Proceedings of the

Workshop on Data analysis with R for Lao PDR's fourth National Forest Inventory cycle

01-05 Sep 2025, FIPD office, Vientiane, Lao PDR

Table of contents

Context

A new training course: "**Training workshop on Data analysis with R for Lao PDR's fourth National Forest Inventory cycle**", was organised for FIPD from 01 to 05 September 2025, in FIPD office in Vientiane (Lao PDR). Its goal was continue capacity building of FIPD on national forest inventory data analysis and to present the calculation chain for the cycle 4 updated method.

The training objectives were to:

1. Continue build capacities of FIPD in forest data analysis with R and Rstudio.
2. Present and practice on the code base used for analyzing the 4th National Forest Inventory of Lao PDR, implemented in 2024 and 2025.
3. Explore opportunities to build a R community with research institutions and the civil society to help each other build analytical skills and deal with complex data analysis in the forest, ecology and environment sectors.

The workshop agenda and participants list are shown in Figure 2.

This document contains all the code presented during the training and the exercises with solutions.

Initial Setup

Before loading data and performing the analysis, we needed to prepare the environment by loading packages and setting a few options. Here we didn't load packages directly, but we sourced a script that loaded them.

```r
## Useful when sourcing script from Quarto doc.
if (!require(here)) install.packages("here"); library(here)

source(here("R/00-load-packages.R"))
source(here("R/fct-nfi-aggregate3-new.R"))
```

1 Session 01: Objectives

Session 1 presented the detailed objectives of the training, as a mixture of basic(refresher) and advanced analysis in R. The technical analysis objectives were to cover: Basic calculations, Aggregation with simple sampling, Ratio estimators, and Double sampling for post-stratification.

The institutional objective was to explore a pool of resources available in other institutions to improve analytical capacities of FIPD. This included academia and research institutions in Lao PDR to bring them together for broader technical exchange in R and exploring their potential support to FIPD.

Finally, the workshop aimed at defining a work plan of continuous training to bring in additional pools and calculations.

To achieve the above objectives, the training focused on:

1. Read data into R and visualize trees characteristics
2. Manipulate tables (merge, split, rename)
3. Detect outliers and correct the data
4. Aggregate from tree to plot level
5. Aggregate plot level data to LC types
6. Aggregate to sub-population and national level

2 Session 02: Updated calculation chain for the NFI cycle 4 carbon analysis

The NFI cycle 4 of LAO PDR was based on an updated sampling methodology, with a **double-stage sampling for post-stratification and ratio estimators**. This session went over the core part of the NFI design and calculations for mean carbon related variables for each land cover.

2.1 Sampling design

3 Session 03: Discussion and remarks on the updated calculations

TBD

4 Session 04: Refresher on R for forest data analysis

TBD

5 Session 05: data preparation

Goals

1. Load initial tables and visualize core data
2. Correct errors in subplot table (duplicates in subplot_id)

5.1 Load harmonized tables and ancillary data

5.1.1 Load subplot, lcs, tree tables from NFI

We can read tables or CSV files from the computer with the function `read_csv()`. We can store the initial table in objects with the suffix '_init' to keep the initial version in the environment.

Example: read the table 'training_subplot.csv' from the computer into the object `subplot_init`:

```
subplot_init <- read_csv("data/training_subplot.csv", show_col_types = FALSE)
```

Load the tables 'training_tree.csv' and 'training_lcs.csv' in the Exercise 5.1.

Exercise 5.1 (~ Load 'training_tree.csv' and 'training_lcs.csv' tables)**.**

- Load the table 'training_lcs.csv' in the R objects 'lcs_init'.
- Load the table 'training_tree.csv' in the R objects 'tree_init'.

Answer to Exercise 5.1:

```
## !!! Solution
lcs_init <- read_csv("data/training_lcs.csv", show_col_types = FALSE)
tree_init <- read_csv("data/training_tree.csv", show_col_types = FALSE)
```

5.2 Load ancillary data

Ancillary data are not collected in the field but useful for the analysis of field data.

In the training we use plot level environment factor (E) from Chave et al. 2014 and the NFI Phase 1 plot data. Ancillary data are loaded into a list to keep the environment tidy.

```
anci <- list()
anci$ph1    <- read_csv("data/training_anci_phase1.csv", show_col_types = FALSE)
anci$plot_E <- read_csv("data/training_anci_plotE.csv", show_col_types = FALSE)
```

5.3 Visualize tree locations

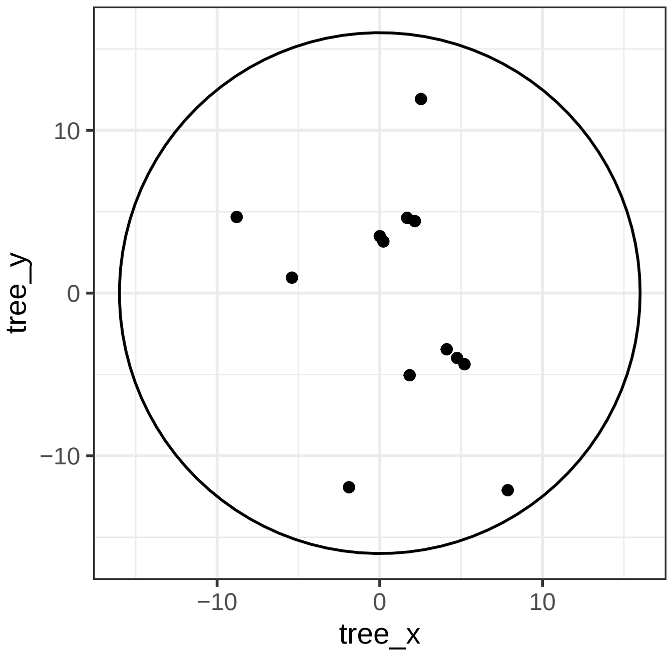
We visualize tree locations from the `tree_init` table with `ggplot()` and `geom_point()` to see if any potential error.

Step-by-step for one subplot:

- Define `circ16` as tibble of 100 rows to draw a circular plot in ggplot, based on a radius and an angle "theta" θ in radian.

- Draw a ggplot with trees from the table `tree_init` in plot 631 and subplot “C”, using `filter()`,
- Use `tree_x` and `tree_y` to show trees based on their location from the subplot center,
- Add plot boundary with `circ16`.

```r
circ16 <- tibble(
  theta = seq(0, 2*pi, length = 100),
  x = 16 * cos(theta),
  y = 16 * sin(theta)
)

tree_init |>
  filter(plot_no == 631, subplot_no == "C") |>
  ggplot(aes(x = tree_x, y = tree_y)) +
  geom_point() +
  geom_path(data = circ16, aes(x = x, y = y)) +
  coord_fixed()
```



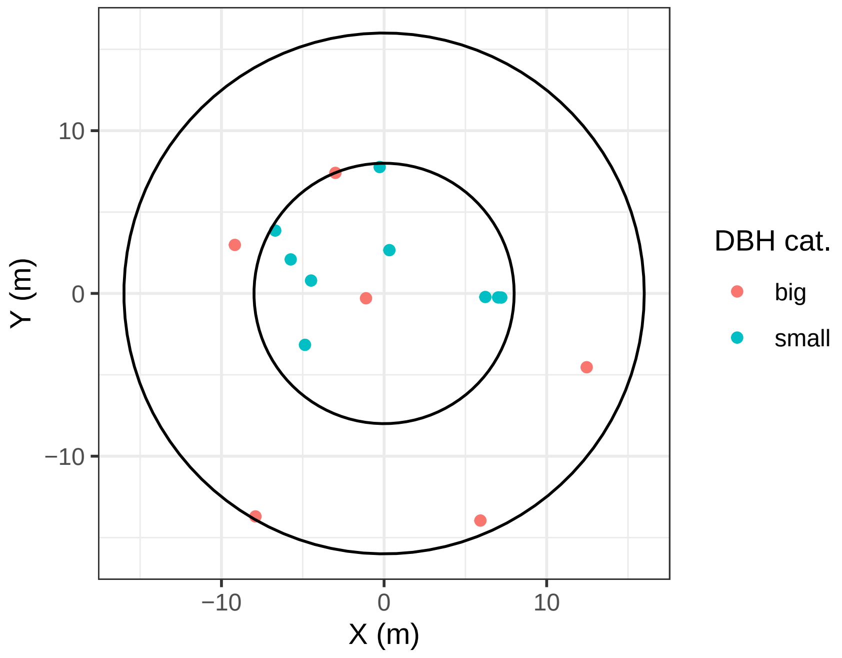
Adapt the previous code to make a tree location map for the subplot ‘1A’ and add different colors based on tree DBH size in Exercise 5.2.

Exercise 5.2 (~ Make a tree location figure).

- create the table `circ08` similar to `circ16` but for a 8m radius circle.
- Filter within `tree` the trees from subplot ‘1A’ with `filter()`.
- Use `mutate()` and `if_else()` to add a column `tree_dbh_cat` that has value “small” for trees with DBH < 30 cm and “big” for the other trees.
- Make the figure of tree location with the 8m and 16m radius circles and change the color of trees based on `tree_dbh_cat`.
- Use `labs()` to create clean labels in English or Lao.

Answer to Exercise 5.2:

```r
## !!! Solution
circ08 <- tibble(
  theta = seq(0, 2*pi, length = 100),
  x = 8 * cos(theta),
  y = 8 * sin(theta)
)

tree_init |>
  filter(plot_no == 1, subplot_no == "A") |>
  mutate(tree_dbh_cat = if_else(tree_dbh < 30, "small", "big")) |>
  ggplot(aes(x = tree_x, y = tree_y)) +
  geom_point(aes(color = tree_dbh_cat)) +
  geom_path(data = circ08, aes(x = x, y = y)) +
  geom_path(data = circ16, aes(x = x, y = y)) +
  coord_fixed() +
  labs(x = "X (m)", y = "Y (m)", color = "DBH cat.")
```



Visualizations can also be used to detect outliers. We use the example of small trees outside the nested subplot circle of 8m radius as guided exercise in Exercise 5.3.

Exercise 5.3 (~ (Optional) detect outliers on graph)**. Part 1: Find outliers**

- Make a figure with tree location of all trees smaller than or equal to 30 cm with the 2 circles of 8 and 16 meters.
- Use filter() to get all trees with DBH <= 30 cm.
- Make the ggplot of tree locations

Part 2: highlight outliers
We can highlight outliers by making a specific table for them.

- Create the table `outliers` containing trees with DBH <= 30 and distance > 8. Use `filter()`.

- Make the same figure as before and add a `geom_point()` with `data = outliers`, `shape = 23` and `size = 4` .

TIP: here we added `alpha = 0.2` to the geometry of the trees. It make tree point semi-transparent and helps to detect potential measurement issues if we would detect patches with few trees.

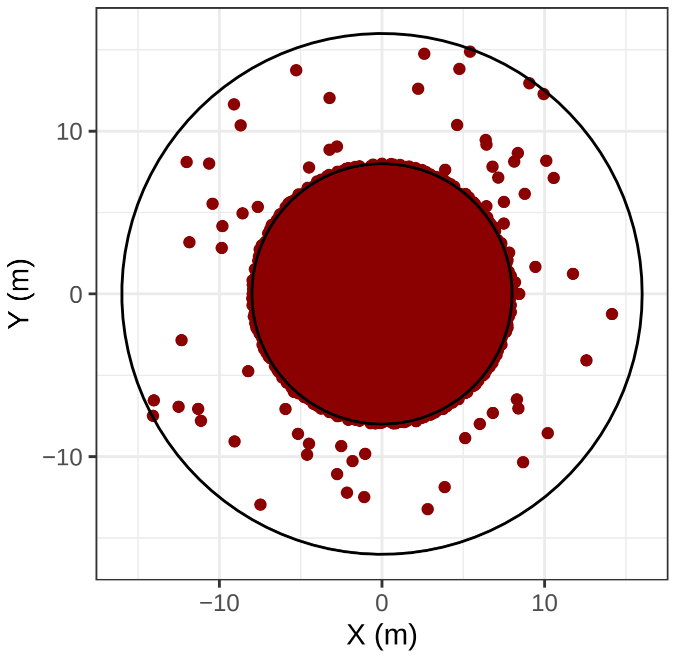
Part 3: explanations

Now are these outliers measurement errors?

- Make the same graph as this exercise's part 1, but this time with tree DBH < 30 and **not** DBH <= 30.

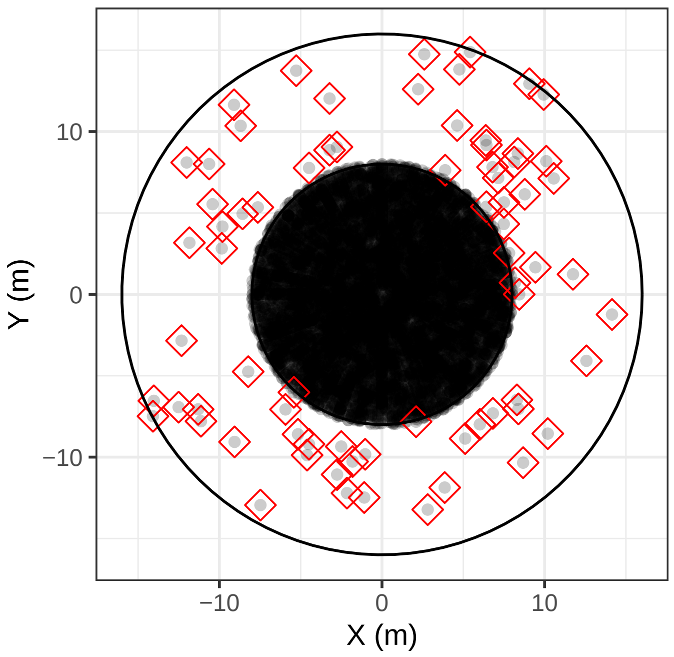
Type here answers to Exercise 5.3 part1:

```r
## !!! Solution
tree_init |>
  filter(tree_dbh <= 30) |>
  ggplot(aes(x = tree_x, y = tree_y)) +
  geom_point(col = "darkred") +
  geom_path(data = circ08, aes(x = x, y = y)) +
  geom_path(data = circ16, aes(x = x, y = y)) +
  coord_fixed() +
  labs(x = "X (m)", y = "Y (m)")
```



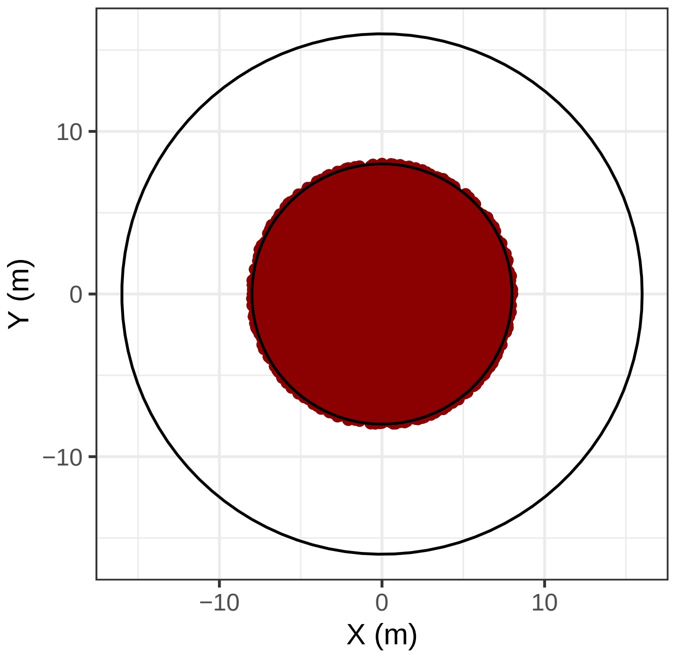
Type here answers to Exercise 5.3 part2:

```r
## !!! Solution
outliers <- tree_init |> filter(tree_dbh <= 30, tree_distance > 8)

tree_init |>
  filter(tree_dbh <= 30) |>
  ggplot(aes(x = tree_x, y = tree_y)) +
  geom_point(alpha = 0.2) +
  geom_point(data = outliers, col = "red", shape = 23, size = 4) +
  geom_path(data = circ08, aes(x = x, y = y)) +
  geom_path(data = circ16, aes(x = x, y = y)) +
  coord_fixed() +
  labs(x = "X (m)", y = "Y (m)")
```



Type here answers to Exercise 5.3 part3:

```
## !!! Solution
tree_init |>
  filter(tree_dbh < 30) |>
  ggplot(aes(x = tree_x, y = tree_y)) +
  geom_point(col = "darkred") +
  geom_path(data = circ08, aes(x = x, y = y)) +
  geom_path(data = circ16, aes(x = x, y = y)) +
  coord_fixed() +
  labs(x = "X (m)", y = "Y (m)")
```



Conclusion of Exercise 5.3: the trees found outside the small circle were trees with DBH = 30 cm. They were not measurement errors, they were part of the big trees and therefore could be found outside the 8m radius circle.

5.4 Find and correct errors

One of the main issue found in the raw data was the entry errors of subplot and tree IDs. It is very important in Database Management Systems (DBMS) to have unique IDs for all tables. In NFI it is particularly important for joining information from one table to the other. For example, land cover information is recorded at subplot level and needs to be passed on to tree level to apply the correct allometric equations. These errors can be detected programatically but often have to be solved manually.

This section introduces how to use `group_by()` and `summarise()` to find duplicates in subplot IDs.

5.4.1 Identify duplicates in subplot IDs

Here we want to check within subplots if there are duplicates in subplot IDs (`subplot_id`) and plot numbers (`plot_no`) in the `subplot_init` table. The objective is to create a vector of subplot Ids that are duplicated. If there are no duplicate the vector will be of length 0.

step-by-step:

- Create a vector `vec_dup` and assign it the following code sequence.
- Group the subplot table by subplot ID with `group_by()` and use `summarise()` to count the number of subplots that have each ID inside a new column `count`. Use the function `n()` to count.
- Filter all the subplot IDs that have a count bigger than 1 with `filter()`.
- Pull the subplot IDs.

```r
vec_dup <- subplot_init |>
  group_by(subplot_id) |>
  summarise(count = n(), .groups = "drop") |>
  filter(count > 1) |>
  pull(subplot_id)
vec_dup
```

```
[1] "553D" "631C" "632C"
```

Notice the results as 3 subplots have duplicate `subplot_id`.

Now practice the same code sequence to check if there are duplicates in `plot_no` in the Exercise 5.4. Note that each plot ID should appears 4 times in the subplot table, one for each subplot, so duplicates are when `count > 4`.

Exercise 5.4 (~ Identify duplicates of plot number)**.**

- Create `vec_dup2` similarly to `vec_dup`, but this time looking for plot_no that have more than 4 subplot IDs, so you will need to group by `plot_no` and filter `count > 4` and pull `plot_no`.

There should be no duplicates of `plot_no`.

Answer to Exercise 5.4:

```r
## !!! Solution
vec_dup2 <- subplot_init |>
  group_by(plot_no) |>
  summarise(count = n(), .groups = "drop") |>
  filter(count > 4) |>
  pull(plot_no)
vec_dup2
```

```
numeric(0)
```

There should be no duplicates of `plot_no`.

5.4.2 Correct the subplot ID issues

To correct the duplicates in subplot_id, we filter the subplots from one of the duplicated IDs and visually check what is wrong. In case one ID is missing and one is duplicated we can use time stamps to identify which of the duplicates should be renamed.

Then we can use `mutate()` and `case_when()` to apply correction in R.

Step-by-step:

- Filter the subplots of one plot with duplicates: 631C.
- Visualize the table (run `View(tt)` or click `tt` in the Environment tab (top-right window in Rstudio).
  - In the case of '631C' there are 2 subplots C but no B. we can use time stamps to identify that `ONA_index` 109 should actually be for subplot B.

- Perform the correction using `case_when()` function for subplot_id 631C. At this stage only `ONA_index` is unique so it can be used to correctly identify the subplot to correct.

```r
tt <- subplot_init |> filter(plot_no == 631)

# View(tt)

subplot <- subplot_init |>
  mutate(
    subplot_no = case_when(
      subplot_id == "631C" & ONA_index == 109 ~ "B",
      ## ADD more corrections,
      TRUE ~ subplot_no
    )
  )
```

You can now copy/paste the above R chunk and complete the correction for the 2 other subplot IDs with duplicates in Exercise 5.5.

Note: we correct `subplot_no` now, but we will need to remake `subplot_id` once `subplot_no` is correct.

Exercise 5.5 (~ Correct duplicates in subplot IDs)**.**

- for each remaining duplicate, run the table `tt` with all subplots for the plot with duplicates
- Visualize `tt` and identify `ONA_index` of the subplot that should be corrected and what is the correct `subplot_no`.
- Copyu/paste the `subplot <- ...` sequence from above and fill in the `case_when()` with all the correction for the 3 duplicated subplots.

Solution to Exercise 5.5:

```r
## !!! Solution
subplot <- subplot_init |>
  mutate(
    subplot_no = case_when(
      subplot_id == "631C" & ONA_index == 109 ~ "B",
      subplot_id == "632C" & ONA_index == 113 ~ "B",
      subplot_id == "553D" & ONA_index == 265 ~ "C",
      TRUE ~ subplot_no
    )
  )
```

Now we can recreate `subplot_id` from the corrected `subplot_no`. Since `subplot_id` is a text field, we need to make it consistent for number of characters to ensure correct sorting of the rows. For this purpose we include texts "0" and "00" to make the number of characters consistent.

```r
subplot <- subplot |>
  mutate(
    subplot_id = case_when(
      plot_no < 10 ~ paste0("00", plot_no, subplot_no),
      plot_no < 100 ~ paste0("0", plot_no, subplot_no),
      TRUE ~ paste0(plot_no, subplot_no)
    )
  )
```

Now if we re-run the `vec_dup` sequence with `subplot` instead of `subplot_init`, `vec_dup` should be of length 0.

```r
vec_dup <- subplot |>
  group_by(subplot_id) |>
  summarise(count = n(), .groups = "drop") |>
  filter(count > 1) |>
  pull(subplot_id)
vec_dup
```

```
character(0)
```

5.4.3 Check outliers in tree table

The tree table we can check for outliers in numeric columns with the `summary()` function.

For example with DBH:

```r
summary(tree_init$tree_dbh)
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
  10.00   13.50   19.80   25.79   34.20  199.00 
```

There are no trees with DBH $<$ 10 cm or unrealistically big DBH, all good.

We can check for outliers in tree distance and azimuth in Exercise 5.6.

Exercise 5.6 (~ Check azimuth and distance).

- Check that azimuth is between 0 and 360.
- Check that distance is between 0 and 16.

Solution to Exercise 5.6:

```r
summary(tree_init$tree_azimuth)
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
      0      90     179     179     268     360 
```

```r
summary(tree_init$tree_distance)
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
  0.000   4.335   6.280   6.785   7.900  16.000 
```

5.5 Assigning clean data to objects

After resolving all errors, save the tables to the entity name without “_init” suffix. This way we have both the initial and corrected version in the environment. If tehre were no errors the inital tables can be passed directly to the corrected tables:

```r
tree <- tree_init
lcs <- lcs_init
```

Finally we can remove the temporary objects that won’t be used later.

```r
rm(vec_dup, vec_dup2, tt, outliers)
```

NOTE: use of temporary object list `tmp`.

In the main analysis the temporary objects are often stored in a list called `tmp`. This avoids cluttering the environment with temporary objects and makes it easy to remove them with `rm(tmp)`.

```r
## Create empty list
tmp <- list()

## Populate the list (example)
tmp$outliers <- tree_init |> filter(tree_dbh <= 30, tree_distance > 8)

## Remove temporary objects
rm(tmp)
```

6 Session 06: data joins

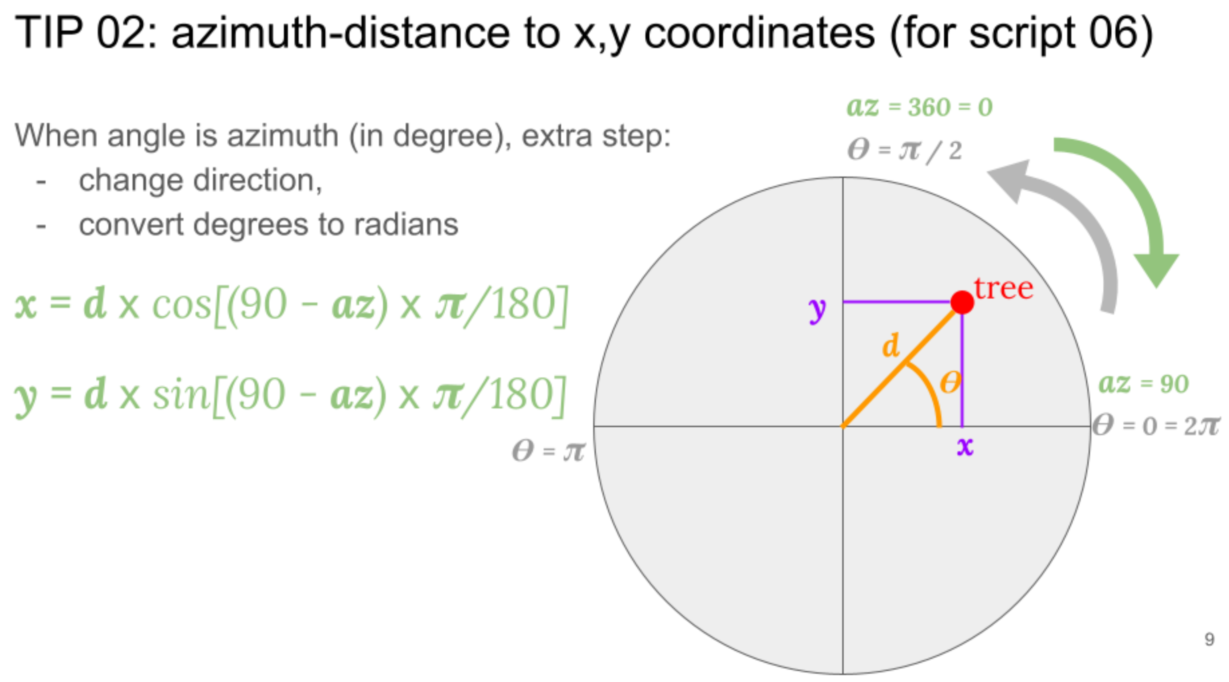
Goals:

1. Assign LCS number to trees
2. Join land cover and Chave E to trees

6.1 Assign LCS number to trees

Since the Land Cover Sections (LCS) are a mix of center square and outside disc portions, we need both tree position as distance d and azimuth az and as coordinates (x, y).

See the classroom presentation: "main calculations, TIP02" for explanations on converting distance d and angle θ to coordinates (x, y) for the subplot center, and converting distance d and azimuth az to coordinates (x, y).



TIP 02: azimuth-distance to x,y coordinates (for script 06)

When angle is azimuth (in degree), extra step:

- change direction,
- convert degrees to radians

$$x = d \times \cos[(90 - az) \times \pi/180]$$

$$y = d \times \sin[(90 - az) \times \pi/180]$$

Figure 1: Conversion of distance/azimuth to x, y

The R implementation is as follows:

```r
tree <- tree |>
  mutate(
    tree_x = cos((90 - tree_azimuth) * pi/180) * tree_distance,
    tree_y = sin((90 - tree_azimuth) * pi/180) * tree_distance,
    )
```

Once we have tree coordinates, we can assign LCS with `case_when()`, identifying trees in the center square first (with their coordinates), then the outer disc portion based on azimuth:

- To be inside the **center square**, trees must have their coordinates (x, y) both between -6 and 6 meters, or their absolute value smaller or equal to 6:

$$-6 \leq x \leq 6 \Leftrightarrow |x| \leq 6 - 6 \leq y \leq 6 \Leftrightarrow |y| \leq 6$$

- To be in one of the **outer disc portions**, trees must not be in the center square and be respectively positioned between NW and NE lines, NE and SE, SE and SW or SW and NW for the north, east, south and west disc portions. These lines' azimuth are 315, 45, 135, 215 degrees respectively for NW, NE, SE and SW lines.

R implementation:

```r
tree <- tree |>
  mutate(
    lcs_no = case_when(
      abs(tree_x) <= 6 & abs(tree_y) <= 6     ~ 1,
      tree_azimuth > 315 | tree_azimuth <=45  ~ 2,
      tree_azimuth >  45 & tree_azimuth <=135 ~ 3,
      tree_azimuth > 135 & tree_azimuth <=225 ~ 4,
      tree_azimuth > 225 & tree_azimuth <=315 ~ 5,
      TRUE ~ NA_integer_
    )
  )
```

We can now check if the code implementation worked with a visual check in Exercise 6.1.

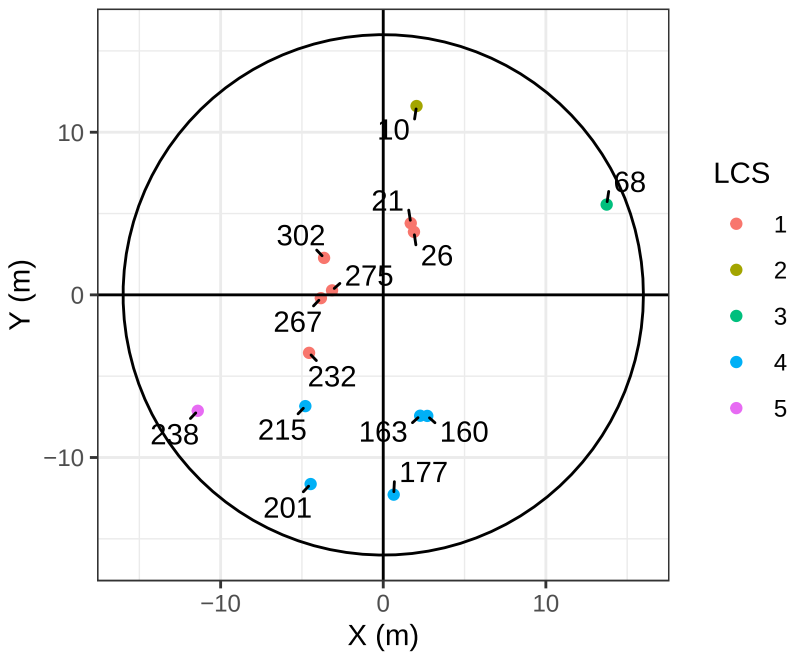
Exercise 6.1 (~ Figure of tree locations)**.**

- Filter the tree data for subplot '**123B**'.
- Make a base figure with `geom_point()`.
- Add subplot boundaries with `geom_path()` and `circ16`.
- **Add cardinal lines with `geom_hline(yintercept = 0)` and `geom_vline(xintercept = 0)`.**
- **Add azimuth to each tree as labels with `geom_label_repel()`.**
- Add `coord_fixed()` to make sure the x and y axis have the same unit length.

Check visually that the trees are positioned correctly based on their azimuth!

Answer to Exercise 6.1:

```r
## !!! Solution
tree |>
  filter(plot_no == 123, subplot_no == "B") |>
  ggplot(aes(x = tree_x, y = tree_y)) +
  geom_point(aes(color = as.character(lcs_no))) +
  geom_path(data = circ16, aes(x = x, y = y)) +
  geom_hline(yintercept = 0) +
  geom_vline(xintercept = 0) +
  geom_text_repel(aes(label = tree_azimuth), min.segment.length = 0) +
  coord_fixed() +
  labs(x = "X (m)", y = "Y (m)", color = "LCS")
```



We can observe that azimuths have been well converted to (x, y) since azimuth labels are in their correct quadrants (0-90, 90-180, 180-270 ad 270-360 degree respectively).

6.2 Join tables

6.2.1 Prepare tables to join

We want to join 2 tables to the tree data:

- Chave et al. 2014 environment factor E from the table `anci$plot_E`,
- Land cover code from the table `lcs`,

Sometimes we don't want all the columns from a tables to be passed on to `tree`. In this case we create a temporary tables with only the **key(s)** = the matching column(s) between two tables to join, and the information to pass.

For example, we don't want `lcs_name` from `lcs` to go to trees, we already have `lcs_no` as key between the two tables. We can either remove the undesired column or keep all other columns. In this case, removing is faster, but for sustainability, it is preferable to write the columns you want to keep so that when you re-open this document in a few weeks you see directly what is kept.

```r
tmp_lcs <- lcs |> select(-lcs_name)

## Same but preferable:
tmp_lcs <- lcs |> select(plot_no, subplot_no, lcs_no, lc_no, lc_code)
```

6.2.2 Make the joins

- **With changing object**

We join LCS (`tmp_lcs`) with the `tree` table using their common keys. In this case there are three keys: `plot_no`, `subplot_no` and `lcs_no`.

```
tree_join <- tree |>
  left_join(tmp_lcs, by = join_by(plot_no, subplot_no, lcs_no))
```

It is recommended to change object name when using join to avoid undesired suffixes. Undesired suffixes happen when we join the same tables twice or more by mistake. The second time we join two tables, since the new tables are already in the joined table, they will be passed again but this time with suffixes `.x` and `.y`. Observe undesired suffix in the Exercise 6.2.

Exercise 6.2 (~ Undesired suffixes).

- Create `tree_err` by joining `tree` and `tmp_lcs`
- Re-create `tree_err` by joining `tree_err` and `tmp_lcs`
- Run `names(tree_err)`

Answers to Exercise 6.2:

```
## !!! Solution
tree_err <- tree |>
  left_join(tmp_lcs, by = join_by(plot_no, subplot_no, lcs_no))
tree_err <- tree_err |>
  left_join(tmp_lcs, by = join_by(plot_no, subplot_no, lcs_no))
names(tree_err)
```

```
 [1] "plot_no"              "subplot_no"           "lcs_no"
 [4] "tree_no"              "tree_stem_no"         "tree_dbh"
 [7] "tree_species_code"    "tree_species_binomial" "tree_distance"
[10] "tree_azimuth"         "tree_x"               "tree_y"
[13] "lc_no.x"              "lc_code.x"            "lc_no.y"
[16] "lc_code.y"
```

- **With keeping the same object name**

Often time we don't want to change object names for a join if we have a long sequence of modifications ahead. We can join and keep the same object names, but in this case we have to **control the suffixes**.

Step-by-step:

1. Identify the **new column** passed from the table to join.
2. **Create these columns in the receiving table** with NAs using `mutate()`,
3. **Join the tables and add suffixes** "_rm" to the receiving table with NAs and " " to the joined columns,
4. **Remove the columns** that end with "_rm" (since they contains NAs).

```r
## 1. Columns to be passed are 'lc_no' and 'lc_code'
tree <- tree |>
  ## 2. Create these columns with NAs
  mutate(lc_no = NA, lc_code = NA) |>
  ## 3. join with controlled suffix names
  left_join(tmp_lcs, by = join_by(plot_no, subplot_no, lcs_no), suffix = c("_rm", "")) |>
  ## 4. remove initial columns with NAs
  select(-ends_with("_rm"))
```

Practice 'joins with controlled suffix' to join Chave E at plot level from `anci$plot_E` with the Exercise 6.3.

Exercise 6.3 (~ Join Chave E to the trees table).

- Identify which column from `anci$plot_E` to pass to `tree` and which column is the key
- from `tree`, create a column with NAs that has the same name as the column from `anci$plot_E` that we want in `tree`
- Join the 2 tables and control the suffixes (add `suffix = c("_rm", "")`)
- Remove the initial column with NAs (they have now suffix "_rm")

After the join, check that there is no NAs in `plot_E` from `tree` with the function `summary()`.

Answers to Exercise 6.3:

```r
## Your code
# names(anci$plot_E)

tree <- tree |>
  mutate(plot_E = NA) |>
  left_join(anci$plot_E, by = join_by(plot_no), suffix = c("_rm", "")) |>
  select(-ends_with("_rm"))

summary(tree$plot_E)
```

```
    Min.  1st Qu.   Median     Mean  3rd Qu.     Max. 
-0.08457  0.19793  0.24559  0.23280  0.28740  0.41069 
```

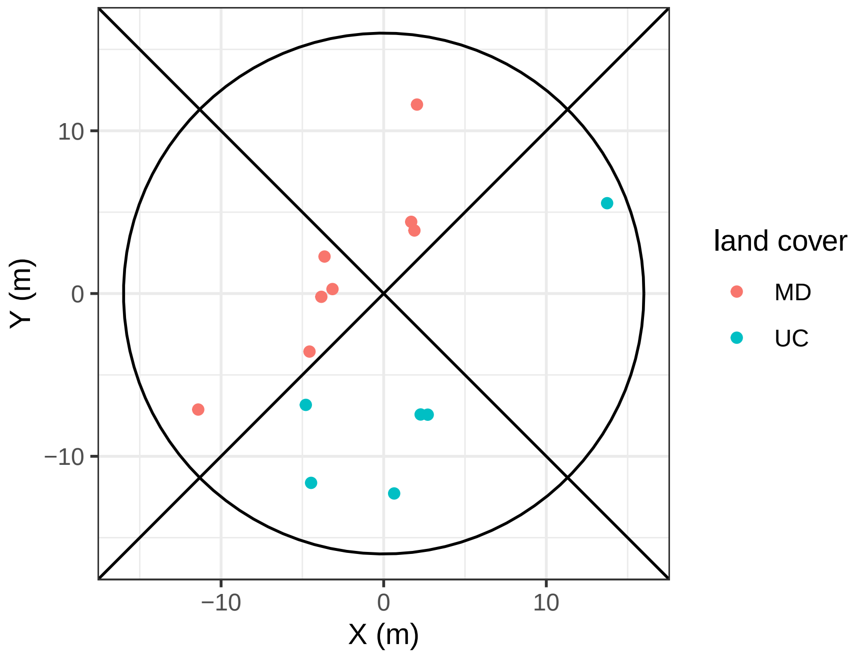
Now that we have land cover information at tree level we can make a figure of tree location with color based on land cover code in Exercise 6.4.

Exercise 6.4 (~ (optional) Figure of tree location with land cover).

- Remake the figure of Exercise 6.1 with these differences:
- Change the color of trees based on lc_code
- Remove geom_label_repel() to stop showing the azimuths
- Remove hline and vline
- Add geom_abline() with intercept = 0 and slope = 1
- Add geom_abline() with intercept = 0 and slope = -1
- Check visually that different land cover are in different quadrants

Type here answers to Exercise 6.4:

```
## !!! Solution
tree |>
  filter(plot_no == 123, subplot_no == "B") |>
  ggplot(aes(x = tree_x, y = tree_y)) +
  geom_point(aes(color = lc_code)) +
  geom_path(data = circ16, aes(x = x, y = y)) +
  geom_abline(intercept = 0, slope = 1) +
  geom_abline(intercept = 0, slope = -1) +
  coord_fixed() +
  labs(x = "X (m)", y = "Y (m)", color = "land cover")
```

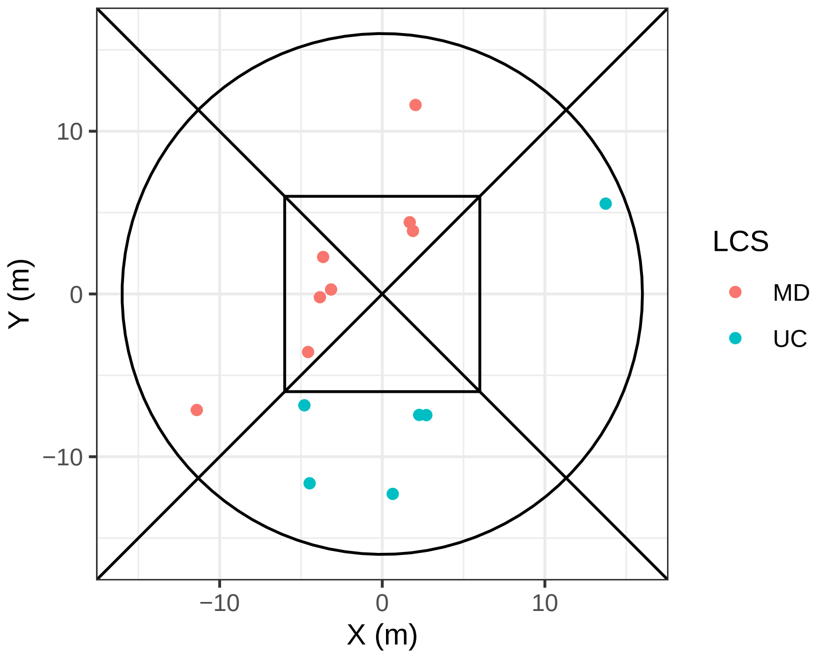


Subplot '123B' is split between two land covers, Mixed deciduous (MD) in the land cover sections (LCS) 1, 2 and 5, and agriculture in LCS 3 and 4.

6.2.3 (optional) Final graph

We can add the square center LCS with a specific table

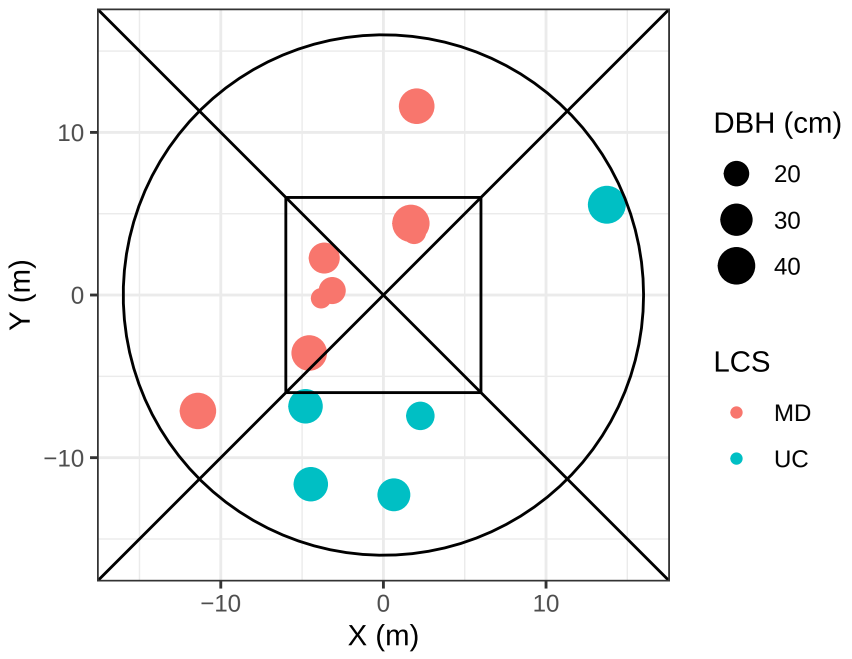
```r
center_sq <- tibble(x = c(-6, 6, 6, -6, -6), y = c(6, 6, -6, -6, 6))

tree |>
  filter(plot_no == 123, subplot_no == "B") |>
  ggplot(aes(x = tree_x, y = tree_y)) +
  geom_point(aes(color = lc_code)) +
  geom_path(data = circ16, aes(x = x, y = y)) +
  geom_abline(intercept = 0, slope = 1) +
  geom_abline(intercept = 0, slope = -1) +
  geom_path(data= center_sq, aes(x = x, y = y)) +
  coord_fixed() +
  labs(x = "X (m)", y = "Y (m)", color = "LCS")
```



6.2.4 Bonus: Tree DBH true to size on figures with the `ggforce` package.

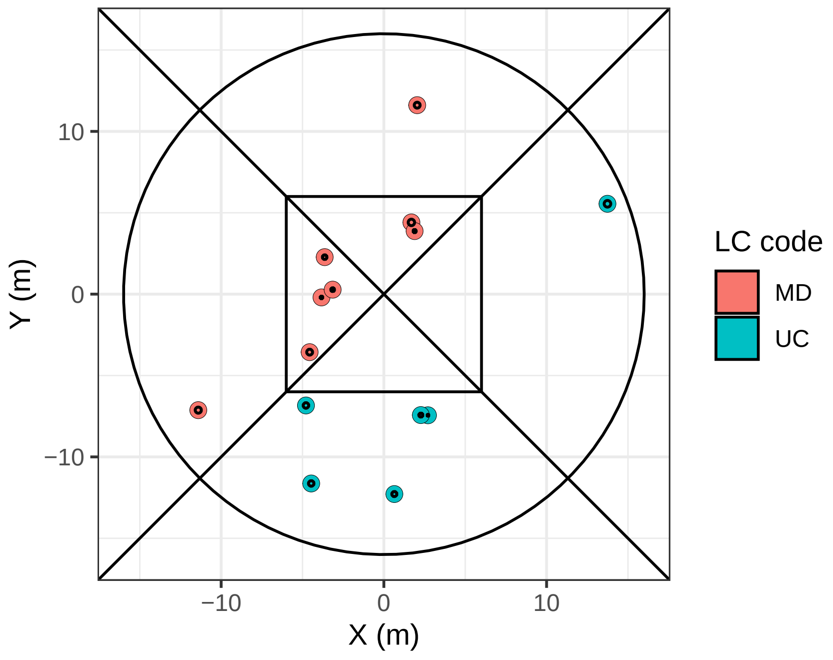
Graphs with `ggplot()` don't natively intend to represent points true to size with `geom_point()`, as size aesthetic is optimized for visibility. Here we represent the same figure as above but with point size change based on tree DBH

```r
tree |>
  filter(plot_no == 123, subplot_no == "B") |>
  ggplot(aes(x = tree_x, y = tree_y)) +
  geom_point(aes(color = lc_code, size = tree_dbh)) +
  geom_path(data = circ16, aes(x = x, y = y)) +
  geom_abline(intercept = 0, slope = 1) +
  geom_abline(intercept = 0, slope = -1) +
  geom_path(data= center_sq, aes(x = x, y = y)) +
  coord_fixed() +
  labs(x = "X (m)", y = "Y (m)", color = "LCS", size = "DBH (cm)")
```



Now if we want to see trees' DBH true to size, with the plot area, we can use `geom_circle()` from the package `ggforce`.

```
tree |>
  filter(plot_no == 123, subplot_no == "B") |>
  ggplot(aes(x = tree_x, y = tree_y)) +
  geom_point(aes(fill = lc_code), shape = 21, size = 3, stroke = 0) +
  geom_circle(aes(x0 = tree_x, y0 = tree_y, r = tree_dbh/200, fill = lc_code)) +
  geom_path(data = circ16, aes(x = x, y = y)) +
  geom_abline(intercept = 0, slope = 1) +
  geom_abline(intercept = 0, slope = -1) +
  geom_path(data= center_sq, aes(x = x, y = y)) +
  coord_fixed() +
  labs(x = "X (m)", y = "Y (m)", fill = "LC code", size = "DBH (cm)")
```



Here we kept the points as the trees are small and wouldn't be visible otherwise.

7 Session 07: tree weights and basal area

Goals

1. Calculate tree weights for ratio estimators and for subplot per ha values
2. Calculate tree basal area

7.1 Tree weights for nested circles adjustment

7.1.1 Tree weight for ratio estimator

In the context of **ratio estimator,** tree weight converts small size trees' variables to their equivalent if small trees were measured in the same area as the bigger sized trees.

For Lao PDR, the ratio between small and big trees is 4:

```
tree <- tree |> mutate(tree_weight = if_else(tree_dbh < 30, 4, 1))
```

Let's calculate where this value 4 comes from in Exercise 7.1.

Exercise 7.1 (~ Where does weight 4 come from?)**.**

- Calculate 'Asmall' as the area of a circle of 8m radius
- Calculate 'Abig' as the area of a circle of 16m radius
- Calculate Abig / Asmall

Type here answers to Exercise 7.1:

```
## Solution
Asmall <- pi * 8^2
Abig   <- pi * 16^2
Abig / Asmall
```

```
[1] 4
```

7.1.2 Tree weight for subplot per ha values

To get per ha at the subplot level, we first assign to each tree its subplot nested circle area, then the weight is the inverse of this area

```
tree <- tree |>
  mutate(
    tree_spha = if_else(tree_dbh < 30, pi * 16^2 / 10000, pi * 16^2 / 10000),
    tree_weight_spha = 1 / tree_spha
  )
```

There might be an error in the above code snippet, let's solve it in Exercise 7.2.

Exercise 7.2 (~ Find the error).

- Copy/paste the previous code snippet
- Find and correct the error in `tree_spha`

Type here answers to Exercise 7.2:

```
## !!! Solution
tree <- tree |> mutate(
  tree_spha = if_else(tree_dbh < 30, pi * 8^2 / 10000, pi * 16^2 / 10000),
  tree_weight_spha = 1 / tree_spha
  )
```

7.1.3 Calculate tree basal area

The basal area of a tree is its stem area footprint based on the measurement of it diameter at breast height.

```
tree <- tree |> mutate(tree_ba = pi * (tree_dbh/200)^2)
```

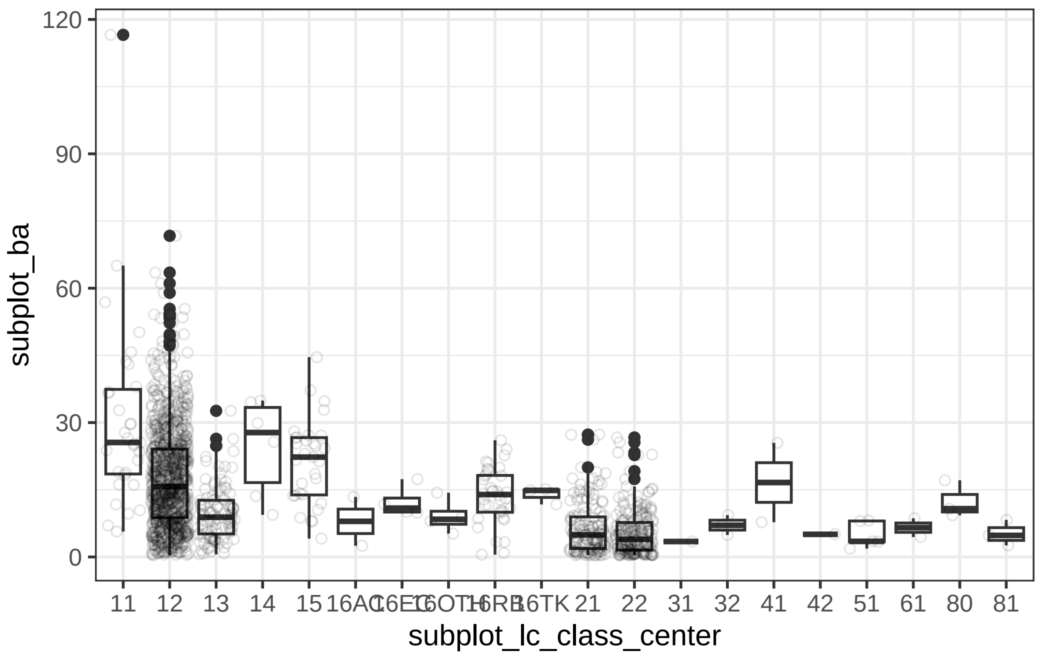
Let's check subplot level basal area per land cover class in Exercise 7.3.

Exercise 7.3 (~ (optional) Basal area per subplot).

- From tree, use `group_by()` and `summarise()` to create `subplot_ba`, the sum of `tree_ba * tree_weight_spha` per plot and subplot number,
- Join `lc_class_center` from the `subplot` table using suffix control,
- Make a boxplot of `subplot_ba` against `lc_class_center`.

Type here answers to Exercise 7.3:

```
## !!! Solution
tmp_sp <- subplot |> select(plot_no, subplot_no, subplot_lc_class_center)

tree |>
  group_by(plot_no, subplot_no) |>
  summarise(subplot_ba = sum(tree_ba * tree_weight_spha), .groups = "drop") |>
  left_join(tmp_sp, by = join_by(plot_no, subplot_no)) |>
  ggplot(aes(x = subplot_lc_class_center, y = subplot_ba)) +
  geom_boxplot() +
  geom_jitter(alpha = 0.1, shape = 21)
```



BONUS: The function `geom_jitter()` is a nice addition to boxplot figures to see all the subplot values on top of their distribution.

8 Session 08: Tree aboveground biomass

GOALS:

1. add AGB at tree level for each forest type
2. Add AGB from chave 05 and 14 without height
3. Make a figure for natural forest and one for plantations, showing forest type AGB, and chave models

8.1 List of models

- `"EG" ~ 0.3112 * tree_dbh^2.2331,`
- `"MD" ~ 0.523081 * tree_dbh^2,`
- `"DD" ~ 0.2137 * tree_dbh^2.2575,`
- `"CF" ~ 0.1277 * tree_dbh^2.3944,`
- `"MCB" ~ 0.1277 * tree_dbh^2.3944,`
- `"P_AC" ~ 0.1173 * tree_dbh^2.454,`
- `"P_EC" ~ 0.199 * tree_dbh^2.185,`
- `"P_RB" ~ 0.0082 * (pi*tree_dbh)^2.5623, ## Rubber model uses circumference`
- `"P_TK" ~ 0.077 * tree_dbh^2.546,`
- `"P_OTH" ~ 0.3112 * tree_dbh^2.2331,`
- `"RV" ~ 0.6 * exp(-1.499 + 2.148 * log(tree_dbh) + 0.207 * (log(tree_dbh))^2 - 0.0281*(log(tree_dbh))^3),`
- `"B" ~ 0.6 * exp(-1.499 + 2.148 * log(tree_dbh) + 0.207 * (log(tree_dbh))^2 - 0.0281*(log(tree_dbh))^3),`
- `Chave14 = round(exp(-1.803 - 0.976plot_E + 0.976log(0.6) + 2.673log(tree_dbh) -0.0299(log(tree_dbh))^2)`
- `Chave05 = round(0.6 * exp(-1.499 + 2.148log(tree_dbh) + 0.207(log(tree_dbh))^2 - 0.0281*(log(tree_dbh))^3)`

8.2 AGB per forest type

We can assign different AGB models to trees for different forest types with `mutate()` and `case_when()`:

```r
#table(tree$lc_code)

tree <- tree |>
  mutate(
    tree_agb_final = case_when(
      lc_code == "EG" ~ 0.3112 * tree_dbh^2.2331,
      ## ADD OTHER equations here
      TRUE ~ 0
    )
  )
```

Let's fill in the other cases in the Exercise 8.1.

Exercise 8.1 (~ Complete tree_agb_final)**.**

- For all forest land covers add their equations as shown at the beginning of the document.
- For land cover with no equation, assign 0 with `TRUE ~ 0`.
- Make a figure with 'tree_agb_final' against 'tree_dbh' as points (optional)
- Make a figure with 'tree_agb_final' against 'tree_dbh' as line with color based on 'lc_code' (optional)

Type here answers to Exercise 8.1:

```
## !!! Solution
tree <- tree |>
  mutate(
    tree_agb_final = case_when(
      lc_code == "EG"    ~ 0.3112 * tree_dbh^2.2331,
      lc_code == "MD"    ~ 0.523081 * tree_dbh^2,
      lc_code == "DD"    ~ 0.2137 * tree_dbh^2.2575,
      lc_code == "CF"    ~ 0.1277 * tree_dbh^2.3944,
      lc_code == "MCB"   ~ 0.1277 * tree_dbh^2.3944,
      lc_code == "P_AC"  ~ 0.1173 * tree_dbh^2.454,
      lc_code == "P_EC"  ~ 0.199 * tree_dbh^2.185,
      lc_code == "P_RB"  ~ 0.0082 * (pi*tree_dbh)^2.5623,
      ## > Rubber model uses circumference
      lc_code == "P_TK"  ~ 0.077 * tree_dbh^2.546,
      lc_code == "P_OTH" ~ 0.3112 * tree_dbh^2.2331,
      lc_code == "RV"    ~ 0.6 * exp(-1.499 + 2.148 * log(tree_dbh) + 0.207 * (log(tree_dbh))^
      lc_code == "B"     ~ 0.6 * exp(-1.499 + 2.148 * log(tree_dbh) + 0.207 * (log(tree_dbh))^
      TRUE ~ 0
    )
  )

# tree |>
#   ggplot(aes(x = tree_dbh, y = tree_agb_final)) +
#   geom_point()

# tree |>
#   ggplot(aes(x = tree_dbh, y = tree_agb_final)) +
#   geom_line(aes(color = lc_code))
```

8.3 AGB from Chave 2005 and 2014

Let's add 3 models to all trees, regardless of their forest type to compare with the forest type based models, in Exercise 8.2.

Exercise 8.2 (~ Add AGB allometric equations for all trees)**.**

- Add 'tree_agb_chave05' with Chave et al. 2005 equation
- Add 'tree_agb_chave14' with Chave et al. 2014 equation

- Add ‘tree_agb_EG’ with the evergreen forest model

Type here answers to Exercise 8.2:

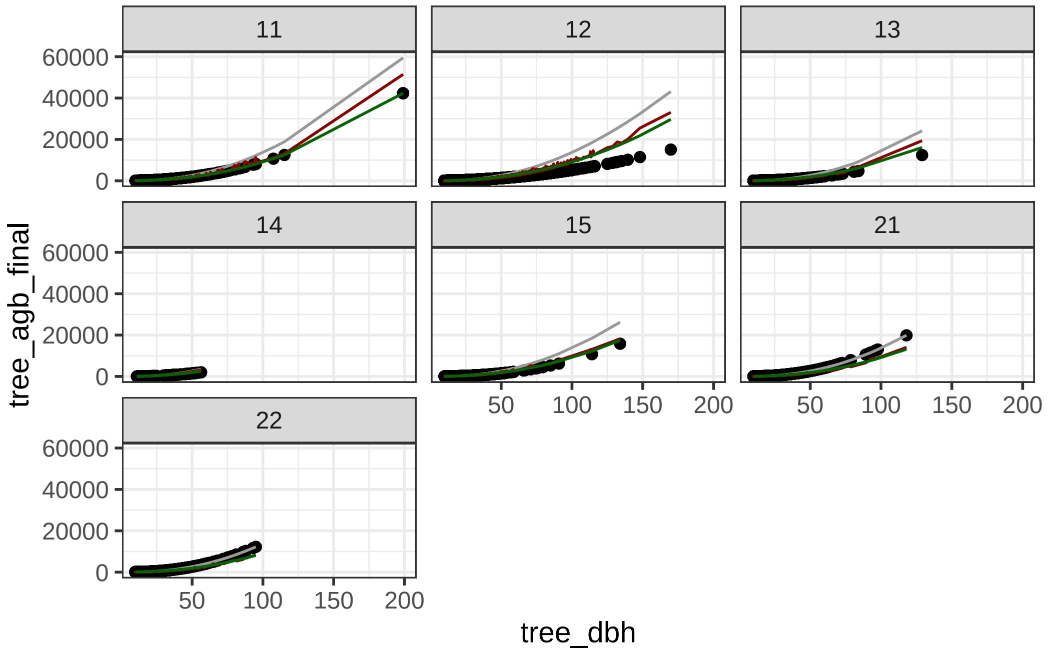
```r
## !!! Solution
tree <- tree |>
  mutate(
    tree_agb_chave05 =
      0.6 * exp(
      -1.499 + 2.148*log(tree_dbh) +
        0.207*(log(tree_dbh))^2 - 0.0281*(log(tree_dbh))^3
      ),
    tree_agb_chave14 =
      exp(
        -1.803 - 0.976*plot_E + 0.976*log(0.6) +
          2.673*log(tree_dbh) - 0.0299*(log(tree_dbh))^2
        ),
    tree_agb_EG = 0.3112 * tree_dbh^2.2331
    )
```

8.4 Compare AGB models

8.4.1 Natural forest

Let’s compare our forest type specific tree aboveground with the Chave’s models and the model for Evergreen forests. Since there are many forest type let’s look at natural forests first with `filter()`.

```r
tree |>
  filter(tree_agb_final != 0, lc_no <= 29) |>
  ggplot(aes(x = tree_dbh, y = tree_agb_final)) +
  geom_point() +
  geom_line(aes(y = tree_agb_chave05), color = "grey60") +
  geom_line(aes(y = tree_agb_chave14), color = "darkred") +
  geom_line(aes(y = tree_agb_EG), color = "darkgreen") +
  facet_wrap(~lc_no)
```



We can observe in this figure that the forest type specific models are quite conservative, since our AGB final model predicts smaller AGB for the same DBH compared to both Chave's models.

8.4.2 (optional) Planted forest

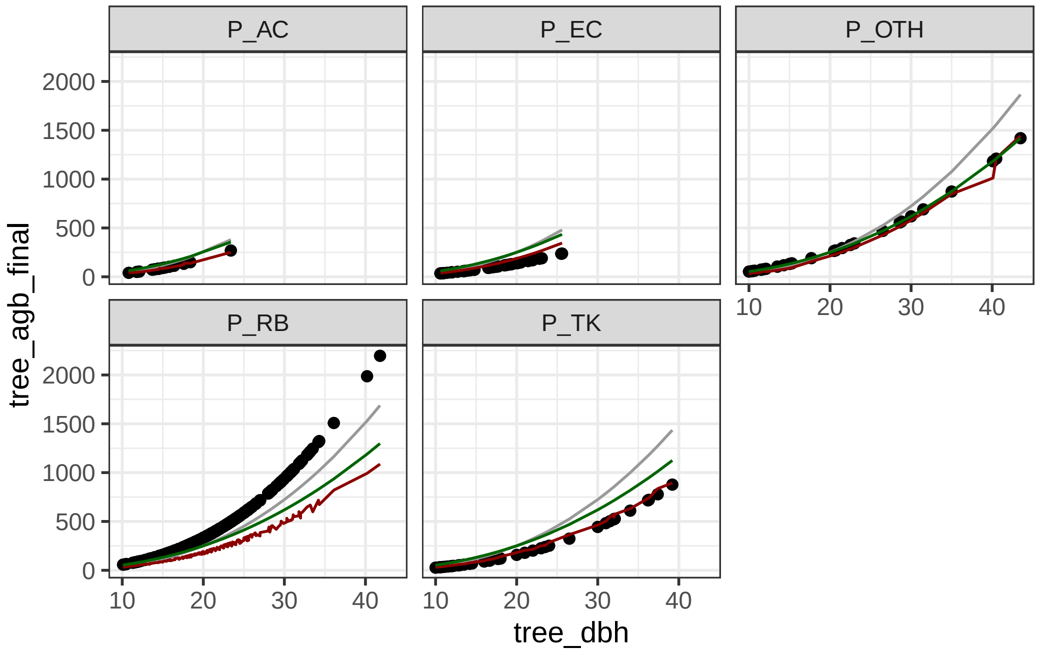
Let's produce the same figure for planted forests in Exercise 8.3.

Exercise 8.3 (~ Compare models for plantations)**.**

- Make a similar graph than 3.1 but for planted forest only

Type here answers to the Exercise 8.3:

```r
## Your code here

## !!! SOL
tree |>
  filter(tree_agb_final != 0, lc_no >= 160) |>
  ggplot(aes(x = tree_dbh, y = tree_agb_final)) +
  geom_point() +
  geom_line(aes(y = tree_agb_chave05), color = "grey60") +
  geom_line(aes(y = tree_agb_chave14), color = "darkred") +
  geom_line(aes(y = tree_agb_EG), color = "darkgreen") +
  facet_wrap(~lc_code)
```



```
## !!!
```

Plantation models selected for AGB final are also conservative in general, except for the rubber model.

9 Session 09: Aggregate tree to forest type AGB

GOALS:

1. Calculate the mean aboveground biomass for all forest types using **plot level** land cover and **systematic sampling**.
2. Calculate the mean aboveground biomass for all forest types using **ratio estimators** and **2-stages sampling for post-stratification**.
3. Compare estimates.

9.1 AGB with plot level land cover and systematic sampling

Step-by-step

1. Sum tree AGB to subplots.
2. Make plot majority LC class.
3. Create plot level AGB based on mean subplot AGB.
4. Create Forest type AGB as the mean of plot AGB per forest type.
5. Calculate sampling error.

9.1.1 Sum tree AGB to subplot

We will use `group_by()` and `summarise()` to bring tree characteristics to the subplot level.

```
subplot_agb <- tree |>
  group_by(plot_no, subplot_no) |>
  summarise(
    sp_agb = sum(tree_agb_final * tree_weight_spha / 1000),
    sp_ba  = sum(tree_ba * tree_weight_spha),
    sp_dens = sum(tree_weight_spha),
    tree_meas = n(),
    .groups = "drop"
  )
```

9.1.2 Load plot level main land cover

Plot level land cover was generated, from the subplots center LC, with the following rules:

1. For plots with one unique LC, they were directly assigned to plot level.
2. For plots with multiple land cover, we used the majority LC for the whole plot.
3. If some plots have equal importance of two land covers, we used to minimum land cover class.

```
plot_lc <- read_csv("data/training_plot_lc.csv", show_col_types = FALSE)
```

9.1.3 Aggregate subplot to plot

We take the mean of subplot variables (except tree measured count).

```
plot_agb <- subplot_agb |>
  group_by(plot_no) |>
  summarise(
    plot_agb = mean(sp_agb),
    plot_ba = mean(sp_ba),
    plot_dens = mean(sp_dens),
    plot_tree_meas = sum(tree_meas)
  ) |>
  left_join(plot_lc, by = join_by(plot_no))
```

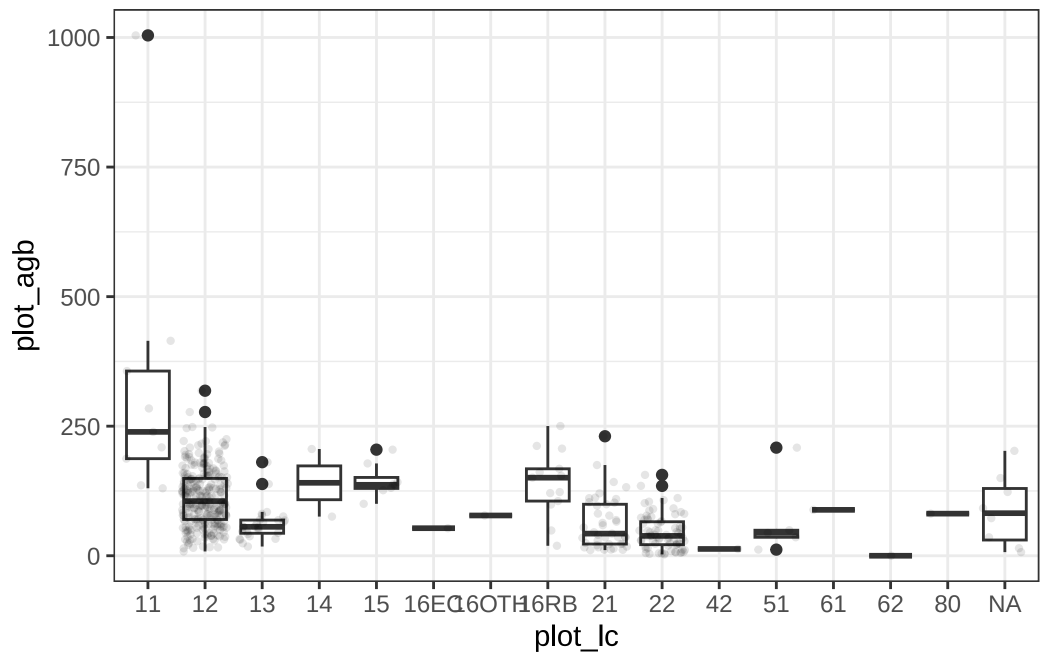
Let's look at plot AGB per land cover in Exercise 9.1.

Exercise 9.1 (~ Boxplot of plot AGB)**.**

- Make a boxplot of plot AGB against land cover classes

Type here the answers to Exercise 9.1:

```
## !!! Solution
plot_agb |>
  ggplot(aes(x = plot_lc, y = plot_agb)) +
  geom_boxplot() +
  geom_jitter(alpha = 0.1, size = 0.8)
```



9.1.4 Aggregate to forest type AGB

Now we can take averages of plot AGB for each forest type. For the uncertainty, we also need **standard deviation** of AGB and the **number of plots**.

```
ftype_agb <- plot_agb |>
  group_by(plot_lc) |>
  summarise(
    plot_count = n(),
    total_tree_meas = sum(plot_tree_meas),
    ftype_dens = mean(plot_dens),
    ftype_ba = mean(plot_ba),
    ftype_agb = mean(plot_agb),
    ftype_agb_sd = sd(plot_agb)
  )
```

We can calculate the uncertainty for systematic design with this formula:

$$U\% = t_{n-1}^{1-\alpha/2} \times \frac{\sigma}{\sqrt{n}} \times \frac{1}{\mu} \times 100$$

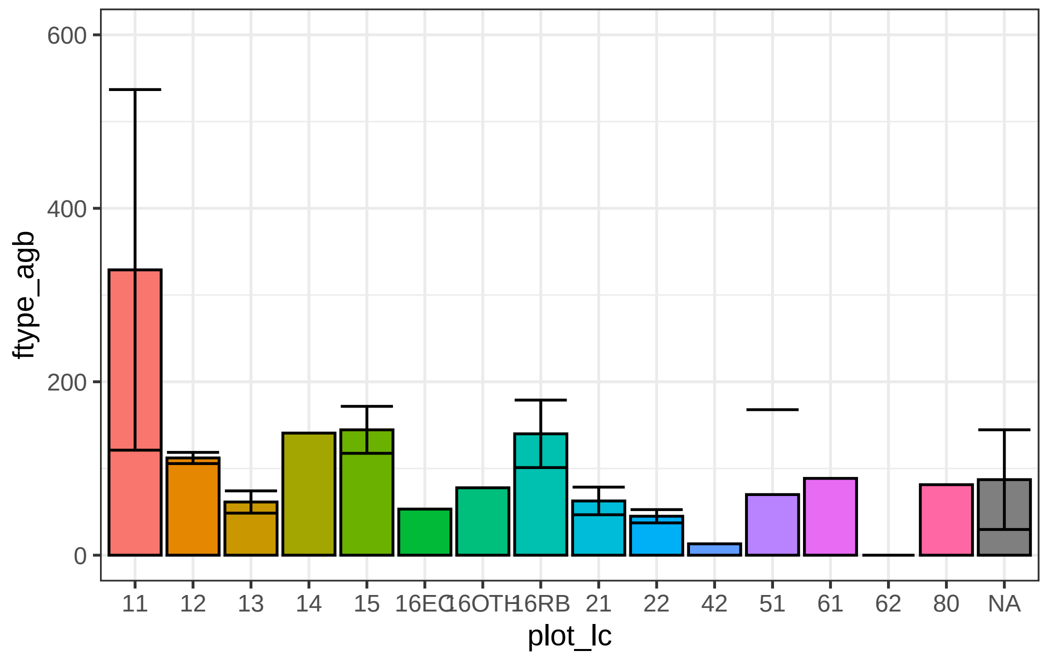
The student's law function in R is `qt()`, and uncertainty can be calculated from the AGB mean and standard deviation as follows:

```
ftype_agb <- ftype_agb |>
  mutate(
    ftype_agb_se = ftype_agb_sd / sqrt(plot_count),
    ftype_agb_me = qt(0.975, plot_count - 1) * ftype_agb_se,
    ftype_agb_U  = ftype_agb_me / ftype_agb * 100,
    ftype_agb_cilower = ftype_agb - ftype_agb_me,
    ftype_agb_ciupper = ftype_agb + ftype_agb_me
  )
```

```
Warning: There was 1 warning in `mutate()`.
i In argument: `ftype_agb_me = qt(0.975, plot_count - 1) * ftype_agb_se`.
Caused by warning in `qt()`:
! NaNs produced
```

We can then make a plot of forest type AGB with sampling uncertainty. We added a limit on the y-axis to avoid uncertainty of Coniferus Forest (CF, code 14) condensing the whole figure with `ylim(0, 600)`.

```
ftype_agb |>
  ggplot(aes(x = plot_lc, y = ftype_agb)) +
  geom_col(aes(fill = plot_lc), col = "black") +
  geom_errorbar(aes(ymin = ftype_agb_cilower, ymax = ftype_agb_ciupper)) +
  theme(legend.position = "none") +
  ylim(0, 600)
```



9.2 AGB with ratio estimator and 2-stages sampling for post-stratification

Step-by-step:

1. Prepare subplot x LCS level table
2. Use the aggregate function 'nfi_aggregate3()' to get all aggregation levels
3. Compare results with equal plot statistiscs (table 'ftype')

9.2.1 Prepare the subplot x LCS table for aggregation

9.2.1.1 Initial data preparation

For the second approach, all subplots must be accounted for, visited or not, accessible or not. This table has been prepared and can be loaded into the ancillary data list `anci`.

```
anci$ph2 <- read_csv("data/training_anci_phase2.csv", show_col_types = F)

ph2_subplot <- anci$ph2

# format(nrow(anci$ph2) / 20, big.mark = ",")
```

We can observe that `anci$ph2` contains 1,067 *(1,067)* plots, as initially planned in the 12 x 18 km grid.

To generate this table, instead of starting from tree aggregates or from the subplot table, we start from the vector of planned Phase 2 plots and we recreate all combinations of subplot x LCS with `expand_grid()` .

```r
vec_ph2 <- anci$ph1 |>
  filter(!is.na(plot_id)) |>
  pull(plot_id) |>
  sort()

test_ph2 <- expand_grid(
  plot_id = vec_ph2,
  subplot_no = c("A", "B", "C", "D"),
  lcs_no = 1:5
) |>
  mutate(subplot_id = paste0(subplot_no, lcs_no))
```

We can check that this table is identical to `anci$ph2` with `all.equal()`:

```r
tmp_ph2 <- anci$ph2 |>
  select(plot_id, subplot_no, lcs_no, subplot_id)

all.equal(tmp_ph2, test_ph2)
```

```
[1] TRUE
```

We then need to add **accessibility** from subplot and subpopulation and **strata** from phase 1 data with `left_join()` in Exercise 9.2.

Exercise 9.2 (~ (optional) Join core info to phase 2 table).

- Prepare 'tmp_sp' a table containing `plot_no`, `subplot_no` and `subplot_access` from the subplot table and rename `plot_no` into `plot_id`.
- Prepare 'tmp_ph1' a table containing `plot_id`, `subpop`, `stratum` , `ph1_prov_no` and `ph1_prov_name` from the table `anci$ph1` and rename `ph1_prov_no` into `prov_no` and `ph1_prov_name` into `prov_name`
- Join these 2 tables to `ph2_subplot` with suffix control method.

Type here answers to Exercise 9.2:

```r
## !!! Solution
tmp_sp <- subplot |>
  select(plot_id = plot_no, subplot_no, subplot_access)

tmp_ph1 <- anci$ph1 |>
  select(plot_id, subpop, stratum, prov_no = ph1_prov_no, prov_name = ph1_prov_name)


ph2_subplot <- ph2_subplot |>
  mutate(
    subplot_access = NA,
    subpop = NA,
    stratum = NA,
    prov_no = NA,
    prov_name = NA
  ) |>
  left_join(tmp_ph1, by = join_by(plot_id), suffix = c("_rm", "")) |>
  left_join(tmp_sp, by = join_by(plot_id, subplot_no), suffix = c("_rm", "")) |>
  select(-ends_with("_rm"))
```

In the final preparation phase 2 subplot table, correction for shifted plots was implemented beforehand.

9.2.1.2 (NOTE) Investigating phase 2 plots in stratum 4: non-forest

The Phase 2 focused on forest and RV land cover, but some plots from Phase 1 and planned for phase 2 were non-forest in the Phase 1 interpretation and still visited.

```
tmp_ph2_nonforest <- anci$ph1 |>
  filter(!is.na(plot_id), stratum == 4)

vec_ph2nf <- tmp_ph2_nonforest |> pull(ph1_plot_no)

tmp_ph2nf_visited <- subplot |>
  filter(plot_no %in% vec_ph2nf, subplot_no == "A")

## Save the tables in a new folder
if (!"results" %in% list.files()) dir.create("results")

write_csv(tmp_ph2_nonforest, "results/ph2_nonforest_planned.csv")
write_csv(tmp_ph2nf_visited, "results/ph2_nonforest_visited.csv")
```

9.2.1.3 Re-coding accessibility

Accessibility plays an important role in the final estimates. Since not all the initial 1,067 plots were visited, there is missing information on **many non-forest plots** and **some forest plots**. To simplify the calculations, it was considered that:

- **all non-visited forest plots would be considered as non-accessible,**
- **all non-visited non-forest plots would be considered as accessible with biomass 0.**

This is important to keep in mind in case biomass of trees outside forest becomes relevant for future NFI cycles.

Code for re-coding accessibility:

```
ph2_subplot <- ph2_subplot |>
  mutate(
    access = case_when(
      plot_id <= 636 & subplot_access == "accessible" ~ TRUE,
      plot_id <= 636 & subplot_access != "accessible" ~ FALSE,
      plot_id <= 636 & is.na(subplot_access) ~ FALSE,
      stratum %in% 1:3 ~ FALSE,
      stratum == 4 ~ TRUE,
      TRUE ~ NA
    )
  )
```

9.2.1.4 Subplot largest area

For aggregation with ratio estimator we need to keep track of the areas where trees are measured. Since we have a tree weight for small trees, we are only interested in the larger area:

- 12 m side square of a quarter
- difference between the 16m radius circle and the 12m side square.

```
ph2_subplot <- ph2_subplot |>
  mutate(
    sp_area = if_else(lcs_no == 1, 12^2, (pi*16^2 - 12^2)/4) / 10000
  )
```

9.2.1.5 Aggregate tree to subplot x LCS

We can now aggregate trees to the subplot x LCS level and join with the phase 2 subplot table.

```
ph2_sp_tree <- tree |>
  select(plot_id = plot_no, subplot_no, lcs_no, tree_weight, tree_ba, tree_agb_final) |>
  mutate(
    subplot_id = paste0(subplot_no, lcs_no)
  ) |>
  group_by(plot_id, subplot_id) |>
  summarise(
    dens = sum(tree_weight),
    ba   = sum(tree_ba * tree_weight),
    agb  = sum(tree_agb_final * tree_weight) / 1000, ## Convert kg to tons
    .groups= "drop"
  )
```

```
ph2_subplot <- ph2_subplot |>
  mutate(
    dens = NA,
    ba = NA,
    agb = NA
  ) |>
  left_join(ph2_sp_tree, by = join_by(plot_id, subplot_id), suffix = c("_rm", "")) |>
  select(-ends_with("_rm")) |>
  mutate(
    dens = if_else(!is.na(dens), n_tree, 0),
    ba   = if_else(!is.na(ba), ba, 0),
    agb  = if_else(!is.na(agb), agb, 0)
  )
```

9.2.2 Run the aggregation function

The aggregation function 3, for 2-stages sampling with ratio estimator, takes phase 1 and phase 2 data as input as well as the attributes y and x for the ratio. y can change based on what attribute we want to aggregate, while x is always the minimum measured area.

Let's see an example with tree density and no subpopulation.

```
ph2_data <- ph2_subplot |> mutate(subpop = 1)

res3_dens <- nfi_aggregate3(
  .ph1_df = anci$ph1,
  .ph2_sp = ph2_data,
  .class_d = lc_no,
  .attr_y = dens,
  .attr_x = sp_area,
  .aoi_area = 23680000
    )

res3_dens_tot <- res3_dens$totals_short
```

The function output is a list with different levels of data aggregation.

Practice now the function `nfi-aggregate3()` to get basal area and aboveground biomass in Exercise 9.3.

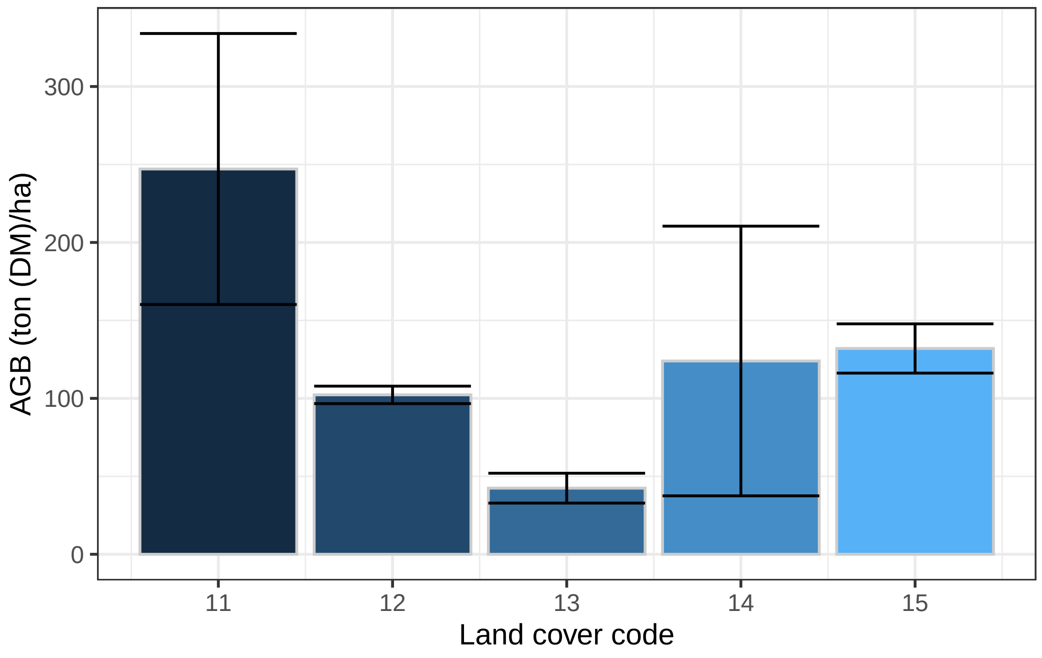
Exercise 9.3 (~ Aggregate basal area and AGB).

- (optional) Create `res3_ba` as the output of the aggregation function with basal area as `.attr_y` .
- Create `res3_agb` as the output of the aggregation function with aboveground biomass as `.attr_y` .
- Save the totals_short output from res3_agb to a new object `res3_agb_tot`.
- Make a barplot with AGB per ha against land cover with their error bars, for natural forest only.

Type here answers to Exercise 9.3:

```
## !!! Solution
ph2_data <- ph2_subplot |> mutate(subpop = 1)

res3_agb <- nfi_aggregate3(
  .ph1_df = anci$ph1,
  .ph2_sp = ph2_data,
  .class_d = lc_no,
  .attr_y = agb,
  .attr_x = sp_area,
  .aoi_area = 23680000
    )

res3_agb_tot <- res3_agb$totals_short
```

```
res3_agb_tot |>
  filter(lc_no < 20) |>
  ggplot(aes(x = lc_no, y = Rd)) +
  geom_col(aes(fill = lc_no), col = "grey80") +
  theme(legend.position = "none") +
  geom_errorbar(aes(ymin = Rd - Rd*Rd_mep/100, ymax = Rd + Rd*Rd_mep/100)) +
  labs(
    x = "Land cover code",
    y = "AGB (ton (DM)/ha)"
  )
```



We can use an additional package: `kableExtra`, to customize tables with adding extra headers and even colors. However this package focuses on HTML and PDF outputs, for DOCX, we stick to simple `kable()` function. Let's show the totals for AGB:

```
tab_agb <- res3_agb$totals_d |>
  select(lc_no, Xd, Xd_mep, Xtot, Yd, Yd_mep, Rd, Rd_mep, Ytot) |>
  filter(lc_no <= 22 | lc_no >= 160)


if (knitr::is_html_output() | knitr::is_latex_output()){

  kbl(
  tab_agb,
  col.names = c(
    "Land cover", "prop.", "U (perc)", "tot.", "prop.", "U (perc)",
    "ton (DM)/ha", "U (perc)", "tot."
    ),
  format.args = list(big.mark = ",", digits = 2),
  booktabs = F
  ) |>
  kable_styling(
    bootstrap_options = c("bordered", "condensed", "responsive"),
```

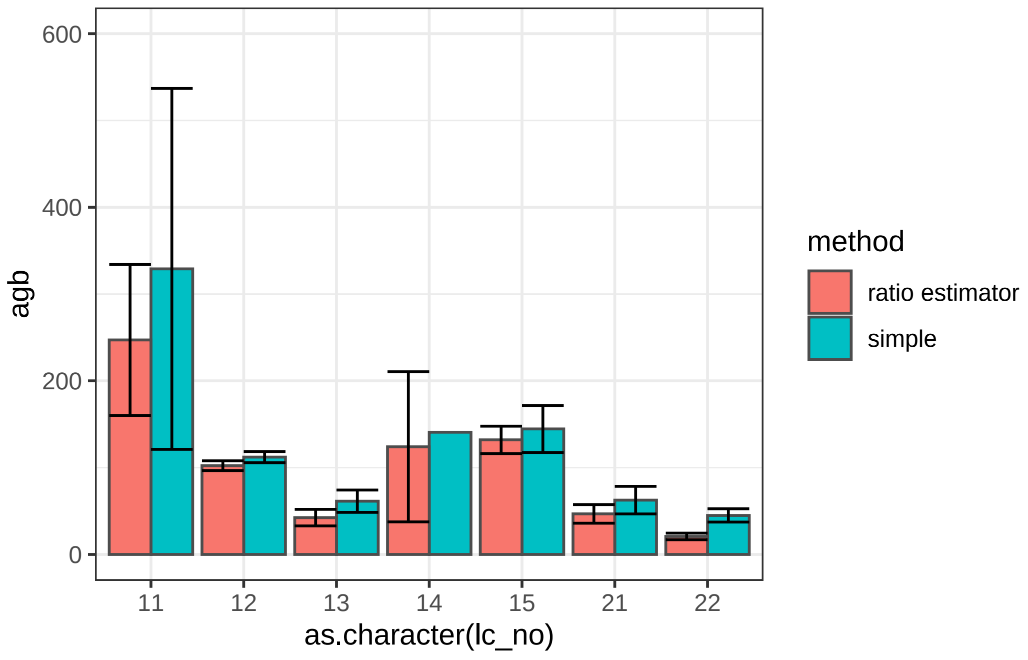
	Area			AGB				
Land cover	prop.	U (perc)	tot.	prop.	U (perc)	ton (DM)/ha	U (perc)	tot.
11	0.00525	47.4	124,260	1.297	60.0	247	35.2	3.1e+07
12	0.15406	6.2	3,648,029	15.750	8.1	102	5.5	3.7e+08
13	0.01993	23.9	471,951	0.845	35.3	42	22.6	2.0e+07
14	0.00107	101.4	25,424	0.133	141.4	124	69.8	3.2e+06
15	0.00456	45.9	107,889	0.601	49.5	132	12.0	1.4e+07
21	0.02898	18.7	686,166	1.354	29.2	47	23.0	3.2e+07
22	0.07202	10.7	1,705,431	1.495	21.1	21	18.3	3.5e+07
161	0.00041	116.5	9,725	0.017	143.5	42	73.6	4.1e+05
162	0.00159	84.0	37,725	0.029	164.7	18	125.0	6.9e+05
164	0.00736	34.5	174,181	1.025	40.6	139	22.5	2.4e+07
165	0.00035	105.5	8,177	0.028	101.2	81	11.7	6.6e+05
169	0.00132	84.8	31,237	0.041	132.9	31	78.8	9.7e+05

```
    full_width = FALSE
  ) |>
  add_header_above(
    c(" " = 1, "Area" = 3, "AGB" = 5), border_left = T, border_right = T
    ) |>
  column_spec(1, border_left = T) |>
  column_spec(9, border_right = T)

} else {

  kable(
  tab_agb,
  col.names = c(
    "Land cover", "Area prop.", "Area U (perc)", "Area tot. (ha)", "AGB prop.",
    "AGB prop. U (perc)", "AGB (ton DM/ha)", "AGB U (perc)", "AGB tot. (tons)"
    ),
  format.args = list(big.mark = ",", digits = 2)
  )

}
```

9.3 Compare AGB estimates from the 2 aggregation methods

Check AGB from 2 methods

```
tmp_agb1 <- ftype_agb |>
  filter(plot_lc %in% c("11", "12", "13", "14", "15", "21", "22")) |>
  mutate(
    plot_lc = as.numeric(plot_lc),
    method = "simple"
    ) |>
  select(method, lc_no = plot_lc, agb = ftype_agb, agb_U = ftype_agb_U)

tmp_agb2 <- res3_agb$totals_short |>
  filter(lc_no <= 22) |>
  mutate(method = "ratio estimator") |>
  select(method, lc_no, agb = Rd, agb_U = Rd_mep)

agb_compa <- tmp_agb1 |> bind_rows(tmp_agb2)

agb_compa |>
  ggplot(aes(x = as.character(lc_no), y = agb)) +
  geom_col(aes(fill = method), col = "grey30", position = position_dodge()) +
  geom_errorbar(aes(group = method, ymin = agb - agb*agb_U/100, ymax = agb + agb*agb_U/100), p
  ylim(0, 600)
```



The ratio estimator method had big impact on EG and RV. Also big influence on uncertainty of CF.

10 Session 10: Building a R community of practice in Lao PDR

TBD

Conclusion

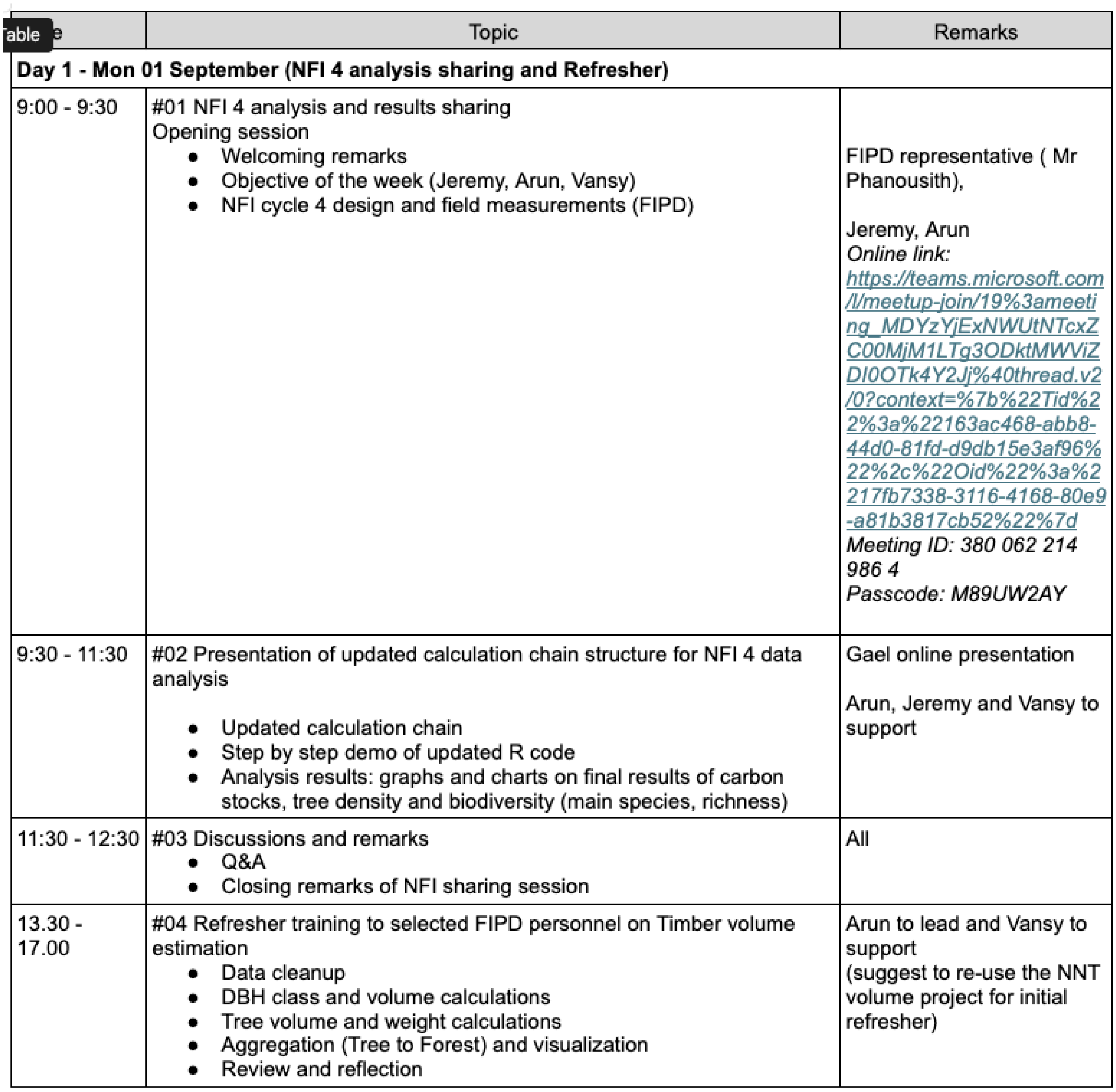
The training aimed to continue FIPD capacity building to handle the NFI data analysis with R. This training session focused on core parts of full calculation chain for the NFI cycle 4. The trainings elements were:

- Solving subplot coding errors manually based on time stamps
- Joining subplot level and phase 1 information to trees
- Calculating tree weights, basal area and aboveground biomass
- Aggregating tree aboveground biomass to domain average AGB per ha based on equal area plots and simple averages
- Practicing a custom function to get areas and AGB estimates per ha and totals following double-stage sampling for post-stratification with ratio estimator.

This cycle adopted a more complex method than previous cycles with the introduction of ratio estimators and the inclusion of the Phase 1 observations in the estimation of domain estimates of core variables' averages and totals. This method is more sensitive to stratum assignment and few plots from stratum 4 (non-forest) were found to be forested and have tree records. This may explain the large differences between simple averages and weighted averages for double-stage sampling, in particular for evergreen forest and regenerating vegetation and requires further investigation. These plots may be shifted plots where the land cover was re-assigned but stratum was not.

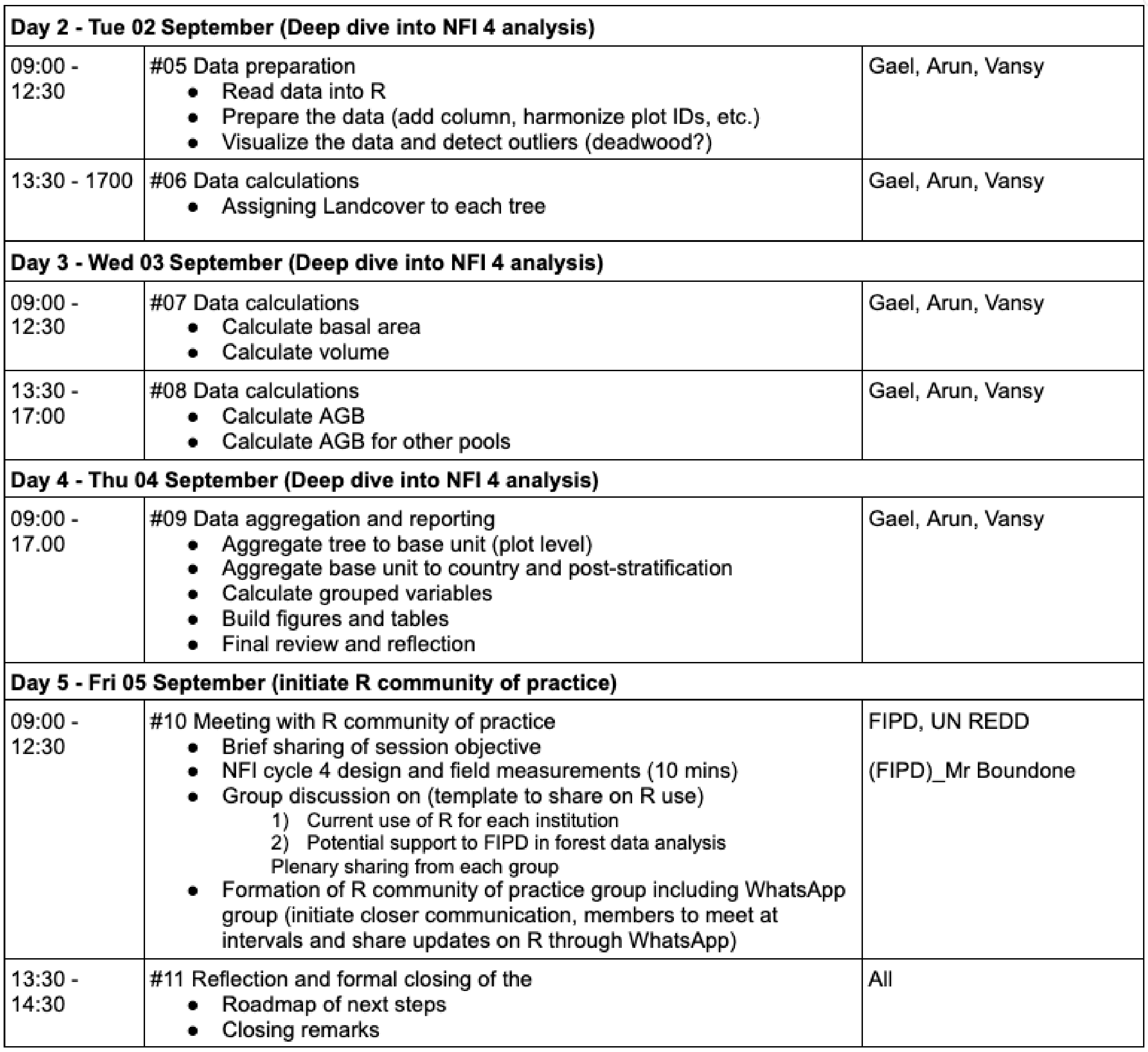
In terms of capacity building, FIPD team expressed interest in adopting R for forest management and small scale forest estimations, which are more frequent requests for them, in order to have more opportunities to practice. Additional training was also suggested to share and practice the calculations for other pools.

ANNEX: Agenda



able e	Topic	Remarks
Day 1 - Mon 01 September (NFI 4 analysis sharing and Refresher)		
9:00 - 9:30	#01 NFI 4 analysis and results sharing Opening session • Welcoming remarks • Objective of the week (Jeremy, Arun, Vansy) • NFI cycle 4 design and field measurements (FIPD)	FIPD representative (Mr Phanousith), Jeremy, Arun *Online link:* *https://teams.microsoft.com/l/meetup-join/19%3ameeting_MDYzYjExNWUtNTcxZC00MjM1LTg3ODktMWViZDI0OTk4Y2Jj%40thread.v2/0?context=%7b%22Tid%22%3a%22163ac468-abb8-44d0-81fd-d9db15e3af96%22%2c%22Oid%22%3a%2217fb7338-3116-4168-80e9-a81b3817cb52%22%7d* *Meeting ID: 380 062 214 986 4* *Passcode: M89UW2AY*
9:30 - 11:30	#02 Presentation of updated calculation chain structure for NFI 4 data analysis • Updated calculation chain • Step by step demo of updated R code • Analysis results: graphs and charts on final results of carbon stocks, tree density and biodiversity (main species, richness)	Gael online presentation Arun, Jeremy and Vansy to support
11:30 - 12:30	#03 Discussions and remarks • Q&A • Closing remarks of NFI sharing session	All
13.30 - 17.00	#04 Refresher training to selected FIPD personnel on Timber volume estimation • Data cleanup • DBH class and volume calculations • Tree volume and weight calculations • Aggregation (Tree to Forest) and visualization • Review and reflection	Arun to lead and Vansy to support (suggest to re-use the NNT volume project for initial refresher)

Figure 2: Agenda part 1



Day 2 - Tue 02 September (Deep dive into NFI 4 analysis)		
09:00 - 12:30	#05 Data preparation • Read data into R • Prepare the data (add column, harmonize plot IDs, etc.) • Visualize the data and detect outliers (deadwood?)	Gael, Arun, Vansy
13:30 - 1700	#06 Data calculations • Assigning Landcover to each tree	Gael, Arun, Vansy
Day 3 - Wed 03 September (Deep dive into NFI 4 analysis)		
09:00 - 12:30	#07 Data calculations • Calculate basal area • Calculate volume	Gael, Arun, Vansy
13:30 - 17:00	#08 Data calculations • Calculate AGB • Calculate AGB for other pools	Gael, Arun, Vansy
Day 4 - Thu 04 September (Deep dive into NFI 4 analysis)		
09:00 - 17.00	#09 Data aggregation and reporting • Aggregate tree to base unit (plot level) • Aggregate base unit to country and post-stratification • Calculate grouped variables • Build figures and tables • Final review and reflection	Gael, Arun, Vansy
Day 5 - Fri 05 September (initiate R community of practice)		
09:00 - 12:30	#10 Meeting with R community of practice • Brief sharing of session objective • NFI cycle 4 design and field measurements (10 mins) • Group discussion on (template to share on R use) 1) Current use of R for each institution 2) Potential support to FIPD in forest data analysis Plenary sharing from each group • Formation of R community of practice group including WhatsApp group (initiate closer communication, members to meet at intervals and share updates on R through WhatsApp)	FIPD, UN REDD (FIPD)_Mr Boundone
13:30 - 14:30	#11 Reflection and formal closing of the • Roadmap of next steps • Closing remarks	All

Figure 3: Agenda part 2

ANNEX: participants list

TBD