**Understanding and using the ‘ausplotsR’ package and AusPlots data**

The accompanying ‘html’ and ‘Rmd’ files contain the material for the "Ecosystem Surveillance (AusPlots): Understanding and using the 'ausplotsR’ package and AusPlots data" tutorial at the ecoED Champions Training 2018 in Canberra. In this tutorial the use of the package ‘ausplotsR’ & the ‘ausplots’ data that can be downloaded with this package are explored.

ausplotsR is an R package for live extraction and preparation of TERN AusPlots ecosystem monitoring data. Through ausplotsR, users can: (1) directly obtain plot-based data on vegetation and soils across Australia, and (2) preprocess these data into structures that facilitate the visualisation and analysis of ausplots data. Data preprocessing includes the computation of species occurrence, vegetation cover, growth form, and basal area.

In Ecosystem Surveillance (AusPlots) section of the workshop the following aspects are covered:

1. ACCESSING AND INSTALLING THE ausplotsR PACKAGE (plus its Dependencies).
2. OBTAIN & EXPLORE AusPlots DATA: get\_ausplots function:
   * get\_ausplots function.
   * Explore the structure of the obtained AusPlots data.
3. MANIPULATING AusPlots DATA:
   * Find the 5 most sampled Bioregions.
   * Subset sites in the 5 most sampled Bioregions (in all DFs in the list).
4. MAP THE SITES
   * Obtain and prepare a map of Australia.
   * Plot AusPlots sites in the 5 most sampled Bioregions on the map of Australia.
5. SPECIES-LEVEL DATA: species\_table function and species occurrence matrices (for the 5 most sampled Bioregions).
   * First step: Create a species occurrence matrix. Compute Species by Site table using the ‘species\_table’ function.
   * Species Abundance/Percent Cover:
     + Percent Cover (Abundance) by Site Visit x Species (i.e. in all ‘cells’)
     + Abundance (Cover %) by Species. Find and plot 4 most abundant species on a map (dot size proportional to Abundance).
   * Species Diversity:
     + Calculate various indices and create a data frame with these indices.
     + Plot 2 indices: Species Richness (from vouchers, more species recorded), and Shannon Index (from veg.PI for abundances) on a map (dot size proportional to the relevant diversity metric) .
6. PROPORTIONAL VEGETATION COVER (= FRACTIONAL COVER): `fractional\_cover’ function.

* First step: Create Fractional Cover against Site Table. Compute using the ‘fractional\_cover’ function.
* Latitudinal pattern in proportional vegetation cover (for a random subset of 200 sites).

1. GROWTH FORM: growth\_form\_table function (for 5 most sampled bioregions)
   * First step: Create Plant Growth Forms Percent Cover against Sites Table. Compute using the ‘growth\_form\_table’ function.
   * Cluster (Hierarchical Clustering) by Plant Growth Forms Percent Cover, colour branches by bioregion.
2. BASAL AREA (OR NUMBER OF BASAL WEDGE HITS): basal\_area function (for 5 most sampled bioregions).
   * First step: Create Basal Area per plot (m2/ha) table. Compute using the ‘basal\_area’ function.
   * Display Basal Areas on map of Australia (dots size proportional to Basal Area).
   * Boxplot of Basal Areas by Bioregion.

**Notes on programming style**

‘<-‘ and ‘=’

R was inspired on the S language. In the earlier incarnations of both S and R languages ‘<-‘ was the only choice of assignment operator. This was a hangover from one of the main influences in the development of S, the language APL. In fact, APL keyboards and AT&T terminals included a key with the ‘<-‘ symbol.

In 2001 the capability to use ‘=’ as an assignment operator to allow programmers in other languages (e.g. C) an easier transition to R. Similarly, S-plus now also accepts ‘=’ as an assignment operator.

In R, ‘<-‘ and ‘=’ assignment operators still differ in their scope. On most occasions both can be used interchangeably, except in function calls.

* ‘*x = 1’* means, “*assign 1 to* x”
* ‘*f(x = 1)’* means “*set the argument x to 1 and call function f*”. To assign a value in a function call will require the following code
* ‘*f(x <- 1)’* which means “*assign 1 to x and call function f”*.

A more concrete example would be:

* + ‘*mean(x <- 1:10)*’. This will return a value of 5.5 and create a vector *x* containing values of ‘*1 2 3 4 5 6 7 8 9 10*’.
  + ‘*mean(x = 1:10)*’. Also returns a value of 5.5, but doesn’t create a vector x.

Thus, ‘<-‘ and ‘=’ are mostly interchangeable except in function calls. However, constructs where assignment occur within a function are not recommended and they should not be used unless you have a good reason to do so.

There is a strong following for the ‘<-‘ assignment operator, perhaps due to tradition. Hadley Wickham’s and Google R encourage the use ‘<-‘ over ‘=’. However, Robert and Casella recommend using ‘=’ in their book ‘Introducing Monte Carlo Methods with R’.

In my scripts I use ‘=’ mostly because it is faster to type. In addition, it helps avoiding some errors, such as typing ‘*x < -3*’ instead of ‘*x <- 3*’.

If you are unhappy with the use of ‘=’ as an assignment operator you could use one of the ‘*tidy \**’ functions (e.g. ‘*tidy\_source*’) in the R library ‘*formatR*’ to replace ‘=’ with ‘*<-*‘. Some authors might prefer to write their code using ‘=’ and then replace these symbols with ‘<-‘ using this library when releasing their code publicly. I thought of this approach, but I finally decided to keep it honest/true to the way I work!

Commented code

In the script, you will find numerous commands that are commented out; that is, they have the ‘#’ symbol in front. These commands are not necessary for the running the script (i.e. performing data manipulations or analyses or creating graphs). I have included them because they might help you understand intermediate steps, typically by checking the value of an R object. They are commented out for brevity, as they are not necessary. However, if you need help or are curious you might want to remove the comment and run the code.