**Simple Change Detection in Raster using TERN web services**

The accompanying ‘html’ and ‘Rmd’ files contain the Landscape Assessment (AusCover) section of TERN’s workshop at the Ecological Society of Australia 2018 conference. This section of the workshop is based on Peter Scarth’s tutorial "Simple Change Detection in Raster using TERN web services and Google Earth Engine". Although it (more or less) follows Dr. Scarth’s tutorial steps, it has some variation in its content. Dr. Scarth’s tutorial uses a Jupyter Notebook with code in Python. In the workshop, R is used.

To analyse changes in Normalized Difference Vegetation Index (NDVI) over time, datasets containing composited seasonal surface reflectance images (4 seasons per year) are used. These datasets were created from full time series of Landsat TM/ETM+ imagery. The imagery was composited over a season to produce new imagery that is representative of the period. In this process, techniques that reduce contamination by cloud and other problems were used.

The final part of this section of the workshop explores the causes of NDVI change in the area of most noticeable greening. To do so, a time series of the green fraction is plotted. The green fraction is one of the bands in the fractional cover product. This is another AusCover product hosted by TERN.

In this section of the workshop, the following aspects are covered:

1. INTRODUCTORY STEPS (getting acquainted with using raster files in R):

* Download files from remote systems.
* Open raster files.
* Explore raster files.
* Subset (i.e. crop) raster files.
* Combine raster files into a multi-layered raster object.
* Visualise a multi-layer raster object.
* Re-project a raster object.
* Open a remote map to find our bearings.

1. NORMALIZED DIFFERENCE VEGETATION INDEX (NDVI) CHANGE ANALYSIS:

* Explore an Area where NDVI Change might have occurred
* Conduct Tasks in the ‘Introductory Steps’ part (i.e. Download, Read, Subset, and Combine the Landsat Surface Reflectance Data) by:
  + Creating a function to perform these tasks.
  + Call this function on datasets 10 years apart: 2007 and 2017.
* Compute NDVIs: Raw & Normalised
* NDVI Change Analysis: Numerical & Visualisation
* NDVI Time Series of Green Visualisation
* Explore NDVI Change in an Area of Notable Greening.

**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.