Week_2_IntroToRmd-Part1_Basics

Seema Plaisier

06/21/22

What you need to know about R for this course

This research course will be taught using R statistical programming environment because the analysis used done in the research paper about sex differentially expressed genes in the placenta was done in R. Like any programming language, experience is the key to proficiency, but we have designed this tutorial to give you some key concepts that you will need to understand in order to do the analysis required to achieve our research aims. General R tutorials are found easily on the internet if you are interested.

Things we will cover in this Rmd are:

1) Data types in R

a) basic data types: string, int, float, boolean
b) sets of data points: list, vector, matrix, data frame
c) accessing data in lists (indices, subsetting)
d) iterating over lists (for loop)

2) Working with files and directories

a) working directory
b) other directories
c) file reading into variables
d) saving output files

Basic data types and structures in R

Data can be stored as variables so that they can be abstracted and manipulated in R. This is analogous to working with variables in algebra (set a formula to be y = x + 5 so you can set anything to x and calculate the value of y to be 5 more). Depending on what data you store in a variable, R will assign that data a type and use that type to decide what you can do with that variable.

The basic data types in R that you will work with in this course are: 1) character: single character such as "a" or string of characters such as "abc" 2) double: numbers including whole numbers like 1 and numbers with fractions/decimals like 1.5 3) logical: boolean values TRUE or FALSE

The basic data structures (variables containing multiple pieces of information) you will work with in this course are: 1) vector: 1-dimensional collection of elements of one type, usually character, logical, integer or numeric 2) matrix: multi-dimensional collection of elements of one type (specify num rows and num cols) 3) list: multi-dimensional collection of elements that can be multiple data types 4) data frame: special type of

list where every element has the same length ("rectangular" list) 5) factors: data objects used to categorize and order repeated elements in a vector or list

For multi-dimensional data structures, you can use indices to access and modify specific elements of the list. Indices are noted with square braces, [ ]. Indices in R start at 1 (first value in list/vector) and end at the length of the list/vector (last index of a list of length 3 is 3 which holds the last value in the list/vector).

Control structures are programming tools you can use to formulate methods to process data. That is, they help you control which data you are doing something with and when to do it.

The basic control structures you will work with in this course are: 1) conditional: if, else statements to test if something is true before doing something 2) loops: for loops to run through elements of a list

TutorialsPoint is a great resource for more information on these topics: https://www.tutorialspoint.com/r/r_overview.htm Click on the topic on the left side of the page for whatever you'd like to see more examples of.

```
s = "this_is_a_string"  # set variable called s to a character string "this_is_a_string"
typeof(s)  # print out what type of data structure s holds
```

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

```
print(s) # prints the value stored in the variable s
```

```
## [1] "this_is_a_string"
```

```
n = 1 # set variable n to 1
print(n) # show the current value of the variable
```

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

```
typeof(n) # displays data type of current value of variable n
```

```
## [1] "double"
```

```
n = 1.5 # set same variable n to a new value overwriting the old one
print(n) # look to see that it is a new value of 1.5
```

```
## [1] 1.5
```

```
typeof(n) # look to see what data type it is now with the new value
```

```
## [1] "double"
```

```
b = TRUE # set variable b to a logical state TRUE
typeof(b) # display data type of the value of variable b
```

```
## [1] "logical"
```

```
if (n == 1) { # use a conditional statement to see if the variable n is currently set to 1
  b = TRUE  # if it is, set variable b equal to the logical value of TRUE
  print (b) # print the value of b
} else { # if the condition in the if statement above is not true, do the following code
  b = FALSE # set the variable b to logical value of FALSE
  print (b) # print the value of b
}
```

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

```
# this code does something very similar to the if statement above
#   but in far fewer lines of code
print (n == 1) # print whether it is true or false that n is equal to 1
```

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

```
# this shows that you can do the same thing in multiple ways
#   as long as you think through what you want to do
#   and know the coding tools available in R to do it

v = c("this","is","a","vector","of","character","strings")  # set variable v to a vector of strings
typeof(v) # shows the data type of a vector is the data type of its contents
```

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

```
print(v) # print the contents of v
```

```
## [1] "this"      "is"        "a"         "vector"    "of"        "character"
## [7] "strings"
```

```
length(v) # print the length of vector v
```

```
## [1] 7
```

```
v2 = c(1,2,3.4,5.1,8,10) # set the variable v2 to a vector of numbers
typeof(v2) # shows the data type of the vector is the data type of its contents
```

```
## [1] "double"
```

```
length(v2) # show the length of variable v2
```

```
## [1] 6
```

```
l = list("here","we","have","a","list","with",7,"elements") # set variable l to list of strings and num
typeof(l) # shows the data type of a list is 'list' as it can have many data types inside
```

```
## [1] "list"
```

```r
print(l) # print contents of l
```

```
## [[1]]
## [1] "here"
##
## [[2]]
## [1] "we"
##
## [[3]]
## [1] "have"
##
## [[4]]
## [1] "a"
##
## [[5]]
## [1] "list"
##
## [[6]]
## [1] "with"
##
## [[7]]
## [1] 7
##
## [[8]]
## [1] "elements"
```

```r
length(l) # print length of list l
```

```
## [1] 8
```

```r
m = matrix(v2,nrow=2,ncol=3) # set variable m to values of vector v2 filling rows and columns
print(m) # print contents of variable m
```

```
##      [,1] [,2] [,3]
## [1,]    1  3.4    8
## [2,]    2  5.1   10
```

```r
# create data.frame d containing the data from matrix m
# dimensions of m conserved in data frame d
# can set the row names at the same time as creating the data frame
d = data.frame(mat = m, row.names = c("row1","row2"))
# can set the names outside of when you are creating the data frame too
colnames(d) = c("col1","col2","col3")
print(d) # print contents of d
```

```
##      col1 col2 col3
## row1    1  3.4    8
## row2    2  5.1   10
```

```r
dim(d) # show the dimensions of the data frame, length and width for 2 dimensional
```

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

```r
# can also set a data frame to hold differnt types of data, each given a name
emp.data <- data.frame(
   emp_id = c (1:5),
   emp_name = c("Rick","Dan","Michelle","Ryan","Gary"),
   salary = c(623.3,515.2,611.0,729.0,843.25)
)
print(emp.data) # show the contents of employee data frame
```

```
##   emp_id emp_name salary
## 1      1     Rick 623.30
## 2      2      Dan 515.20
## 3      3 Michelle 611.00
## 4      4     Ryan 729.00
## 5      5     Gary 843.25
```

```r
str(emp.data) # show the elements of the data frame with the first few entries
```

```
## 'data.frame':    5 obs. of  3 variables:
##  $ emp_id  : int  1 2 3 4 5
##  $ emp_name: chr  "Rick" "Dan" "Michelle" "Ryan" ...
##  $ salary  : num  623 515 611 729 843
```

```r
summary(emp.data) # show the range of each element of the data frame
```

```
##      emp_id    emp_name            salary
##  Min.   :1   Length:5           Min.   :515.2
##  1st Qu.:2   Class :character   1st Qu.:611.0
##  Median :3   Mode  :character   Median :623.3
##  Mean   :3                      Mean   :664.4
##  3rd Qu.:4                      3rd Qu.:729.0
##  Max.   :5                      Max.   :843.2
```

```r
# different ways to access the data in a data frame

# can use the $ symbol to get all of a specific column/element
print(emp.data$emp_name)
```

```
## [1] "Rick"     "Dan"      "Michelle" "Ryan"     "Gary"
```

```r
# can use a conditional to set which row you are interested in
# indices are given as column,row
#    so indicating the column, comma, no row specified gives you the whole row
emp.data[emp.data$emp_id == 1,]
```

```
##   emp_id emp_name salary
## 1      1     Rick  623.3
```

```
# if the row has multiple entries, you can use the index number to access a specific entry
emp.data[emp.data$emp_id == 1,2]
```

```
## [1] "Rick"
```

```
# you can also use the row name
emp.data[emp.data$emp_id == 1,"emp_name"]
```

```
## [1] "Rick"
```

```
# you can use index number for both row and column
emp.data[1,2]
```

```
## [1] "Rick"
```

```
# you can assign a specific position in the data frame to a value as long as you access it correctly
emp.data[1,2] = "Morty"

# you can print the contents after making changes to see if everything looks right
print(emp.data)
```

```
##   emp_id emp_name salary
## 1      1    Morty 623.30
## 2      2      Dan 515.20
## 3      3 Michelle 611.00
## 4      4     Ryan 729.00
## 5      5     Gary 843.25
```

```
# if you make a mistake, you can also go back and run the original
# statement that filled the data frame to reset the data frame variable
# to its original contents and then try again

# this for loop runs through all the values of vector v2 assigning to the variable 'value'
for (value in v2) {
  print(value) # print the value of the variable 'value'
}
```

```
## [1] 1
## [1] 2
## [1] 3.4
## [1] 5.1
## [1] 8
## [1] 10
```

```
# this for loop makes a list starting at 1 and continuing through the length of vector v2
# then the variable 'i' is filled with the list generated
for (i in 1:length(v2)) {
  # can use 'i' to access a specific index of the vector v2
  print (paste0(i,",",v2[i])) # paste0 function makes a string of what you pass it
}
```

```
## [1] "1,1"
## [1] "2,2"
## [1] "3,3.4"
## [1] "4,5.1"
## [1] "5,8"
## [1] "6,10"
```

Set working directory and data directories

Your working directory is where all your output files will be stored by default. You can set it by copying the path of the directory from the explorer windows. Make sure you change the slashes to forward slashes '/' if needed.

The 'setwd' function sets your working directory to the path you specify. The 'list.files' and 'list.dirs' functions can be used to print what's inside that directory.

In the code you run in this class, you will need to change the path to the working directory so that your output files get put in your /home drive.

```
# store the path to the working directory in a variable
# make sure to include the / at the end of the directory path if you plan to put it together with the p
working_directory = "C:/Users/splaisie/Dropbox (ASU)/GenomicsCURE/Week 2/"  # THIS WILL NEED TO BE CHAN
#working_directory = "/scratch/splaisie/test_CURE"  # example of path to working directory on Agave

# set the working directory to where you want your output to go
setwd(working_directory)

# see what's currently in your working directory
# files:
list.files(working_directory)
```

```
##  [1] "~$tes-R_Walkthrough_video.docx"
##  [2] "~WRL3226.tmp"
##  [3] "dp_plot.pdf"
##  [4] "emp_salary_plot.pdf"
##  [5] "Extras.docx"
##  [6] "Notes-R_Walkthrough_video.docx"
##  [7] "Resources.docx"
##  [8] "subset.csv"
##  [9] "subset.txt"
## [10] "Week_2_IntroToRmd-Map_of_RStudio.pdf"
## [11] "Week_2_IntroToRmd-Map_of_RStudio.Rmd"
## [12] "Week_2_IntroToRmd-Part1-Basics.pdf"
## [13] "Week_2_IntroToRmd-Part1-Basics.Rmd"
## [14] "Week_2_IntroToRmd-Part2-Fancy.Rmd"
## [15] "Week_2_IntroToRmd.pdf"
## [16] "Week_2_IntroToRmd.Rmd"
```

```
# directories:
list.dirs(working_directory)
```

```
## [1] "C:/Users/splaisie/Dropbox (ASU)/GenomicsCURE/Week 2/"
```

```
# set the path to other directories you might need to reference
untrimmed_directory = "C:/Users/splaisie/Dropbox (ASU)/GenomicsCURE/Week 3/"
print(untrimmed_directory)
```

```
## [1] "C:/Users/splaisie/Dropbox (ASU)/GenomicsCURE/Week 3/"
```

```
# don't need to set it to working directory

# can instead use paste0 command to add the file path of a directory to the file name
# make sure you have the / at the end to make sure the file name is not added to the deepest directory
file_path = paste0(untrimmed_directory, "DE_Pipeline_UntrimmedData.Rmd")
print (file_path)
```

```
## [1] "C:/Users/splaisie/Dropbox (ASU)/GenomicsCURE/Week 3/DE_Pipeline_UntrimmedData.Rmd"
```

Reading data files

Reading data into variables is a very important task in R. For most scientific problems, you will read a table of data commonly separated by commas or tabs. The 'read.csv' function can be used to read tabular data from files; the delimiter is set to be a comma by default in this function. If you have tab-delimited data, you can set the delimiter to a tab. Running these commands will put variables containing your data in the workspace.

```
variants = read.csv("subset.csv")

variants2 = read.csv("subset.txt", sep = "\t")
```

Ways to look at the data

Once data is read in from a file (or created with code, see below), you can see them in the Environment tab and should be able to check if everything worked by clicking on the variable name. You can also use the 'head' function to see the top rows of the data table or the 'summary' function to see a summary of the counts and ranges of the data variable or the 'str' function to show the structure of the data.

```
head(variants)
```

```
##   X.1 X X.3 X.2       ALT
## 1   1 1   1   1         G
## 2   2 2   2   2         T
## 3   3 3   3   3         T
## 4   4 4   4   4 CTTTTTTTT
## 5   5 5   5   5    CCGCGC
## 6   6 6   6   6         T
```

```
summary(variants)
```

```
##       X.1             X             X.3           X.2          ALT
##  Min.   :  1   Min.   :  1   Min.   :  1   Min.   :  1   Length:801
##  1st Qu.:201   1st Qu.:201   1st Qu.:201   1st Qu.:201   Class :character
```

```
##  Median :401   Median :401   Median :401   Median :401   Mode  :character
##  Mean   :401   Mean   :401   Mean   :401   Mean   :401
##  3rd Qu.:601   3rd Qu.:601   3rd Qu.:601   3rd Qu.:601
##  Max.   :801   Max.   :801   Max.   :801   Max.   :801
```

```
str(variants)
```

```
## 'data.frame':    801 obs. of  5 variables:
##  $ X.1: int  1 2 3 4 5 6 7 8 9 10 ...
##  $ X  : int  1 2 3 4 5 6 7 8 9 10 ...
##  $ X.3: int  1 2 3 4 5 6 7 8 9 10 ...
##  $ X.2: int  1 2 3 4 5 6 7 8 9 10 ...
##  $ ALT: chr  "G" "T" "T" "CTTTTTTTT" ...
```

Subset the data

One major task in analyzing genome or transcriptome level data is to select specific information of interest for your research aims. In this example, we are taking a subset of the variants data frame loaded in the previous chunk. We are selecting specific columns containing specific information, alternative alleles, which are points in the sequence that are different from the sequence in the reference genome. We are dividing them up by what nucleotide is actually present and using basic plotting tools in R to make a bar graph so we can see which nucleotide is most often present in that position as an alternative allele. Alternative alleles or SNPs (single nucleotide polymorphisms) are the first line of suspects for variations that cause different phenotypes in a cell or tissue.

```
# select specific columns and make a new data frame
subset = data.frame(variants[,c(1:3,5)])

# look at our data to make sure we got the columns we wanted
head(subset) # look at the top rows
```

```
##   X.1 X X.3       ALT
## 1   1 1   1         G
## 2   2 2   2         T
## 3   3 3   3         T
## 4   4 4   4 CTTTTTTTT
## 5   5 5   5    CCGCGC
## 6   6 6   6         T
```

```
str(subset) # look at the components and the first few elements of each
```

```
## 'data.frame':    801 obs. of  4 variables:
##  $ X.1: int  1 2 3 4 5 6 7 8 9 10 ...
##  $ X  : int  1 2 3 4 5 6 7 8 9 10 ...
##  $ X.3: int  1 2 3 4 5 6 7 8 9 10 ...
##  $ ALT: chr  "G" "T" "T" "CTTTTTTTT" ...
```

```
alt_alleles = subset$ALT # select 1 specific column of the data

# divide up the data
# use a conditional statement to determine the indices of the data frame
```

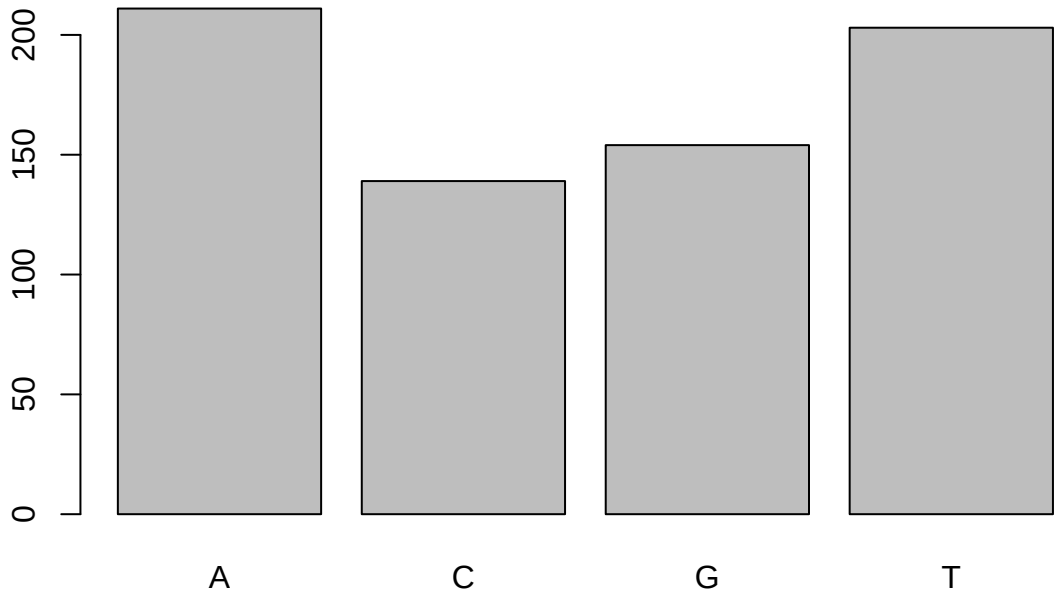
```
#   are have alleles that are a specific nucleotide
#  those indices are used to subset the column you pulled
# put the 4 subsets of data together in a list with the c() command
snps <- c(alt_alleles[alt_alleles=="A"],
  alt_alleles[alt_alleles=="T"],
  alt_alleles[alt_alleles=="G"],
  alt_alleles[alt_alleles=="C"])

factor_snps <- factor(snps) # tell R to treat this data as categories
str(factor_snps)
```

```
##  Factor w/ 4 levels "A","C","G","T": 1 1 1 1 1 1 1 1 1 1 ...
```

```
summary(factor_snps) # take a final look at what you got out to make sure it looks right
```

```
##   A   C   G   T
## 211 139 154 203
```

```
plot(factor_snps) # basic plot of the number of each nucleotide
```



Save data to a file

The 'write.csv' function writes data to a file. The file will be saved in the working directory unless otherwise specified.

```
write.csv(subset, file = "subset.csv")
```

List all the packages used for future reference

```
sessionInfo()
```

```
## R version 4.2.0 (2022-04-22 ucrt)
## Platform: x86_64-w64-mingw32/x64 (64-bit)
## Running under: Windows 10 x64 (build 19044)
##
## Matrix products: default
##
## locale:
## [1] LC_COLLATE=English_United States.utf8
## [2] LC_CTYPE=English_United States.utf8
## [3] LC_MONETARY=English_United States.utf8
## [4] LC_NUMERIC=C
## [5] LC_TIME=English_United States.utf8
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base
##
## loaded via a namespace (and not attached):
##  [1] compiler_4.2.0  magrittr_2.0.3  fastmap_1.1.0   cli_3.3.0
##  [5] tools_4.2.0     htmltools_0.5.2 rstudioapi_0.13 yaml_2.3.5
##  [9] stringi_1.7.6   rmarkdown_2.14  highr_0.9       knitr_1.39
## [13] stringr_1.4.0   xfun_0.31       digest_0.6.29   rlang_1.0.2
## [17] evaluate_0.15
```