown report. New approach reduces the number of files/documents needing managed. And tables generated from SOP should feed directly into forest health report.