CBD2204: Week 3

T. Zourntos, PhD PEng

much of the following material is taken from…

the course notes for the “Introduction to R” course given by the Monash Bioinformatics Platform.

Some of this material is derived from work that is Copyright © Software Carpentry 1 with a CC BY 4.0 license 2 .

Data files we are working with are available from:

http://monashbioinformaticsplatform.github.io/r-intro/

working with *R* (in-depth)

R is both a programming language and interactive tool for mathematical analysis

It bears similarity to software like *Matlab* or *Mathematica*, which are intended for scientific and engineering applications. If you have experience with *python* within a *Jupyter* notebook environment, the *R Studio* will feel quite familiar.

Console

Let’s try some simple arithmetic operations from the console window:

working with *R* (cont’d)

try the following expressions, entered at the *R Studio* command prompt:

- 1 + 1
- 2*7==15
- 2*7==14
- 29**2
- x <- 55
- 65 -> y
- z = y - x

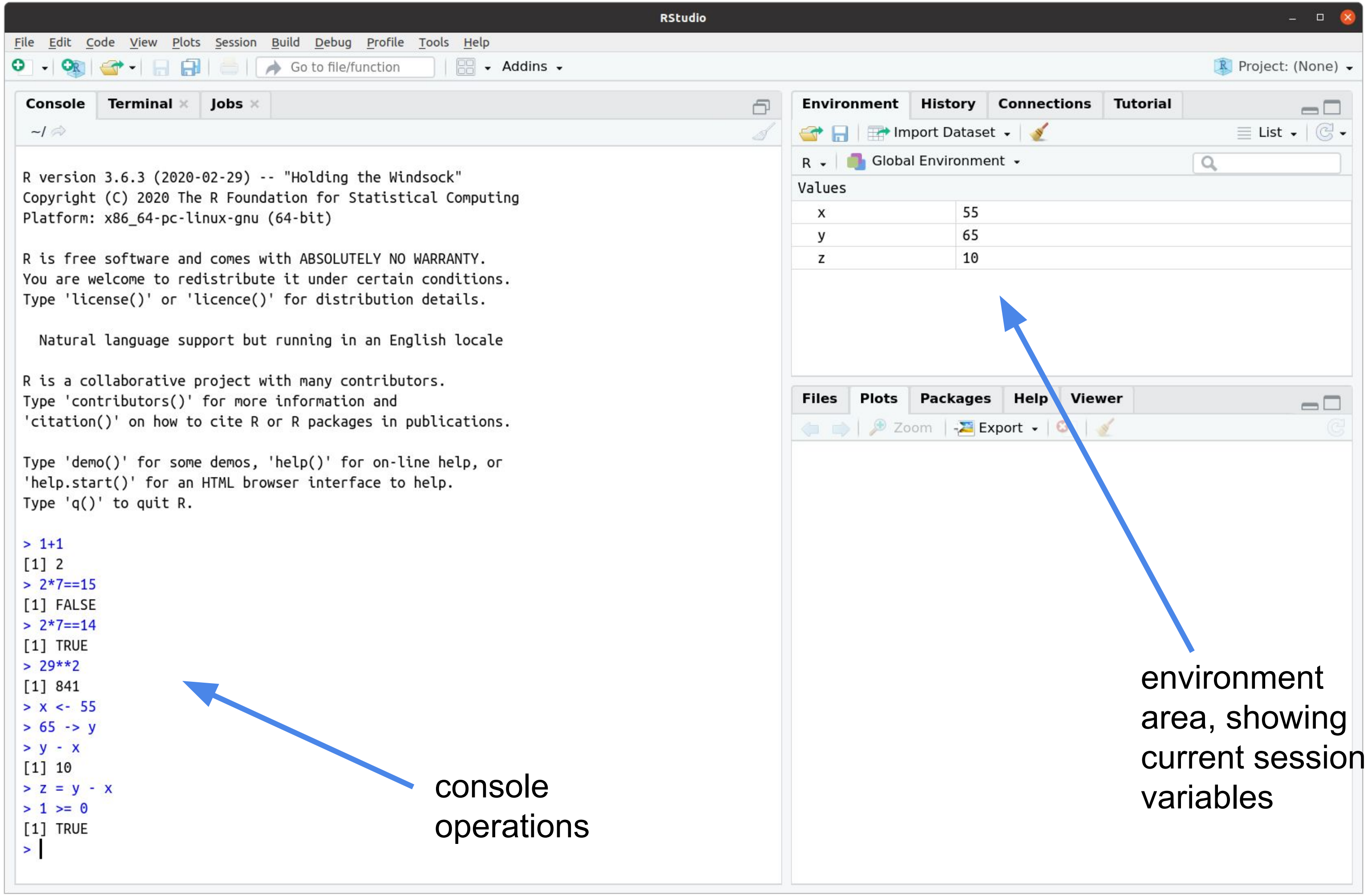
these examples illustrate that the conventional arithmetic operators + - * / as well as brackets ( ) if needed, work within R, following the usual order of operations; logical operators like ==, >=, <, etc., also work. Can you provide some additional examples to illustrate?

working with *R* (cont’d)

these examples also demonstrate the use of variables, and the assignment operators, -> and <-, which are equivalent (but work in opposite “directions”)

you can also view variables, their types and contents in the *Environment* tab

N.B.: “Examples of valid variables names: hello, hello_there, hello.there, value1. Spaces aren’t ok inside variable names. Dots (.) are ok, unlike in many other languages.”



```
R version 3.6.3 (2020-02-29) -- "Holding the Windsock"
Copyright (C) 2020 The R Foundation for Statistical Computing
Platform: x86_64-pc-linux-gnu (64-bit)

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.

> 1+1
[1] 2
> 2*7==15
[1] FALSE
> 2*7==14
[1] TRUE
> 29**2
[1] 841
> x <- 55
> 65 -> y
> y - x
[1] 10
> z = y - x
> 1 >= 0
[1] TRUE
> 
```

Values

x	55
y	65
z	10

console operations

environment area, showing current session variables

working with *R* (cont’d)

from the r-intro.pdf: “We can add comments to our code using the # character. It is useful to document our code in this way so that others (and us the next time we read it) have an easier time following what the code is doing.”

- please add comments to your code, because someone else will have to understand your logic;
    - most often, this “someone else” is you, at some point in the future, so:
        - *be kind to yourself!!*

vectors

```
> myvec <- c(10,20,30,40,50)

> myvec + 1

## [1] 11 21 31 41 51

> myvec + myvec

## [1]

20

40

60

80 100

> length(myvec)

## [1] 5

> c(60, myvec)

## [1] 60 10 20 30 40 50

> c(myvec, myvec)

## [1] 10 20 30 40 50 10 20 30 40 50
```

c( ) is used to *c*reate vectors

- "Access elements of a vector with [ ], for example myvec[1] to get the first element. You can also assign to a specific element of a vector."

functions

- R has various functions, such as sum( ). We can get help on a sum with ?sum.

```
> ?sum

> sum(myvec)

## [1] 135
```

- Here we have called the function sum with the argument myvec.
- Because *R* is a language for statistics, it has many built in statistics-related functions. We will also be loading more specialized functions from “libraries” (also known as “packages”). Some functions take more than one argument. Let’s look at the function *rep*( ), which means “repeat”, and which can take a variety of different arguments. In the simplest case, it takes a value and the number of times to repeat that value.

```
> rep(42, 10)

## [1] 42 42 42 42 42 42 42 42 42 42
```

lists

- Vectors contain all the same kind of thing. Lists can contain different kinds of thing. Lists can even contain vectors or other lists as elements.
- We generally give the things in a list names. Try:

```
> list(num=42, greeting="hello")
```

- To access named elements we use $:

```
> mylist <- list(num=42, greeting="Hello, world")
> mylist$greeting
## [1] "Hello, world"
```

- A final piece of special syntax for lists is [[ ]] to access individual elements by number. We won’t be using this today, but include it for completeness.

```
> mylist[[2]]

## [1] "Hello, world"
```

- Functions that need to return multiple outputs often do so as a list.

matrices

- A matrix is a two dimensional tabular data structure in which all the elements are the same type. We will typically be dealing with numeric matrices, but it is also possible to have character or logical matrices, etc. Matrix rows and columns may have names (rownames, colnames).
- Access an element: mat[3,5] mat["arowname","acolumnname"]
- Get a whole row: mat[3,]
- Get a whole column: mat[,5]
- Creation: matrix( )
- Casting: as.matrix( )
- ?matrix

data frames

- A data frame is a two dimensional tabular data structure in which the columns may have different types, but all the elements in each column must have the same type.
- Data frame rows and columns may have names (rownames, colnames). However in typical usage columns are named but rows are not.
- Accessing elements, rows, and columns is the same as for matrices, but we can also get a whole column using $.
- Creation: data.frame(colname1=values1,colname2=values2,...)
- Casting: as.data.frame( )
- ?data.frame

we’ll be using a set of R libraries known as *tidyverse*

```
install.packages("tidyverse")
```

to load tidyverse, you’ll need:

```
> library(tidyverse)
```

N.B.: for Ubuntu 20.04 LTS, you may need to install an XML package as follows:

```
$ sudo apt install libxml2-dev
```

data science:
visualization & modeling

visualization

we’re going to look at ***ggplot2***

> library(tidyverse)

ggplot2 is often referred to as a layered “grammar of graphics”

let’s consider a problem:

Do cars with larger displacement engines consume more fuel than cars with smaller engines?

What is the relationship between fuel efficiency and engine size? Let’s make this a precise investigation...

visualization (cont’d)

ggplot2 has a built-in data frame, **mpg**, which we can use to explore this problem

```
> mpg
```

```
> head(mpg); tail(mpg)
```

THINK OF THE COLUMNS AS **VARIABLES** AND THE ROWS AS **OBSERVATIONS**

among the variables are *displ* (the engine displacement in litres), *hwy* (the fuel efficiency in highway driving, measured in miles-per-gallon), *year* (the model year of the vehicle).

to plot fuel efficiency (hwy) versus engine displacement (displ), try the following command:

```
> ggplot(data = mpg) + geom_point(mapping = aes(x = displ, y=hwy))
```

identifies the dataset to use

adds a scatterplot “layer” to the graph