Table of Contents

Introduction **1**

1. Reading **2**

2. Processing **3**

2.1. Combining 3

2.2 Subsetting 3

3. Plotting **4**

3.1 Line plot 4

3.2 Scatter plot 4

3.3 histogram 4

3.4 Panels 4

4. Analysis **5**

4.1. Summary stats 5

4.2 Linear models 5

A final note on packages 5

Introduction

In this session we will outline a basic environmental data work-flow. Our goal is to highlight common data tasks, and typical ways to solve them in R.

When working with environmental data, there are usually a few steps that come up each time. These are:

- **reading**. Typically data is read from text files, but can also come the internet as highlighted in our previous session R2-API.ipynb
- **processing**. The data we read is usually a little untidy , for example we may need to subset to correct dates.
- **plotting**. Plotting data is always worth doing as early as possible. Use histograms or simple line plots as your first steps in visualising data.
- **analysis**. This step could include performing a statistical analysis or fitting a model.

To do these efficiently in R is mainly about learning which functions to use, and how to apply these functions.

In this notebook we will work through each step in turn with example data. We will once again work with data downloaded from SMARTSMEAR, in this case we will use flux data measured using the eddy co-variance technique at SMEARII research station. We will use Gross Primary Production (GPP) which is derived from measurements of CO2 exchange, and Evapotranspiration (ET) which is derived from measurements of H2O exchange.

Before we start there is one other thing we should mention.In this session we will assume that terms like *function*, *argument* are familiar to you. If they are not then go back to R1-introduction.ipynb, and check the definition. If you cannot find the definition in there then complain to your instructors to update the intro! Alright, let's get started.

1. Reading

Our first task is to read in our GPP and ET data. Reading data takes data from storage (typically your computer's hard disk) and places it somewhere (in RAM) that is can be operated on by R. We have already downloaded our data as two seperate text files from SMARTSMEAR, and stored these files in the */data* directory (folder) on github: https://github.com/OptPhotLab/EnvDataSciNotebooks/tree/master/data (You can inspect the data files by clicking the github link, but opening the individual files on github could slow your computer down!)

There are a few different functions for reading data in R, these include:

- read.csv
- read.table
- read.delim
- read.csv2

We can use **help** to inspect these functions, see what arguments they have, and how to set these arguments so that you can read your data/file in a proper way.

```
#help(read.csv)
```

Let's use *read.csv* to read in our GPP dataset.

```
gpp<-read.csv('../data/gppsmeardata_20160101120000.csv',header = T,sep = ',',dec='.')
```

The double dots **..** in the path tell R to go up a level in the directory (folder) hierarchy. The full path (location) of the ET data is:

../data/ET smeardata_20160101120000.csv

go ahead and read the ET data:

```
# name the output data ET
```

It is as simple as that!

We have read our data into the memory, the next step is processing. But just before we move on we can use the *head* function to inspect the first few lines of our data object:

```
head(ET)
```

```
## Error in head(ET): object 'ET' not found
```

can you also remember how to check the type of our objects?

2. Processing

Before we can make any graphs or perform any stats we usually have to tidy our data and there are a bunch of techniques in R that can help out with this. Let's check out a few of them that make life easier.

2.1. Combining

We read in two different data files. We can make life easier by combining these into a single dataframe.

Use the *by* argument to set which variables are shared.

```
gpp.ET<-merge(gpp,ET,by=c("Year","Month","Day","Hour","Minute", "Second"),all = T)
```

```
## Error in as.data.frame(y): object 'ET' not found
```

Use *head* to check the combination worked:

2.2 Subsetting

Often we download much more data than we need. Subsetting using the *subset* function is a useway to restrict our datasets to the bits we are actually interested in.

subset accepts column names as a second argument. You can use subset to extract data for the month of September from *gpp.ET* like this:

```
gpp.ET.sep <- subset(gpp.ET, Month==9)
```

```
## Error in subset(gpp.ET, Month == 9): object 'gpp.ET' not found
```

Can you create a new dataframe containing data measured at midday only?

Name this dataframe *gpp.ET.midday*

Use *head* to check the dates are correct:

Did you notice something odd? The days are not in ascending order. We can sort this out using the following (rather complicated!) line:

```
gpp.ET.midday <- gpp.ET.midday[with(gpp.ET.midday, order(Year, Month, Day)), ]
```

```
## Error in eval(expr, envir, enclos): object 'gpp.ET.midday' not found
```

Let's check this has worked out as expected:

BTW my solution to sorting was thanks to Google! You can check out a discussion of the various sorting options here: https://stackoverflow.com/questions/1296646/how-to-sort-a-dataframe-by-multiple-columns

Now we have a single dataframe with data at our desired midday time-step we can start with our visualisations.

3. Plotting

3.1 Line plot

The simplest plot of them all is the dot (or line) plot. The *plot* command is your friend here!

Let's see what our GPP data looks like:

```
plot(gpp.ET.midday$HYY_EDDY233.GPP)
```

```
## Error in plot(gpp.ET.midday$HYY_EDDY233.GPP): object 'gpp.ET.midday' not found
```

3.2 Scatter plot

We can also use *plot* to plot the relationship between variables by making scatter plots. Use the **~** operator to achieve this e.g. *plot(A~B,data=data.AB)*, where *A* and *B* are our variables and *data.AB* is our dataframe that contains our variables.

Try to make a scatter plot between GPP and ET for our midday data:

3.3 histogram

Checking the distribution of your data is usually a very good idea! **hist** is used to draw histograms. How is our midday GPP distributed?

3.4 Panels

Subplots (multiple plots in the same window) in R are achieved with the panels or *par* command. Specify the number of rows and columns as a two element vector and pass it using the *mfrow* key word as an argument to *par* e.g. par(mfrow =c(num.row,num.col)), then use repeated calls to *plot* in the usual way.

Can you complete the box below to draw ET and GPP in the same window but as separate subplots?

```
# first swap num.row and num.col for integers *par(mfrow =c(num.row,num.col))*
# then call plot() for each plot instance
```

4. Analysis

Our final step is to perform some simple analysis on our data. And because R is one of the languages of choice for stats,the possibilities for analyses really are nearly limitless!

Let's start with some simple summary statistics.

4.1. Summary stats

Statistics are at the heart of R, so let's use some! We can use the *mean* function on individual columns. We can use *sapply* with *mean* to work out the mean values for each column:

```
col.means <- sapply(gpp.ET.midday, mean, na.rm=TRUE)
```

```
## Error in lapply(X = X, FUN = FUN, ...): object 'gpp.ET.midday' not found
```

```
print(col.means)
```

```
## Error in print(col.means): object 'col.means' not found
```

The *summary* function applies a number of stats over each column. What do we get back when we try out summary on our midday data?

4.2 Linear models

Fitting models is a very common thing in environmental science, and the straight line is the most common of them all! To fit a line in R we use linear model *lm* function:

Let's model the relationship between ET and GPP in our midday data:

```
model.1<-lm(HYY_EDDY233.GPP~HYY_EDDY233.ET_gapf, data=gpp.ET.midday)
```

```
## Error in is.data.frame(data): object 'gpp.ET.midday' not found
```

summary also works on linear model results, try it below:

The *abline* function can be used to plot linear models over scatter plots. To try it out, you will need to enter the scatter plot code from section 2 and an abline function call.

A final note on packages

In this notebook we relied solely on built-in functionality, or in other words we did not use any R *packages*. But back in the real world of running scripts on our own machines we should make full use of external packages. In fact I would go as far to say that you should **always** check out what others have done and released before writing any code yourself. This could save you re-inventing the wheel!

We will cover some of the most popular packages in upcoming sessions but for a now a small intro on the concept for those of you who are unfamiliar:

A great deal of useful functionality in R is found in external *packages*. These are basically collections of code (functions) written by someone else, and kindly released for our use. Because these are external to our computer and hosted online, they require installation (downloading + building in correct location).

When running notebooks in class packages are installed ahead of time, so the actual installation is hidden from view from the user (you).

However when you are writing scripts to solve your own problems you may need to install these yourself. For example to install the package *ggplot2* which can be used for making publication quality plots, you would type the command *install.packages('ggplot2')*. This command then downloads the ggplot2 code to your machine, in a location specified by R.

Some packages are also hosted on github, for example you can browse the *ggplot2* source code before you install here:

https://github.com/tidyverse/ggplot2

Thinking to the future, could you imagine your own code being released as a package? What would be the benefits of this?