Automation tutorial - Advanced R Workshop

Jessica Clayton

30 Januar 2020

In this section you will learn how to automate data import and graph making by creating functions, using the map function and lists.

purrr::map

Apply a function to each element of a vector

```r
# create a vector of 5 integers
n <- c(6, 3, 7, 2, 10)
```

You want to add 1 and substract 1 from each number

```r
# vector for + and - 1
m <- c(1, -1)
```

What happens when you try n + m?

```r
n + m
```

```
## Warning in n + m: Länge des längeren Objektes
##       ist kein Vielfaches der Länge des kürzeren Objektes
```

```
## [1]  7  2  8  1 11
```

Essentially what happened was:

n[1] + m[1], n[2] + m[2], n[3] + m[1], n[4] + m[2], n[5] + m[1]

As m is a shorter vector than n, the elements of m get repeated. This is not the outcome we wanted.

The map functions in the purrr transform their input by applying a function to each element and returning a vector the same length as the input.

```r
library(purrr)
```

```r
map(m, ~ n + .) # '.' is a placeholder
```

```
## [[1]]
## [1]  7  4  8  3 11
##
## [[2]]
## [1] 5 2 6 1 9
```

When the output is saved, R automatically saves both vectors within a list:

```r
x <- map(m, ~ n + .)
```

Lists

[Note from Jess: I need to think of a good explanation about lists. Basically lists store all kinds of objects within the shell of one object within the environment. You can have a list which consists of all identical length vectors or identical length dataframes, or all non-identical vectors and dataframes, or a mixture of dataframes and vectors. Just look at the lists which are created when you save the output of a linear model

An advantage is that you can apply a function to each element of your list.]

```
x.1 <- map(x, ~ .^2) # square each element
x.2 <- map(x, ~ mean(.)) # calculate the mean of each vector
```

Functions

You can create a function to perform anything you like:

```
# square a number
fun1 <- function(x){
  x^2
}

fun1(4) # when x = 4
```

```
## [1] 16
```

```
# for many numbers
fun1(n)
```

```
## [1]  36   9  49   4 100
```

Using functions in conjunction with map

```
y <- map(n, ~fun1(.)) # use the function you created

x.2 <- map(x, ~ mean(.)) # use existing functions such a mean
```

[This might seem a bit complicated for such simple tasks but it will come clearer why it is useful later]

Automated data import

To merge all files into one dataframe: map_df from purrr

```
# Create a list with the names of all files you wish to import into your environment
files_names <- list.files("File_path/", pattern = "*.csv")

# (this only works if the tables have identical column names and dimensions)
All_data <- map_df(files, ~read.csv(paste0("File_path/", .)))
```

Alternatively work using lists

When your data are not identical in dimensions or file names, it might be better to import the data to a list.

```
# first define an empty list
My.list <- list()

# use the read.csv function with map to store each file as a separate data frame within your list
```

```r
My.list <- map(files, ~ read.csv(paste0("File_path/", .)))

# access each df from list
My.list[[1]]

# name each dataframe with the file name
names(My.list) <- files

My.list[["Filename1.csv"]]
```

Data manipulation accross all dataframes in the list

Tidyr and dplyr are especially useful here

```r
library(dplyr)
library(tidyr)


# E.g create a new column with changed date format
My.list.2 <- map(My.list, ~mutate(.x, New_Date.format = as.Date(Date,  format = "%d.%m.%Y")))

# Delete a column , e.g. a comments column
My.list.2 <- map(My.list2, ~ select(.x, -Comments))

# Perform many operations using the pipe function

My.list.2 <- My.list %>%
  map(~mutate(., O2.L = O2_ml/1000)) %>% # change units from ml to L
  map(~select(., -c(Channel, Treatment))) # remove columns
```

Graph functions

For graph automation using functions, map and ggplot2

A function for graphs with a common x axis

```r
library(ggplot2)

# define which columns you want to plot against x
vars <- names(My.data)[5:10]

# make a function with ggplot where y is the placeholder for your y variable:
fun1.1 <- function(y) {
  My.data %>%
    ggplot() +
    aes(Date, get(y), col = Land.use) + #get(y) removes "" from the variable name
    geom_point() +
    theme_bw() +
    xlab("Date") +
    ylab(y) # don't forget to label axis
}

# map passes each element of the vars vector through the graph function as y
# and thus creates a separate graph for each variable that you defined
map(vars, fun1.1)
```

A function for subsetting data and creating graph for each treatment level

```r
# create a vector for soil type:
ST.filter = levels(My.data$Soiltype)

# e.g.
fun2 <- function(x) {
  My.data %>%
    filter(Soiltype == x) # x is the placeholder for the soil type level
    ggplot() +
    aes(Date, Total.carbon , col = Land.use) +
    geom_point() +
    theme_bw() +
    xlab("Date") +
    ggtitle(x) # don't forget to make a title so you know which soil type is being shown
}
map(ST.filter, fun2)
```

Save the graphs to your computer

Use ggsave()

```r
fun2 <- function(x) {
  My.data %>%
    filter(Soiltype == x) # x is the placeholder for the soil type level
    ggplot() +
    aes(Date, Total.carbon , col = Land.use) +
    geom_point() +
    theme_bw() +
    xlab("Date") +
    ggtitle(x) # don't forget to make a title so you know which soil type is being shown

  # create a unique file name by using x as placeholder soil type in the file name
  ggsave(paste0("File_path/Total.C_", x, ".png"))
}

map(ST.filter, fun2)
```

More automated graph options:

Grids usings facet_wrap_paginate from ggforce

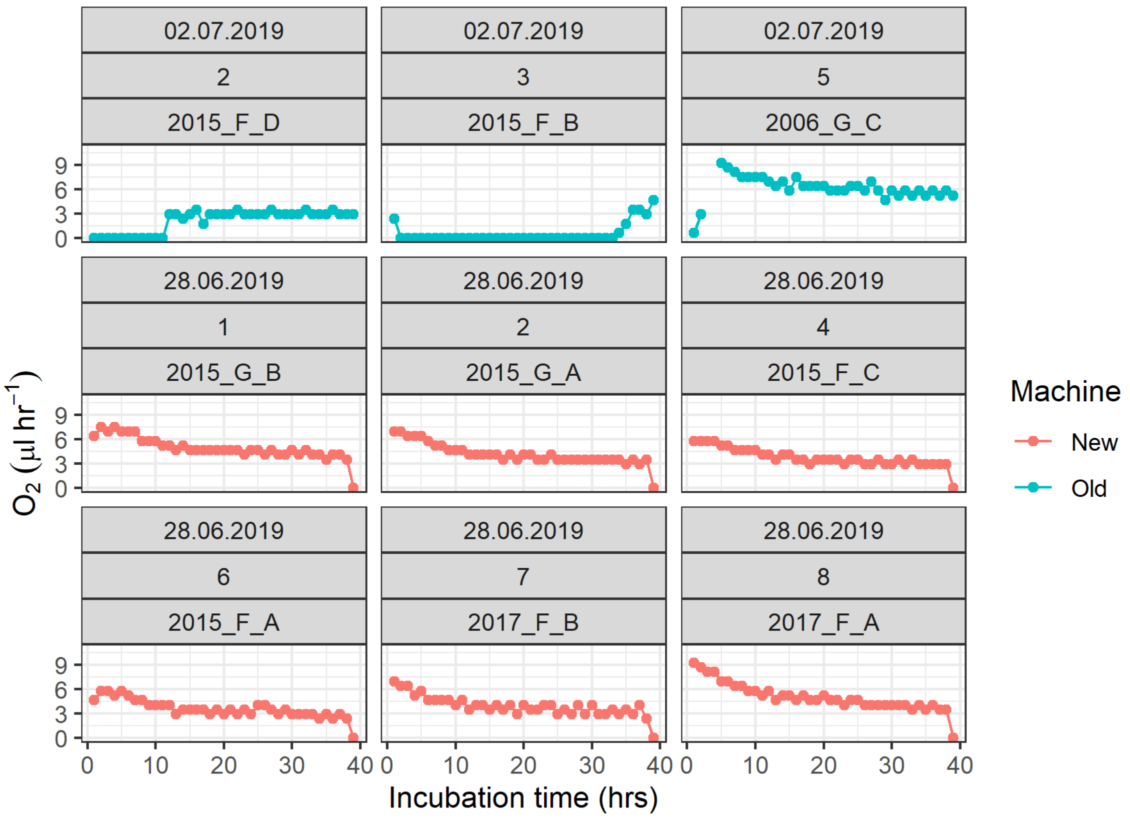
Another way I use automation in my graphing is to quickly check if all the channels on the respiration machines are functioning correctly after I have done a sample run. There are roughly 40 respiration channels combined from two respiration machines (old machine and new machine). Sometimes the channels aren't working or there was an issue with the set up, but this can sometimes only be seen once we look directly at the respiration data.

Soil respiration rate (O2 µl/hr) was measured with the machines at hourly intervals over 24 hours. The data provided is semi processed raw data from the respiration machines and contains the following information:

Sample_ID, Date of run, Machine (old/new), Channel of the machine, Time (hrs) and Respiration rate (O2_muL_hr)

I wanted a quick way to see the respiration rate for each channel at a glance. It would be too much information for one graph, so one graph per channel was preferable. Scrolling through up to 40 graphs at a time was also not ideal, so I needed to make plot with a grid of many individual plots.

As with previous codes, I created a function for my graph, this time I used the facet_wrap_paginate from ggforce to wrap the data to Channel, Date and Sample_ID level. I chose a 3 x 3 grid:



The facet_wrap_paginate function has a page number option. The default page = 1, shows only the first 9 graphs. To get the next 9, use page = 2, etc.

I used the following graph function with map to get all the pages I needed.

Here is the code:

```r
load(file = "O2_data.RData")


graph_fun = function(x) {
  O2.data %>%
    ggplot() +
    aes(Time_hrs, O2_muL_hr, col = Machine) +
    geom_point() +
    geom_line() +
    theme_bw() +
    ylab(O[2]~(mu*l~ hr^-1)) +
    xlab("Incubation time (hrs)") +
    facet_wrap_paginate(vars(Date, Channel, Sample_ID), ncol = 3, nrow = 3, page = x)
    # note that x is the place holder for number of pages
}

# I know that there shouldn't be more than 5 pages worth of graphs
# so I set I map to use 1:5 in the graph function
map(1:5, ~graph_fun(.x) )
```

Scroll through the plots in the RStudio environment using the arrows. Or save to your hard drive using ggsave().

Quick graphs with mean and error for many variables

This code utilises mutate_at to calculate mean and sd for many variables and reshapes the data using gather and spread to create a table where it is easy to graph the mean with errorbars for many variables against a common x axis.

Sample data: Chronosequence soils

Soil samples were taken from the restored land after coal mining from 13 sites aged 1-55 years from time of restoration to time of sampling. For each site there were two land use factors (Arable/Grass margin). 5 replicates were taken for each site and land use type and analysed for nutrient content, microbial biomass and soil respiration.

The task is to see how soil properties change with increasing age after restoration, comparing the arable to grass margins. Mean and standard deviation per site (age) and land use should be calculated and plotted against soil age for each soil property.

```
load(file = "Chrono.RData")


str(Chrono)
```

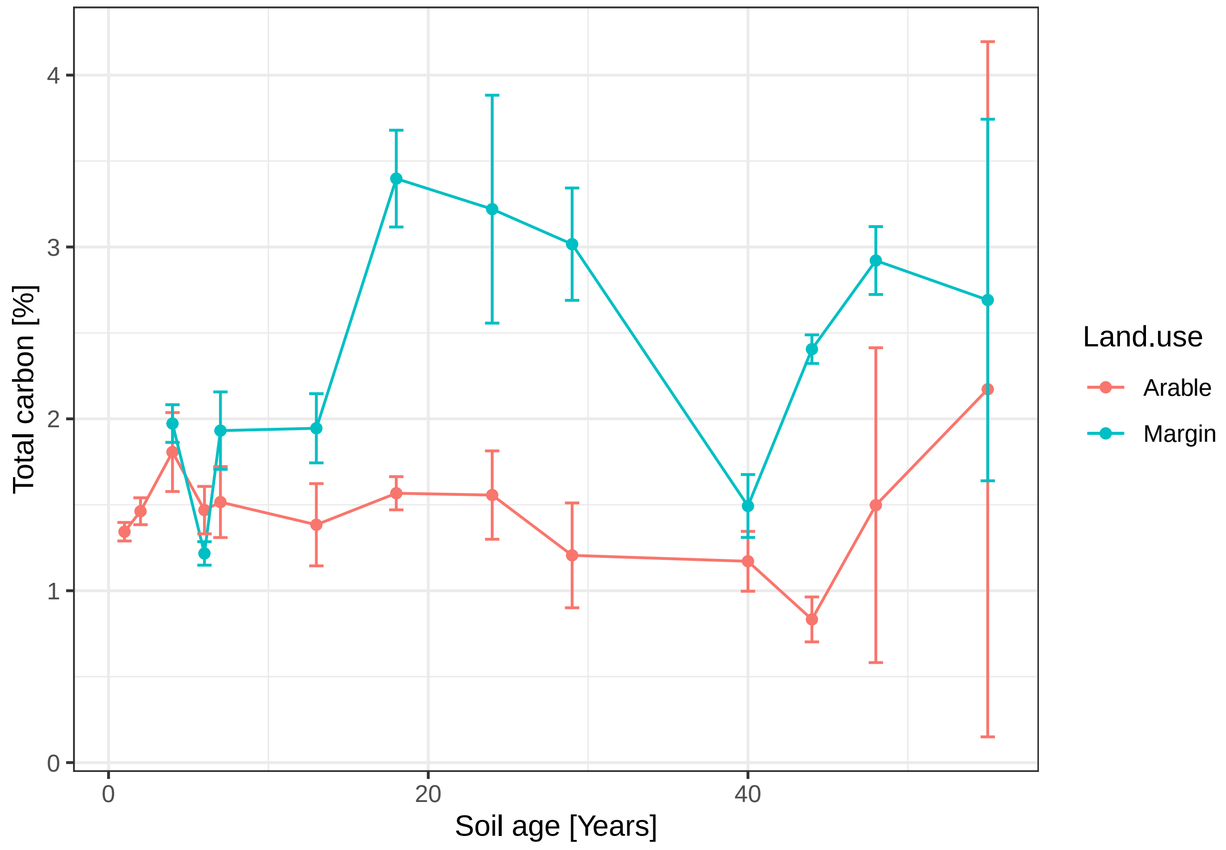
```
## 'data.frame':    125 obs. of  8 variables:
##  $ Age      : int  55 55 55 55 55 55 55 55 55 55 ...
##  $ Land.use : Factor w/ 2 levels "Arable","Margin": 1 1 1 1 1 2 2 2 2 2 ...
##  $ Replicate: Factor w/ 5 levels "A","B","C","D",..: 1 2 3 4 5 1 2 3 4 5 ...
##  $ Ctot.pc  : num  1.15 1.47 1.3 1.16 5.78 ...
##  $ N.pc     : num  0.0836 0.0667 0.097 0.0899 0.0898 ...
##  $ Cmic     : num  0.981 1.008 1.236 1.021 1.043 ...
##  $ Nmic     : num  0.1055 0.078 0.1551 0.0889 0.1206 ...
##  $ BAS.O2   : num  1.29 1.39 1.77 1.36 1.42 ...
```

```
# define a vector list of the name of the variables you want to plot
vars <- c("Ctot.pc", "N.pc", "Cmic", "Nmic", "BAS.O2")

# calculate mean and sd per variable
# use summarise_at with the defined variable vector
Chrono_mean <- Chrono %>%
  group_by(Age, Land.use) %>%
  summarise_at(vars,  funs(mean(., na.rm=TRUE), sd(., na.rm = T)) )


head(Chrono_mean)
```

```
## # A tibble: 6 x 12
## # Groups:   Age [4]
##     Age Land.use Ctot.pc_mean N.pc_mean Cmic_mean Nmic_mean BAS.O2_mean
##   <int> <fct>           <dbl>     <dbl>     <dbl>     <dbl>       <dbl>
## 1     1 Arable           1.34    0.0227     0.190    0.0102       0.664
## 2     2 Arable           1.46    0.0268     0.345    0.0560       1.30
## 3     4 Arable           1.81    0.0343     0.363    0.0359       1.20
## 4     4 Margin           1.97    0.0279     0.346    0.0449       1.39
## 5     6 Arable           1.47    0.0450     0.452    0.0137       1.15
## 6     6 Margin           1.22    0.0474     0.589    0.120        1.97
## # ... with 5 more variables: Ctot.pc_sd <dbl>, N.pc_sd <dbl>, Cmic_sd <dbl>,
## #   Nmic_sd <dbl>, BAS.O2_sd <dbl>
```

Basic graph code:

```
Chrono_mean %>%
  ggplot() +
  aes(Age, Ctot.pc_mean, col = Land.use) +
  geom_point() +
  geom_line() +
  geom_errorbar(aes(ymin = Ctot.pc_mean - Ctot.pc_sd, ymax = Ctot.pc_mean + Ctot.pc_sd)) +
  theme_bw() +
  xlab( "Soil age [Years]") +
  ylab("Total carbon [%]")
```



To make a similar graph for the next variable, 6 parts of the code need to be changed.

This is very tedious!

How to automate? Change the layout of the data...

Gather the table into a long format with separate columns for the variable, mean and sd in order to automate making the graphs.

```
Chrono_graph <- Chrono_mean %>%
  # gather all data - key = variable (Ctot.pc, N.pc,...) _ statistic (mean/sd), value = "Value"
  gather("Variable_stat", "Value", -c(Age, Land.use)) %>%
  # separate the Variable_stat column to have separate columns for variable and statistic
  # e.g. C.tot and mean
  separate(Variable_stat, c("Variable", "stat"), sep = "_", remove = T) %>%
  # spread out the stat column so that the mean and sd have their own column
  spread(stat, Value)

head(Chrono_graph)
```

```
## # A tibble: 6 x 5
## # Groups:   Age [2]
##     Age Land.use Variable   mean      sd
##   <int> <fct>    <chr>     <dbl>   <dbl>
## 1     1 Arable   BAS.O2   0.664  0.267
## 2     1 Arable   Cmic     0.190  0.124
## 3     1 Arable   Ctot.pc  1.34   0.0539
## 4     1 Arable   N.pc     0.0227 0.00235
## 5     1 Arable   Nmic     0.0102 0.0113
## 6     2 Arable   BAS.O2   1.30   0.403
```

Code to produce automated graphs:

```
# create a vector for axis labels for your graphs including units
# this must be in the same order as your vars vector!
# the expression() function allows for special characters, sub- and superscript

units <- expression("Total carbon [%]",
                    "Total nitrogen [%]",
                    "Cmic [mg g soil"^-1*"]",
                    "Nmic [mg g soil"^-1*"]",
                    "Basal respiration [O"[2]*" "*mu*"l hr"^-1* "g soil"^-1*"]")

# the mean with errorbars is always plotted against soil age,
# the variable changes for each graph

graph_fun <- function(x) {
  Chrono_graph %>%
    # filter for the variable using the vars vector with placeholder x
    filter(Variable == vars[x]) %>%
    ggplot() +
    # x = Age, y = mean, different colour for land use
    aes(Age, mean, col = Land.use) +
    geom_point() +
    geom_line() +
    # plot error bars using the mean and sd columns
    geom_errorbar(aes(ymin = mean - sd, ymax = mean + sd)) +
    # give y axis label using units vector with placeholder x
    ylab( units[x] ) +
    xlab("Soil age [Years]") +
    labs(col = "Land use") +
    theme_bw()
```

```r
}

map(1:5, graph_fun)
```

Voila!

You could also think about making a vector for the plot title.