COMP 120: Lab 1 Worksheet[[1]](#footnote-1)

Welcome to the first lab in COMP 120. The goal of the first two labs in COMP 120 is to become comfortable with R and RStudio, particularly around building simple scripts and performing basic computations. In later labs we’ll move on to data science tasks such as reading and writing data from files, manipulating data, creating plots, and communicating results.

The majority of this document is for review purposes, but anything that you see in a monospace font can (and should!) be executed in RStudio as you’re reading the document. The commands you need to execute are preceded by the “>” symbol. In addition, we have a set of exercises that you are required to complete as part of this lab – these serve to demonstrate that you have the basics of the lab covered, and so you are ready to continue with work in future labs.

# Introduction

R is a powerful language and environment for statistical computing and graphics. It is a public domain (a so called “GNU”) project which is similar to the commercial S language and environment which was developed at Bell Laboratories (formerly AT&T, now Lucent Technologies) by John Chambers and colleagues. R can be considered as a different implementation of S, and is much used in as an educational language and research tool. The main advantages of R are the fact that R is free (“open source”) software and that there is a lot of help available online. It has many similarities to other programming packages such as MatLab (not free) but is more user-friendly than programming languages such as C++ or Fortran. You can use R as it is, but for educational purposes we prefer to use R in combination with the RStudio interface (also free software), which has an organized layout and several extra options.

This document contains explanations, examples and exercises, which should be understandable by people without any programming experience. Going through all text and exercises takes about 1 to 2 hours. By the end of this process, you will have experienced the main aspects of R and RStudio.

# Getting Started with R and RStudio

## Install R (**only do this on your own machine – skip if are using Student Desktop**)

To install R on your computer (legally for free!), go to the home website of R:

<http://www.r-project.org/>

and do the following (assuming you work on a windows computer):

* click download CRAN in the left bar
* choose a download site
* choose Windows as your target operating system from the options (Linux, Mac OS, Windows)
* click base
* choose Download R 4.0.2 for Windows, run the download and choose default answers for all questions (Note: The version installed on lab machines is 3.6.2).

Note: a similar process is done for Mac OS (download the latest pkg option and follow the instructions), while the major flavours of Linux each provide R through their package manager, so you should use that process to obtain R.

## Install RStudio (**only do this on your own machine – skip if using the Student Desktop**)

After finishing this setup, you should install RStudio, as it provides a more consistent user interface across all platforms. To install RStudio, go to:

<https://rstudio.com/products/>

and do the following (assuming you work on a windows computer, a similar process exists for all the other platforms):

* click on R Studio (The premier IDE for R) under Open Source Products
* click the RStudio Desktop option which takes you to the download option
* click Download RStudio Desktop button
* choose the free download option
* download the .exe file and run it (choose default answers for all questions)

## RStudio Layout

Once you have R and RStudio installed, you can start RStudio through the normal process on your machine (e.g., through the Start menu in Windows). The RStudio layout was discussed in Lecture 2 and is consistent across all platforms. To recap, the layout is:

* Bottom left: console window (also called command window). Here you can type simple commands after the “>” prompt and R will then execute your command. This is the most important window, because this is where R actually does stuff.
* Top left: editor window (also called script window). Collections of commands (scripts) can be edited and saved. When you don’t get this window, you can open it with File → New → R script. Just typing a command in the editor window is not enough, it has to get into the command window before R executes the command. If you want to run a line from the script window (or selected code), you can click Run or press CTRL+ENTER to send it to the command window.
* Top right: environment / history window. In the environment window you can see which data and values R has in its memory. The history window shows what has been typed before.
* Bottom right: files / plots / packages / help window. Here you can open files, view plots (also previous plots), install and load packages or use the help function.

You can change the size of the windows by dragging the bars between the windows.

## Setting up a folder for the course

All your work for this course should be stored in the **H: drive**. H: drive is identified by your student username. This is your personal virtual drive where you do all your work. Create a new folder at the root of the H: drive called COMP120 (no spaces in the name). It is probably a good idea to do the same for your other courses. This will make navigating and finding your files easier. Inside COMP120 folder create a folder called labs. Inside that folder create another folder called lab01. For each lab you are expected to create appropriate folders (e.g. lab02, lab03). In the appropriate lab folder (e.g. lab01) you should save the files corresponding to that lab. These include the lab description document (usually in the .docx format), data files (e.g. csv files) and R files that we may provide for mastery tasks that you need to complete.

## Working Directory

Your working directory is the folder on your computer in which you are currently working. When you ask R to open a certain file, it will look in the working directory for this file, and when you tell R to save a data file or figure, it will save it in the working directory.

Before you start working, please set your working directory to where all your data and script files are or should be stored. Type in the command window: setwd("directoryname"). For example:

> setwd("H:/COMP120/labs/lab01")

Make sure that the slashes are forward slashes and that you don't forget the quotes. R is case sensitive, so make sure you write capitals where necessary to match the actual directory names. Note that the appropriate directory must exist before you can set the working directory.

Within RStudio you can also go to *Session* tab and then choose an appropriate option from *Set Working Directory* menu option to set the working directory through RStudio's interface.

You can check the working directory by the following command:

> getwd()

## Lib Paths (**only do this if you’re working on the Student Desktop**)

The way that R and RStudio are configured on the University Student Desktop prevents them from writing to certain parts of the C: drive (for obvious reasons). Therefore, we need to tell R to set up a directory that allows us to install extensions (e.g., new libraries). **First, we need a suitable directory: create a directory under the root of your H: drive with the name R**. Then, every time that you start RStudio, enter the following command into the console:

> .libPaths("H:/R")

And this will tell R to install new libraries and code into this directory (which is writeable).

## Libraries

R can do many statistical and data analyses. They are organized in so-called packages or libraries. With the standard installation, most common packages are installed. To get a list of all installed packages, go to the packages window or type library() in the console window. If the box in front of the package name is ticked (in the packages window), the package is loaded (activated) and can be used.

There are many more packages available on the R website. If you want to install and use a package (for example, the package called "geometry") you should:

1. Install the package: click install packages in the packages window and type geometry or type install.packages("geometry") in the command window.
2. Load the package: check box in front of geometry or type library("geometry") in the command window.

We'll start using some useful libraries later in the semester, so we'll revisit this concept later when we need to use it again. Many of the examples that you’ll encounter in lectures, labs, and assessments will make use of the tidyverse package which come preinstalled in the C: drive. You just need to load the package to use it (i.e., use the command in the second step above). However, if you have installed R in your own machine (laptop or desktop), you should proceed with installing that package. So, follow the two steps indicated above to install the tidyverse package.

# Some first examples of R commands

Right! We’re actually at the point where we can start using R (through RStudio). We’ll start with using R as a basic calculator

## R as a Calculator

R can be used as a calculator. You can just type your equation in the command window after the “>”:

> 10^2 + 36

and R will give the answer:

[1] 136

If you use brackets and forget to add the closing bracket, the “>” on the command line changes into a “+”. The “+” can also mean that R is still busy with some heavy computation. If you want R to quit what it was doing and give back the “>”, press ESC.

## Variables and Environments

You can also give numbers a name. By doing so, they become variables in your environment that can be used later. For example, you can type in the command window and press ENTER:

> a <- 4

As discussed in the lecture, what the above piece of code means is that you create a variable named a, and assign the value 4 to it. The assignment operator <- can be created using keyboard shortcut: by pressing ALT and – keys simultaneously. Note there shouldn’t be a space between < and -. If there is space you would get weird results. That is one of the reasons to remember the keyboard shortcut.

You can see that a appears in the environment window, which means that R now remembers what a is. You can also ask R what a is (just type a into the command window and then press ENTER):

> a

[1] 4

or do calculations with a:

> a \* 5

[1] 20

If you specify a again, it will forget what value it had before. You can also assign a new value to a using the old one:

> a <- a + 10

> a

[1] 14

To remove all variables from R’s memory, type:

> rm(list=ls())

or click “clear all” in the environment window. You can see that RStudio then empties the environment window. If you only want to remove just the variable a, you can type rm(a).

## Data types

So far, we have explored examples of typical commands that you would use to interact with the R environment – these examples have all dealt with numeric data. While numeric data is by far the most common type of data that you will deal with, there are many other types of data that you will encounter. R has a number of built-in types for handling most of the common examples of data that you will need to manage and these will be covered in the upcoming lectures:

* Numeric – basically anything that can have a mathematical operation applied to it (e.g., addition, multiplication, logarithm, etc.) – numeric data can be whole numbers (integers) or real numbers (with a fractional part).
* Logical – data that is Boolean (TRUE or FALSE) in nature. You manipulate this data using logical operators (e.g., AND, OR, NOT)
* Textual – data that is character-based (in some programming languages, these would be called strings).
* Ordered/categorical – labels that represent (possibly arbitrary) groupings. When these groupings have no specific ordering (e.g., gender), these are simply called factors. When these groupings have a logical ordering (e.g., age group) they are called ordered factors.

We will discuss converting between these types of data at a later date in the semester (Week 5).

## Scalars, vectors and matrices

Like in many other programs, R organizes values into scalars[[2]](#footnote-2) (a single number), vectors (a sequence of numbers, also called arrays – 1-dimensional) and matrices (like a table – 2- dimensional). The variable a you defined before was a scalar. To define a vector with the numbers 3, 4 and 5, you need the function c(), which is short for concatenate (paste together):

> b <- c(3,4,5)

There is one special case of the vector that will be of use to you in many situations – the sequence. This is simply a vector in which the values within are in ascending or descending order, with uniform spacing between the values. There are two main methods for generating sequences. The most flexible approach is the seq(from, to, by) function, which will (intuitively) generate a sequence starting at from and ending at to, with a gap of by between each subsequent value. An example of using seq() would be:

> seq(0, 2, 0.25)

[1] 0.00 0.25 0.50 0.75 1.00 1.25 1.50 1.75 2.00

Another way of generating sequences, useful when the sequence contains whole numbers with no gaps in the intervals, is the colon (:) operator. For example, we can generate a sequence of the first 10 digits starting at zero using:

> 0:9

[1] 0 1 2 3 4 5 6 7 8 9

We’ll look at vector indexing (i.e., retrieving and updating specific values in the vector) in the next lecture and lab.

## Functions

If you would like to compute the mean of all the elements in the vector b from first example in the previous section, you could type:

> (3+4+5)/3

But when the vector is very long, this is very boring and time-consuming work. This is why things you do often are automated in so-called functions. Some functions are standard in R or in one of the packages. You can also program your own functions (see the next lab). When you use a function to compute a mean, you’ll type:

> mean(x=b)

Within the brackets you specify the arguments. Arguments give extra information to the function: in this case, the argument x says of which set of numbers (vector) the mean should computed (namely of b). Sometimes, the name of the argument is not necessary: mean(b) works as well.

The function rnorm, as another example, is a standard R function which creates random samples from a normal distribution. Hit the ENTER key and you will see 10 random numbers as:

> rnorm(10)

[1] -0.949 1.342 -0.474 0.403

[5] -0.091 -0.379 1.015 0.740

[9] -0.639 0.950

* Line 1 contains the command: rnorm is the function and the 10 is an argument specifying how many random numbers you want — in this case 10 numbers (typing n=10 instead of just 10 would also work).
* Lines 2-4 contain the results: 10 random numbers organised in a vector with length 10.

Entering the same command again produces 10 new random numbers. Instead of typing the same text again, you can also press the upward arrow key (↑) to access previous commands. If you want 10 random numbers out of normal distribution with mean 1.2 and standard deviation 3.4 you can type

> rnorm(10, mean=1.2, sd=3.4)

showing that the same function (rnorm) may have different interfaces and that R has so called named arguments (in this case mean and sd). By the way, the spaces around the “,” and “=” do not matter. Comparing this example to the previous one also shows that for the function rnorm only the first argument (the number 10) is compulsory, and that R gives default values to the other so-called optional arguments. In the previous example, the mean and sd were set to default values which are 0 and 1 respectively.

RStudio has a nice feature of tab completion for commands: when you type rnorm( in the command window and press TAB, RStudio will show the possible arguments.

# Scripts

R is an interpreter that uses a command line based (or REPL - Read, Execute, Print, Loop) environment. This means that you have to type commands, rather than use the mouse and menus. However, issuing commands repetitively can become tedious. Instead, you can store your commands in files, the so-called *scripts*. These scripts typically have file names with the extension .R, e.g. foo.R. You can open an editor window to edit these files by clicking File and New File or Open File…. Once the file is open, the editor window will appear in RStudio, and you can start to record your commands in this file.

As with all scripts, writing the commands in the file does not immediately issue them into the environment. You can run (send to the console window) part of the code by selecting lines and pressing CTRL+ENTER or click Run in the editor window. If you do not select anything, R will run the line your cursor is on. You can always run the whole script with the console command source, so e.g., for the script in the file foo.R you type:

> source("foo.R")

You can also click Run all in the editor window or type CTRL+SHIFT + ENTER to run the whole script at once.

## Comments in Scripts

Scripts in R can become quite long, and may contain several commands that, upon revisiting at a later date, seem confusing or arbitrary. It is useful to add notes or “comments” in your script to remind you of what particular parts of your script are doing. Adding comments is very easy, simply start the comment with a hash symbol (#) and everything on the rest of the line is not executed. For example, you might want to include a comment in your code to explain why a variable has been set to a specific value:

x <- 42 # the answer to the Ultimate Question of Life,

# the Universe, and Everything

You are encouraged to make extensive use of comments in your scripts – something that you write today that looks entirely obvious may become quite obtuse at a later date, so spending the few seconds required to write a comment is well worth the effort.

# Final Tasks for Today

1. Create a new script called firstscript.R inside the lab01 folder. Write code to perform computations indicated in steps 2 and 3 below.
2. Perform calculation that computes the area of a square whose side is 4 units. Print the result of the computation.
3. Create a sequence with the following values using both the seq function and the colon operator (:). How are these two ways of creating values similar and how are they different?

22, 23, 24, 25, 26

1. What are the keyboard short cuts for the following in R-Studio? (Hint: see the section below)
   1. Assignment operator (<-)
   2. Executing one line or selected lines of code from the script window
   3. Executing the entire script from the script window

# Mastery Task 1

Now that you’ve been exposed to R and R-Studio environments, it’s time to use it to demonstrate that you understand the basics of R. Download the mastery-01.R and examplescript.R document from Blackboard into the lab01 folder, and complete the tasks specified in the document by writing the appropriate commands as indicated by the comments in the script. When you have completed the tasks, **submit your completed work on Blackboard before 4pm on Thursday the 16th of July**.

**Where to submit the mastery file?** When you complete the mastery, you must logon to Blackboard and navigate to the Mastery Tasks folder (*Course Documents -> Mastery Tasks*). Then, click on the link which says Mastery Task 1​. An easy way to identify the clickable link is through the underline that appears in the name of the folder (i.e., Mastery Task 1). When you click on this link you will be able to submit the file (by selecting the right file from the computer and then clicking the “Submit” button).

# How to connect to the lab environment from home/flat?

You can access the same student desktop that is available to you in the labs from anywhere in the world from any device as long as you are connected to the internet. The instructions for accessing the student desktop system can be found here: <https://blogs.otago.ac.nz/studentit/student-desktop/student-desktop-own-device/>. Option 2 (using the light version) is often the easiest. We strongly recommend that you test the usage of this system. This will be very handy when you work on your mastery tasks from home/flat.

# Miscellaneous 1 – Some useful keyboard shortcuts

Here are some useful keyboard shortcuts you might find handy when using R-Studio on Windows. For an exhaustive list of short cut keys press **Alt + Shift + K.**

* **Ctrl + 1**: Move focus to the Script Editor
* **Ctrl + 2**: Move focus to the Console (command window)
* **Ctrl + L**: Clear the Console
* **Esc**: Interrupt R
* **Alt +** - : Create the assignment operator (<-)
* **Ctrl + Enter**: Execute the current line of code or selected lines of code
* **Ctrl + Shift + Enter** — Execute the whole script

# Miscellaneous 2 – Do you know?

Answer the following questions about the course. If you don’t know the answers refer to the *course outline* or the first lecture slides.

1. How many mastery tasks do we have in this course? How much are the mastery tasks worth in total? Will each mastery submission be marked (and will all questions in each mastery be marked or just a subset)? How much is each marked submission worth?
2. How many tests do we have in this course? How much is each test worth?
3. You may know that you need to score at least 50% to pass this paper. There are also additional hurdles. What are those hurdles?
4. How many lecturers are involved in this course? How can you contact the course coordinator?

1. A lot(!) of the text and examples for these labs is taken from “A (very) short introduction to R” by Paul Torfs & Claudia Brauer: <https://cran.r-project.org/doc/contrib/Torfs+Brauer-Short-R-Intro.pdf> [↑](#footnote-ref-1)
2. Scalars are in-fact vectors of length 1. [↑](#footnote-ref-2)