Introduction to R and RStudio

(ACSPRI Summer Program 2017, 6-10 February 2017)

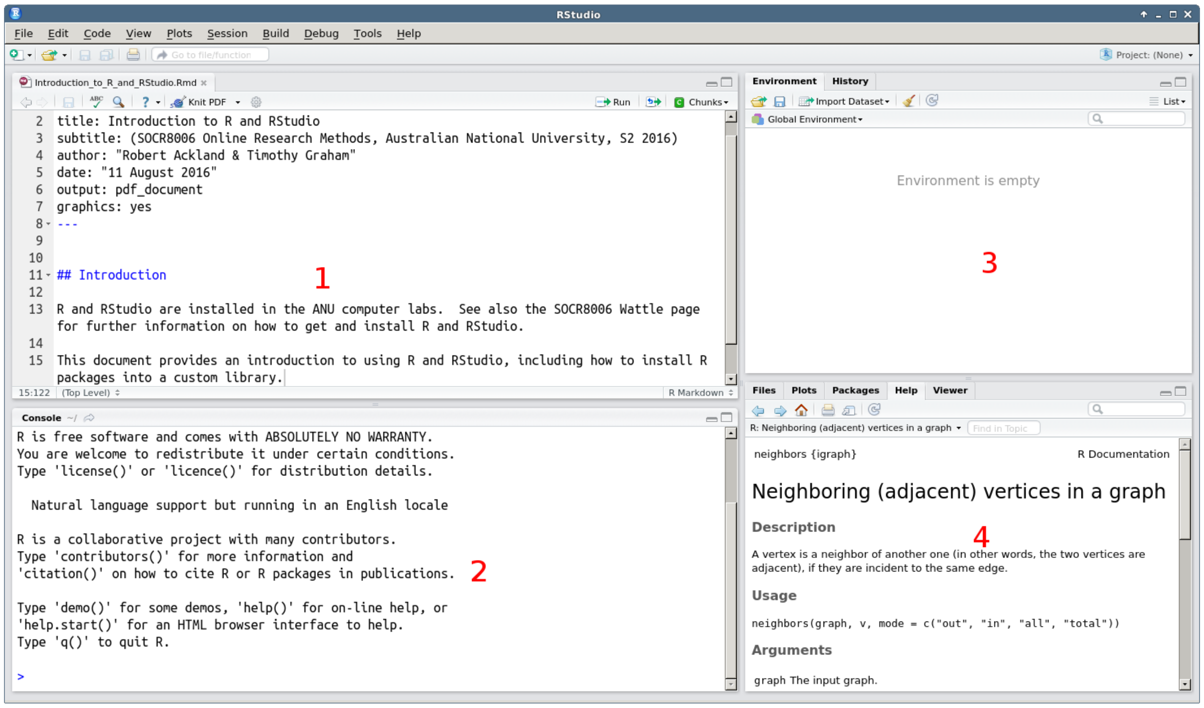
Robert Ackland & Timothy Graham

4 February 2017

Introduction

This document provides an introduction to using R and RStudio, including how to install R packages into a custom library. RStudio provides a nice user interface to R, and we will use it in our classes. See the course page for further information on how to get and install R and RStudio, if you want to install them on your own computers.

When you open up RStudio it will look something like the following:


RStudio
File Edit Code View Plots Session Build Debug Tools Help
Go to file/function
Project: (None)
Introduction_to_R_and_RStudio.Rmd
Knit PDF
Run
Chunks
Environment History
Import Dataset
List
Global Environment
Environment is empty
Files Plots Packages Help Viewer
R: Neighboring (adjacent) vertices in a graph
Find in Topic
Console ~/
15:122 (Top Level) R Markdown
1
2
3
4

```
title: Introduction to R and RStudio
subtitle: (SOCR8006 Online Research Methods, Australian National University, S2 2016)
author: "Robert Ackland & Timothy Graham"
date: "11 August 2016"
output: pdf_document
graphics: yes
---


## Introduction

R and RStudio are installed in the ANU computer labs.  See also the SOCR8006 Wattle page
for further information on how to get and install R and RStudio.

This document provides an introduction to using R and RStudio, including how to install R
packages into a custom library.
```

```
R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

  Natural language support but running in an English locale

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

>
```

neighbors {igraph} R Documentation

Neighboring (adjacent) vertices in a graph

Description

A vertex is a neighbor of another one (in other words, the two vertices are adjacent), if they are incident to the same edge.

Usage

```
neighbors(graph, v, mode = c("out", "in", "all", "total"))
```

Arguments

graph The input graph.

There are the following windows/panes:

1. **Source** This is where you work with R scripts and R markdown files (explained below). You can run a single line in an R script by pressing the "Run" button in the top left corner, or else you can run groups of lines by highlighting them and pressing "Run" (note: you need to highlight the entire line!)
2. **Console** This is the R console (similar to if you were running R by itself, not within RStudio). As you run lines in the source pane, the results will show in the console pane. You can also run commands directly in the console pane.
3. **Environment etc.** This is where you can see what variables/objects have been loaded into the environment.
4. **Files etc.** This is for: the file system, plots, package management, and help.

We will mainly work with the source and console panes. Note the following useful shortcuts: ctrl+1 will take you to the source pane and ctrl+2 will take you to the console.

R source files and R markdown files

I will be providing you with both R source files (*.R) and R markdown files (*.Rmd). An R markdown file is a special R script that has additional commands that allow you to produce PDF documents that incorporate R syntax and the outputs of running the syntax. When I provide an R markdown file, I will generally also provide the PDF output because it is generally easier to read than the associated Rmd file but note: you load the Rmd file into the source pane.

One word of caution about using PDF files created from Rmd files: while you can copy and paste the R syntax from the PDF, be aware that sometimes long lines of R syntax will run off the page, and so copying and pasting such lines into the R console will cause an error.

Finally, you will see in the Rmd files R syntax that is enclosed by three backticks and "{r eval=TRUE}" and another link with three backticks - make sure you only run the code that is enclosed by these two lines.

A very brief introduction to R

There are many resources on the web for learning R, and I've included links to some on the course page. The following is a very brief introduction to using R. As the course progresses I'll introduce other syntax as required.

R syntax (like most computer languages) is case sensitive!

Generally speaking:

1. Everything in R is an object
2. Everything that happens is a function call

R data structures are extremely important to understand, so lets first have a look at some of the major data structures.

Vectors

Probably the most important/common data structure in R is the **vector**.

A vector can be a vector of elements that are most commonly character, logical, integer or numeric.

```r
# a simple numeric vector (containing one element)
x <- 5
x
```

```
## [1] 5
```

```r
# add x to itself
x <- x + x
x
```

```
## [1] 10
```

```r
# Assigning a character vector
myName <- "John"
myName
```

```
## [1] "John"
```

```r
# look at attributes of vectors, e.g. myName
nchar(myName)
```

```
## [1] 4
```

```r
# Assigning a logical vector
skyIsBlue <- TRUE
skyIsBlue
```

```
## [1] TRUE
```

```r
isTRUE(skyIsBlue)
```

```
## [1] TRUE
```

```r
# Numeric vector (e.g. 5, 19.33, 17.2)
# We can create a sequence of numbers using:
countToTen <- c(1:10)
countToTen
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10
```

```r
# get the length attribute of this vector:
length(countToTen)
```

```
## [1] 10
```

```r
# access the fifth element of this vector
countToTen[5]
```

```
## [1] 5
```

```r
# access the first 3 elements of this vector
countToTen[1:3]
```

```
## [1] 1 2 3
```

```r
# examine this vector
typeof(countToTen)
```

```
## [1] "integer"
```

```r
str(countToTen)
```

```
##  int [1:10] 1 2 3 4 5 6 7 8 9 10
```

```r
# examine the myName vector
str(myName)
```

```
##  chr "John"
```

```r
# Mixing data types in vectors is not possible and R will coerce your data to the lowest common denomin
# e.g. integers and characters will coerce to 'character'
mixedVector <- c(1,"a",2,"b")
str(mixedVector)
```

```
##  chr [1:4] "1" "a" "2" "b"
```

Matrices

Matrices are special vectors in R.

A maxtrix is basically a multi-dimensional atomic vector, i.e. with rows and columns

```r
# Matrices are filled column-wise, for example:
myMatrix <- matrix(1:6, nrow = 2, ncol = 3)
myMatrix
```

```
##      [,1] [,2] [,3]
## [1,]    1    3    5
## [2,]    2    4    6
```

```r
# You can make a matrix out of two vector objects, for example:
vector1 <- 1:5
vector2 <- 6:10
myMatrix2 <- cbind(vector1,vector2)

# access the column names of myMatrix2:
colnames(myMatrix2)
```

```
## [1] "vector1" "vector2"
```

```r
# access the element in the first row and second column
myMatrix2[1,2]
```

```
## vector2
##       6
```

```r
# access the first 3 rows in the second column
myMatrix2[1:3,2]
```

```
## [1] 6 7 8
```

```r
# access all of the second column
myMatrix2[,2]
```

```
## [1]  6  7  8  9 10
```

Lists

Lists in R act like 'containers'

```r
# They are different to vectors because each element can be different type, e.g.
myList <- list("Hello", 1, TRUE, "Goodbye")
myList
```

```
## [[1]]
## [1] "Hello"
##
## [[2]]
## [1] 1
##
## [[3]]
## [1] TRUE
##
## [[4]]
## [1] "Goodbye"
```

```r
# Access elements in lists slightly differently:
myList[1][[1]]
```

```
## [1] "Hello"
```

```r
# what is the class of the first element of the first list element?
class(myList[1][[1]])
```

```
## [1] "character"
```

```r
# and what about the first element of the second list element
class(myList[2][[1]])
```

```
## [1] "numeric"
```

Factors

Factors are special vectors that represent categorical data.

```r
myFactor <- factor(c("yes", "no", "no", "yes", "yes"))
myFactor
```

```
## [1] yes no  no  yes yes
## Levels: no yes
```

```
# Operate on the factor to find which elements equal "yes"
which(myFactor=="yes")
```

```
## [1] 1 4 5
```

Data frames

Data frames are very important in R, and we will use them a lot in this course.

```
# each column in the data frame can have a different data type, for example:
df <- data.frame(names = c("John","Jane","Sally"), testScores=c(99,84,30), failingGrade=c(FALSE, FALSE,
df
```

```
##   names testScores failingGrade
## 1  John         99        FALSE
## 2  Jane         84        FALSE
## 3 Sally         30         TRUE
```

```
# we can access the third column in two ways:
df[,3]
```

```
## [1] FALSE FALSE  TRUE
```

```
df$failingGrade
```

```
## [1] FALSE FALSE  TRUE
```

```
# number of rows in data frame
nrow(df)
```

```
## [1] 3
```

```
# find out the structure of each column
str(df)
```

```
## 'data.frame':    3 obs. of  3 variables:
##  $ names       : Factor w/ 3 levels "Jane","John",..: 2 1 3
##  $ testScores  : num  99 84 30
##  $ failingGrade: logi  FALSE FALSE TRUE
```

Naming objects

```
x <- 1:3
names(x) <- c("Charlie", "Echo", "Foxtrot")
x
```

```
## Charlie    Echo Foxtrot
##       1       2       3
```

Missing values

```
# denoted by NA or NaN (for undefined mathematical operations)
x <- c(1,2,NA,4,5)
x
```

```
## [1]  1  2 NA  4  5
```

```
is.na(x)
```

```
## [1] FALSE FALSE  TRUE FALSE FALSE
```

Functions in R

Functions objects that take some input and then return some output

```
myFunction <- function(x,y) {
  result <- x + y
}

output <- myFunction(4,5)
output
```

```
## [1] 9
```

```
someNumber <- 5
anotherNumber <- pi * 10 ^ 2
output <- myFunction(someNumber,anotherNumber)
output
```

```
## [1] 319.1593
```

Reading and writing files from R

```
df <- data.frame(names = c("John","Jane","Sally"), testScores=c(99,84,30), failingGrade=c(FALSE, FALSE,
write.csv(df,"dataframe.csv")
dataIn <- read.csv("dataframe.csv")
dataIn
```

```
##   X names testScores failingGrade
## 1 1  John         99        FALSE
## 2 2  Jane         84        FALSE
## 3 3 Sally         30         TRUE
```

Setting the working directory

The working directory is where R will read data files from and write files to.

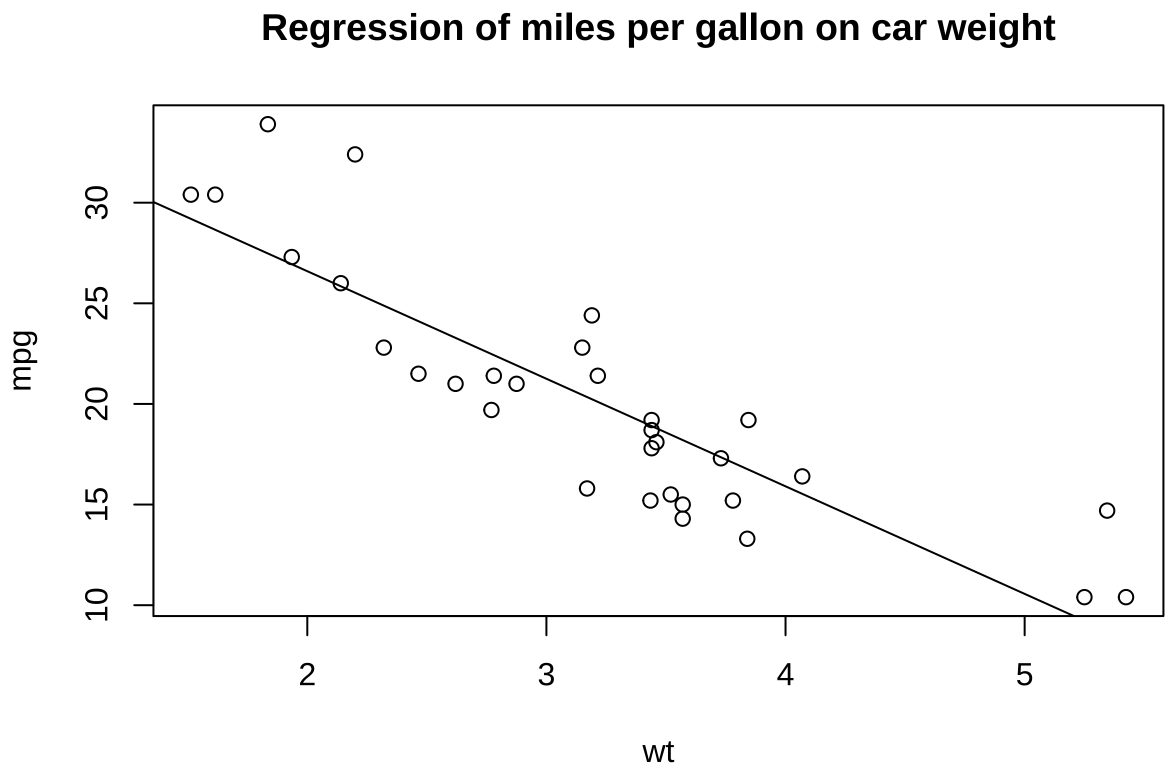
It is generally useful to set the working directory to the same directory where you are storing your R syntax files. This can be done in RStudio with the following menu selection: Session->Set Working Directory->To Source File Location

```r
# this is equivalent to the following R command (in Linux...Windows is different):
setwd("~/my_working_directory")
# to find your current working directory:
getwd()
```

Creating a plot or graph

R has very advanced plotting capabilities. The following is a simple example.

```r
attach(mtcars)
plot(wt,mpg)
abline(lm(mpg~wt))
title("Regression of miles per gallon on car weight")
```



The above code will create the plot in the "Plots" tab in the **Files etc.** pane.

It is often useful to open up an X window to view the plot.

```r
# this opens a separate graphics window outside of RStudio
x11()
# the following closes the window
dev.off()
```

```
## pdf
##   2
```

To print the plot to a bitmap ('png') use the following:

```
png("plot_mpg.png")
plot(wt,mpg)
abline(lm(mpg~wt))
title("Regression of miles per gallon on car weight")
dev.off()
```

Getting help in R

R has an excellent help file system. You can find help for a particular command (for example, 'abline') by typing the following in the console: ?abline

In RStudio, this will open up the relevant help file in the **Files etc.** pane.

Installing R packages

We will need to install the additional R packages in a custom library: we will use the default location/folder on the network drive (we discuss this further in class). Note that we probably don't have administrative rights to install packages onto the computer hard drive, so we need to install to the network drive.

```
# the following command prints out the directories in the library path (this will be different when run
.libPaths()
# the following adds a new directory to the library path (you shouldn't need to do this)
# if you do use this, you will need to put this command at the top of your R syntax files, so R knows w
# to load custom packages from
.libPaths("/path/to/directory")
```

The following will install a new package "igraph" into your custom library:

```
install.pacakges("igraph")
```

To use the package, you first need to load it into the session using:

```
library(igraph)
```