Module 4: GitHub and Pair coding

FREC 3004: Environmental Informatics

Objectives

The primary goal of this exercise is to gain experience working collaboratively to develop a scientific workflow. As such, this assignment is best completed with a partner. Specifically, we will outline a simple analysis, break the overall job into parts, and have each person complete part of the project. To put these parts together we will be using Github. Along the way we will also be exploring the statistical concept of Likelihood.

Prairie Phenology

The goal of our analysis is to investigate the phenological state of the U. Illinois Tall Grass Prairie. We will be using green chromatic coordinate (GCC), which is a color index. Before building the workflow you are encouraged to take a look at the site https://phenocam.sr.unh.edu/webcam/sites/uiefprairie/ and the raw csv data https://phenocam.sr.unh.edu/data/archive/uiefprairie/ROI/uiefprairie_GR_1000_1day.csv

The workflow for this analysis with have three components:

1. Download PhenoCam data for the U. Illinois Tall Grass Prairie site
2. Visualize the data with a mean and 95% confidence interval
3. Fit a simple logistic model to spring leaf out for one specific year

From this overall design, let's next outline the specific steps involved as pseudocode

```
### Prairie Phenology Workflow

## Download phenology data

## Plot overall phenology data

## Create and visualize subset of data for leaf out

## Fit logistic model

## Visualize model and data
```

Modular Design

From this overall design we can look for ways to modularize the analysis. One feature that jumps out is that we need to visualize the data three times, so we should definitely make a function to do that. The inputs to the function would be an x-axis (`date`), y-axis (`gcc_mean`), and error estimate (`gcc_std`), which we might pass as a dataframe for convenience. Since this is a graphing function we'd also like the ability to set all sorts of plot characteristics, which can be done in R by passing `...` as an argument and then passing that on to the internal *ggplot* call. The proposed function interface would thus be

```
##' Plot Phenocam data
##'
##' @param dat  dataframe of date, gcc_mean, gcc_std
##' @param ...  additional graphing parameters
```

```
plot_phenocam <- function(dat,...)
```

Next, because the raw data will be downloaded off the web and has embedded meta-data to handle, let's go ahead and create a download function. This function just needs to know the URL for where to find the data. Unlike the plot function, this function will return something (the data that was downloaded), so it would be good design to document what is returned and how it will be formatted

```
##' Download Phenocam data
##'
##' @param URL  web address where data is located
##' @return data.frame with days as rows, variables as columns
download_phenocam <- function(URL)
```

Let's also create a function to fit the logistic model to the spring leaf-out data, since we could easily see applying this same function to other data sets. The input to such a fit would obviously be the same data.frame that we're using to make the plot. To do the fit itself we'll use non-linear least squares, so in addition to the data we'll need to provide an initial guess at the model parameters, which we'll pass on to the numerical optimization. We'll also want to return the the estimated parameters for the model.

```
##' Fit logistic model
##'
##' @param dat  dataframe of day of year (doy), gcc_mean, gcc_std
##' @param par  vector of initial parameter guess
##' @return  output from numerical optimization
fit_logistic <- function(dat, par)
```

Finally, because we'll want to make a plot of the logistic model after we're done, let's create a function for performing the model calculation. This function will also come in handy within the `fit_logistic` function.

```
##' Logistic model
##'
##' @param theta  parameter vector
##' @param x      vector of x values
##' @return vector of model predictions
pred_logistic <- function(theta, x)
```

At this point we've spent a good bit of time up front on organization – we have a detailed plan of attack and have thought carefully about what each module is responsible for doing. Each task has well-defined inputs, outputs, and goals. Rather than facing a thankless job of documenting our code after we're done, even though we haven't written a single line of code yet we are largely done with our documentation. What remains to do is implementation.

Task 1: Create & Clone Repository

Because we're going to employ version control in our project, our first step is to create the repository that our project will be stored in. **To ensure that both you and your partner get to see every step of how to work with version control, for the rest of this exercise you are going to complete every step twice, once from the perspective of the OWNER of the repository and once as the COLLABORATOR**.

OWNER

1. Go to your account on github.com and under the Repositories tab click on the "New" button with a picture of a book on it
2. Choose a name for your repository (make sure it's different from your partner's)
3. Click the "Initialize this repository with a README" checkbox
4. Optionally also provide a Description, Add a license (e.g. MIT), and add R to the .gitignore
5. Click "Create Repository"

6. Copy the URL of your new repository by clicking the clipboard icon
7. To clone the repository,open up RStudio Cloud. Select the down arrow that is beside the New Project Button. You should see "New Project from GitHub Repository". Select this using the URL from #6.

Task 2: Add the first function: download_phenocam

Within this project we'll create separate files for each part of the analysis. To make the order of the workflow clear we'll want to name the files systematically. In the first file we'll implement the `download_phenocam` function

```
##' Download Phenocam data
##'
##' @param URL  web address where data is located
download_phenocam <- function(URL) {
  ## check that we've been passed a URL
  if (length(URL) == 1 & is.character(URL) & substr(URL, 1, 4) == "http") {

      ## read data
      dat <- read_csv(URL, skip = 22)

      ## convert date
      dat$date <- as_date(as.character(dat$date))

      return(dat)
  } else {
    print(paste("download.phenocam: Input URL not provided correctly", URL))
  }
}
```

OWNER

1. In RStudio, click File > New File > R Script
2. Copy and Paste the above function into this file
3. Save the file as `01_download_phenocam.R`
4. From the Git tab, click the "Staged" box next to the file you just created. This is equivalent to *git add*
5. Click Commit, enter a log message, and click Commit. This is equivalent to *git commit*
6. To push the change up to Github click on the green up arrow. This is equivalent to *git push*

Task 3: Collaborator adds plot_phenocam

With the first function complete, let's now imagine that a **COLLABORATOR** has been tasked with adding the second function. To do so they must first fork and clone the repository

COLLABORATOR

1. Go to Github and navigate to the project repository within the OWNER's workspace.
2. Click Fork, which will make a copy of the repository to your own workspace.
3. Copy the URL to your own version and follow the instructions above for cloning the repository in RStudio.
4. Open a new file, enter the code below, and then save the file as `02_plot_phenocam.R`

```
##' Plot Phenocam data
##'
##' @param dat  dataframe of date, gcc_mean, gcc_std
##' @param ...  additional graphing parameters
plot_phenocam <- function(dat, pred = NULL){
```

```
  if(!is.null(dat)){

    dat <- dat %>%
      mutate(ylo = gcc_mean - 1.96 * gcc_std,
             yhi = gcc_mean + 1.96 * gcc_std)

    ##
    p <- ggplot(dat, aes(x = date)) +
      geom_ribbon(aes(ymin = ylo, ymax = yhi),
                            alpha = 0.70,
                            fill = "lightblue") +
      geom_point(aes(y = gcc_mean)) +
      labs(x = "Date", y = "GCC_90", title = "U. Illinois Tall Grass Prairie (2015)")

    if(!is.null(pred)){
      p <- p  +
        geom_line(data = pred, aes(x = date, y = pred))

    }

    p

  } else {
    print("plot_phenocam: input data not provided")
  }

}
```

5. Follow the instructions above to Add, Commit, and Push the file back to your Github repository
6. Next you want to perform a "pull request", which will send a request to the OWNER that they pull your new code into their mainline version. From your Github page for this project, click **New Pull Request**.
7. Follow the instructions, creating a title, message, and confirming that you want to create the pull request

OWNER

1. Once the COLLABORATOR has created the pull request, you should get an automatic email and also be able to see the pull request under the "Pull Requests" tab on the Github page for the project.
2. Read the description of the proposed changes and then click on "Files Changed" to view the changes to the project. New code should be in green, while deleted code will be in pink.
3. The purpose of a pull request is to allow the OWNER to evaluate the code being added before it is added. As you read through the code, if you hover your mouse over any line of code you can insert an inline comment in the code. The COLLABORATOR would then have the ability to respond to any comments. In larger projects, all participants can discuss the code and decide whether it should be accepted or not. Furthermore, if the COLLABORATOR does any further pushes to Github before the pull request is accepted these changes will automatically become part of the pull request. While this is a very handy feature, it can also easily backfire if the COLLABORATOR starts working on something different in the meantime. This is the reason that experienced users of version control will use BRANCHES to keep different parts separate.
4. Click on the "Conversation" page to return where you started. All participants can also leave more general comments on this page.
5. If you are happy with the code, click "Merge Pull Request". Alternatively, to outright reject a pull request click "Close pull request"

Task 4: Owner adds pred.logistic and fit.logistic

We are now past the 'set up' stage for both the OWNER and the COLLABORATOR, so for this task we'll explore the normal sequence of steps that the OWNER will use for day-to-day work

OWNER

1. Pull the latest code from Github. In RStudio this is done by clicking the light blue down arrow on the Git tab. This is equivalent to the command line *git pull origin main* where origin refers to where the where you did your original clone from and main refers to your main branch (if you use branches you can pull other branches)
2. Next, open up a new R file, add the code below, and save as `03_logistic.R`

```
##' Logistic model
##'
##' @param theta  parameter vector
##' @param x      vector of x values
##' @return vector of model predictions
pred_logistic <- function(theta, x){
   z <- exp(theta[3]+theta[4]*x)
   Ey <- theta[1]+theta[2]*z/(1+z)
}

##' Fit logistic model
##'
##' @param dat  dataframe of day of year (doy), gcc_mean, gcc_std
##' @param par  vector of initial parameter guess
##' @return  output from numerical optimization
fit_logistic <- function(dat, par){

  fit <- nls(gcc_mean ~ pred_logistic(theta = theta, x = doy), data = dat, start = list(theta = par))

  fit$m$getPars()

}
```

3. As before, add your new file under the Git tab, Commit the change, and push it back to Github

To estimate the parameters in the logistic model we will use a non-linear least squares approach that finds the parameters that minimize the difference between the model prediction and the observations. To do this we need to define the non-linear model and plug that in the non-linear least squares function.

We are looking for the value of θ, the vector parameters in the logistic model, that minimize the difference between the model prediction and the observations .

The code for this comes in two parts.

First is the logistic model itself, `pred_logistic`, which translates the equation

$$\theta_1 + \theta_2 \frac{exp(\theta_3 + \theta_4 x)}{1 + exp(\theta_3 + \theta_4 x)}$$

into code. The logistic has an overall S shape, with θ_1 defining the minimum and $\theta_1 + \theta_2$ defining the max. The midpoint of the curve – the x value where the function is halfway between the minimum and maximum – occurs at $-\theta_3/\theta_4$, while the slope at that point is $\theta_4/4$. x is the day of year (doy).

Second, is the part runs the non-linear least squares in R (`nls`) using the `pred_logistic` function to predict `gcc_mean`. The data are the data frame `dat` and start are the first parameters used by the algorithm that searches for the best parameters.

`fit <- nls(gcc_mean ~ pred_logistic(theta = theta, x = doy), data = dat, start = list(theta = par))`

them

`fit$m$getPars()`

extracts the best parameters from the object `fit`

Task 5: Collaborator adds the main script

The day-to-day workflow for the COLLABORATOR is similar, but not exactly the same as the OWNER. The biggest differences are that the COLLABORATOR needs to pull from the OWNER, not their own repository, and needs to do a pull request after the push.

COLLABORATOR

1. Pull from OWNER. Unfortunately, this has to be done from the command line rather than the pull button within RStudio, which just pulls from the COLLABORATOR's repository. In RStudio go to Tools > Shell to open a terminal
2. At the terminal type

   git pull URL main

where URL is the address of the OWNER's Github repository. Because it is a pain to always remember and type in the OWNER's URL, it is common to define this as *upstream*

`git remote add upstream URL`

which is a one-time task, after which you can do the pull as

`git pull upstream main`

3. Open a new Rmd file and add the code below. This code just flushes out the pseudocode outline we started with at the beginning of this activity.

```
library(tidyverse)
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.0 --
## v ggplot2 3.3.3     v purrr   0.3.4
## v tibble  3.1.0     v dplyr   1.0.5
## v tidyr   1.1.3     v stringr 1.4.0
## v readr   1.4.0     v forcats 0.5.1
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```
library(lubridate)
```

```
##
## Attaching package: 'lubridate'
## The following objects are masked from 'package:base':
##
##     date, intersect, setdiff, union
```

```
## Main script for phenology analysis

## Load required functions
if(file.exists("01_download_phenocam.R")) source("01_download_phenocam.R")
```

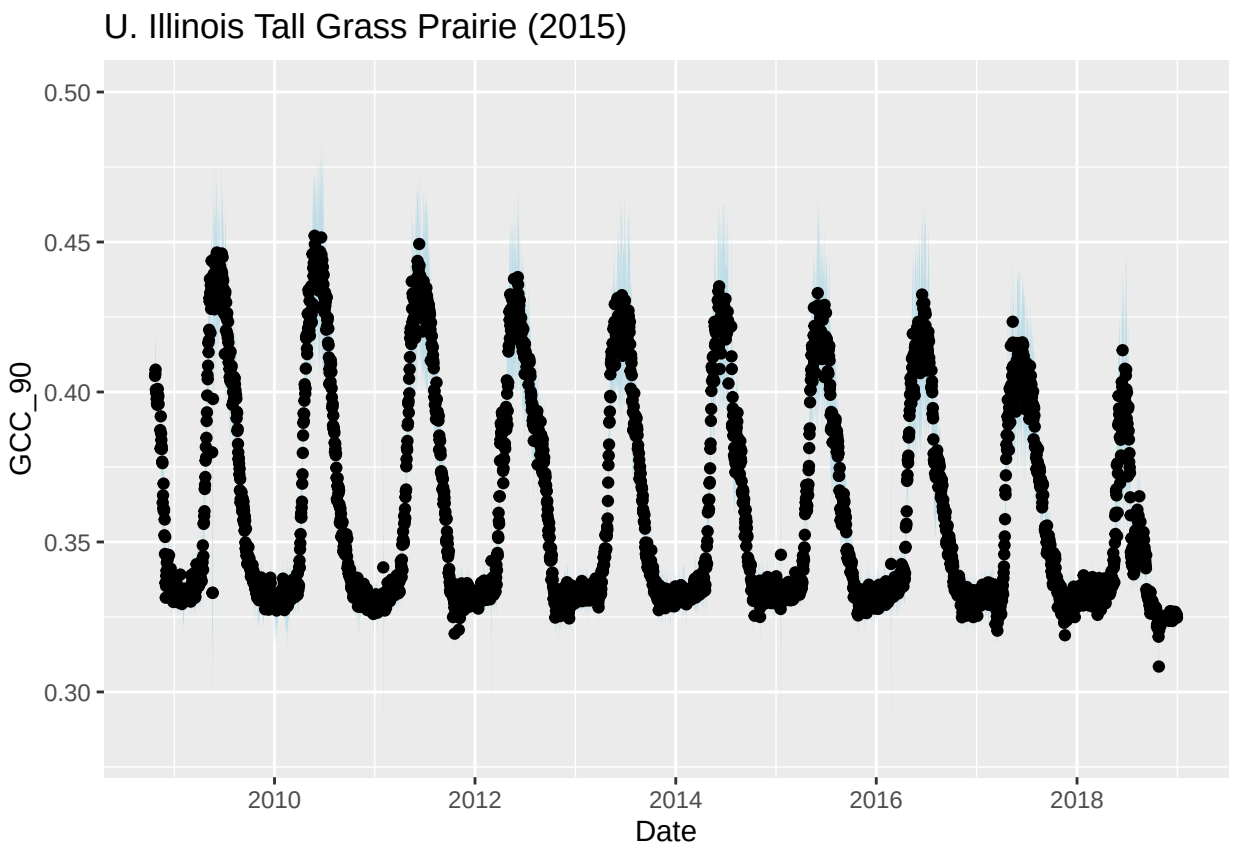
```
if(file.exists("02_plot_phenocam.R"))     source("02_plot_phenocam.R")
if(file.exists("03_logistic.R"))          source("03_logistic.R")

## Download phenology data
URL <- "http://phenocam.sr.unh.edu/data/archive/uiefprairie/ROI/uiefprairie_GR_1000_1day.csv"
prairie_pheno <- download_phenocam(URL)
```

```
##
## -- Column specification --------------------------------------------------------
## cols(
##   .default = col_double(),
##   date = col_date(format = ""),
##   midday_filename = col_character(),
##   snow_flag = col_logical(),
##   outlierflag_gcc_mean = col_logical(),
##   outlierflag_gcc_50 = col_logical(),
##   outlierflag_gcc_75 = col_logical(),
##   outlierflag_gcc_90 = col_logical()
## )
## i Use `spec()` for the full column specifications.
```

```
## Plot overall phenology data
plot_phenocam(prairie_pheno)
```

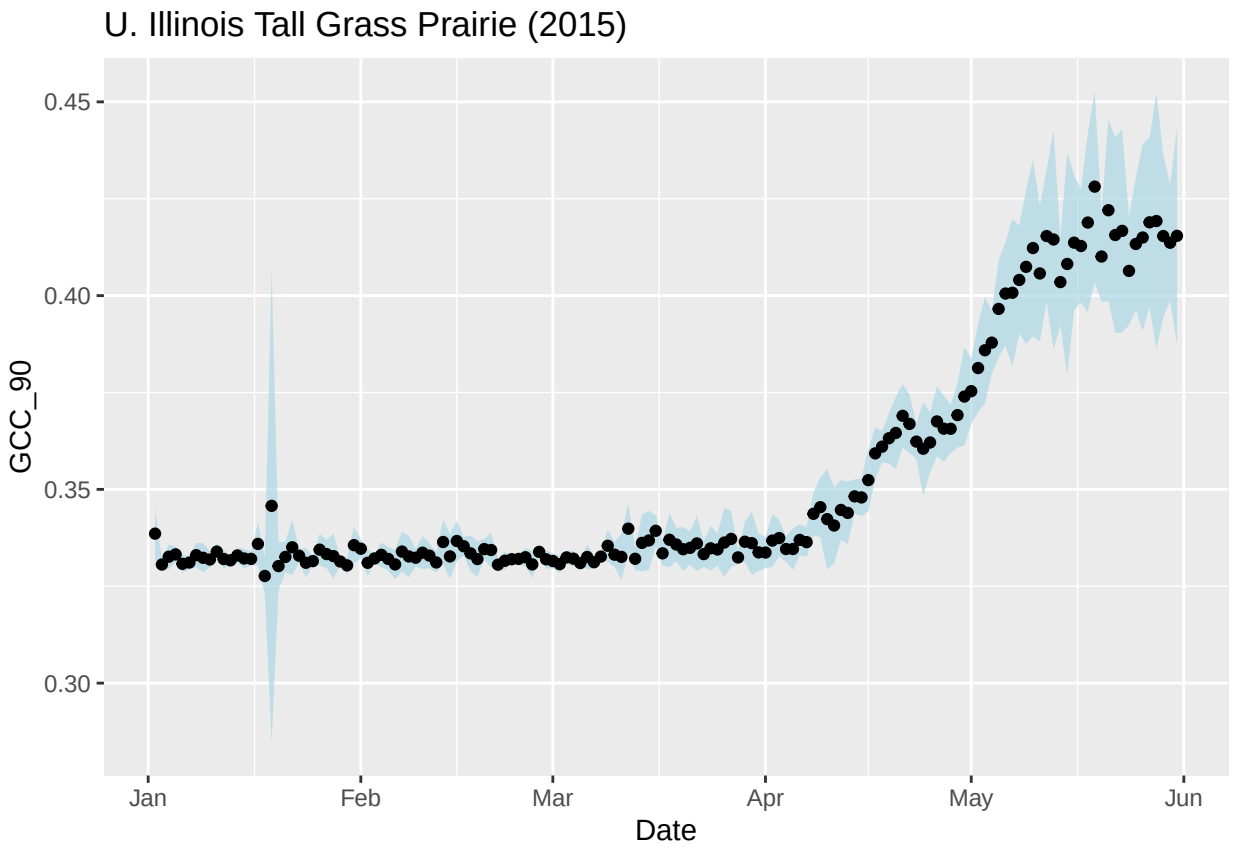
```
## Warning: Removed 65 rows containing missing values (geom_point).
```




```
## Create and visualize subset of data for leaf out
spring <- as_date(c("2015-01-01","2015-06-01"))
```

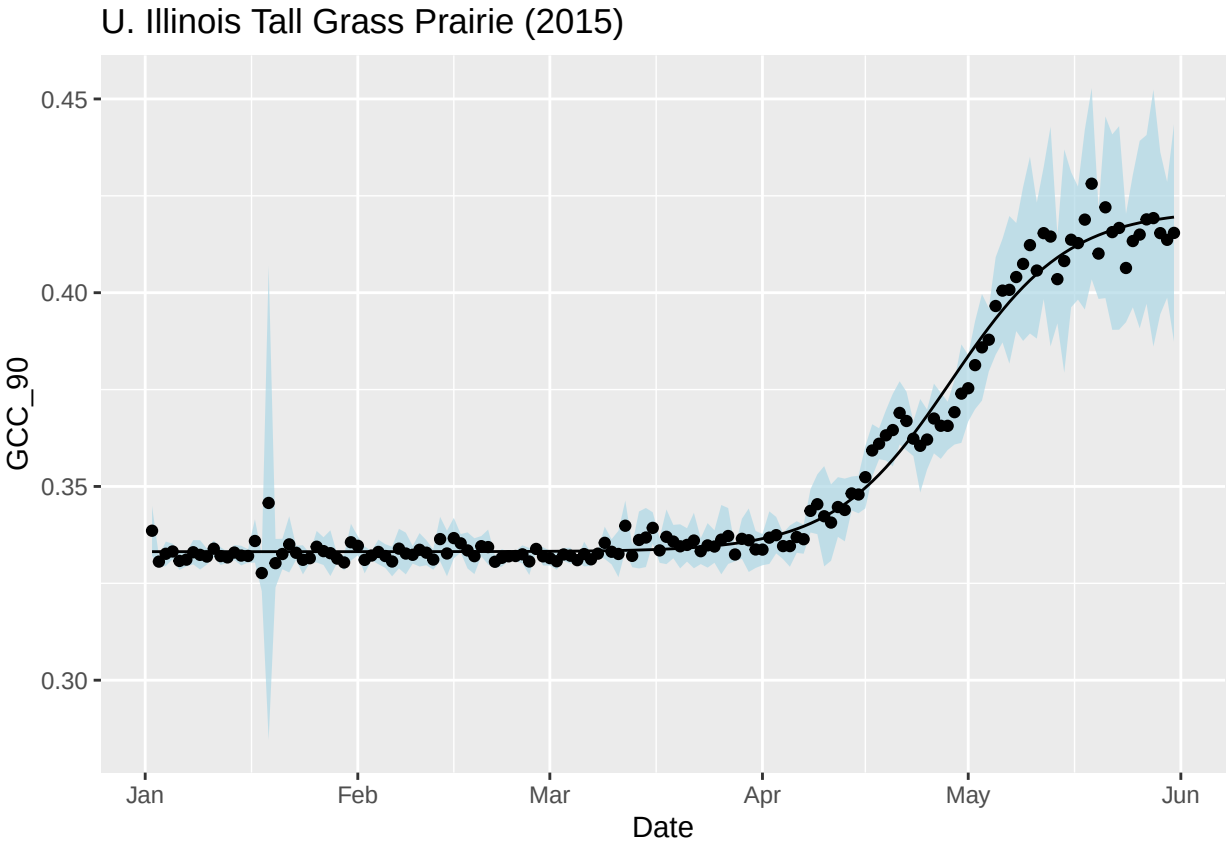
```
dat <- prairie_pheno %>%
  filter(date > spring[1],
         date < spring[2]) %>%
  select(date, gcc_mean, gcc_std)

plot_phenocam(dat)
```




```
## Fit logistic model
dat <- dat %>%
  mutate(doy = yday(date))
par <- c(0.33,0.11,-10,0.1)
fit_pars <- fit_logistic(dat, par)
pred <- tibble(date = dat$date,
               pred = pred_logistic(fit_pars, dat$doy))

## Visualize model and data
plot_phenocam(dat, pred = pred)
```



4. Save this file as `04_Main.Rmd`.
5. Within RStudio's Git tab, add the file and Commit. Use the Push (up arrow) button to push this to your own repository
6. On Github.com, submit a pull request

OWNER

1. Evaluate and accept pull request.

At this point your workflow should be complete and you should be able to run the analysis.

Task 6: Knit and post document

OWNER

1. Pull updated files to local computer (or Rstudio Cloud)
2. Knit script to produce a PDF document (see Knit to PDF)
3. Commit and push PDF to GitHub.

Assignment requirements

For your assignment, please submit the url for your GitHub repository to Canvas

Acknowledgements

This module is based on a similar module developed by Mike Dietze at Boston University: https://github.com/EcoForecast/EF_Activities/blob/master/Exercise_04_PairCoding.Rmd