R Zero

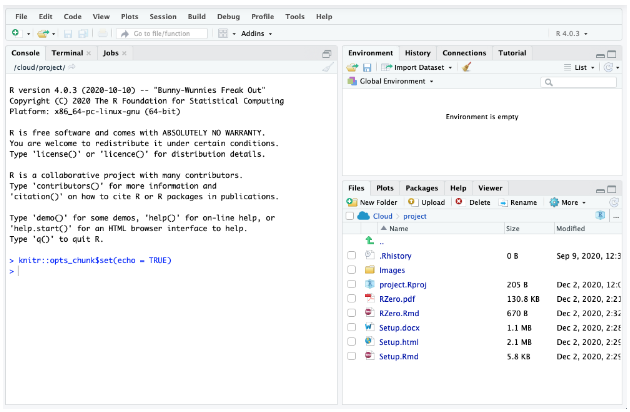
Andrew Bell

Learning Objectives

- Become familiar with the RStudio environment
- Identify the key elements of the R programming language
  - Functions
  - Packages
  - Objects
  - Operators
  - Vectors
  - Data frames
- Write your first R code

The RStudio Environment

Load RStudio Workbench in your favorite browser, open the Transition to R project, and you should see a window similar to that shown below.



```
R version 4.0.3 (2020-10-10) -- "Bunny-Wunnies Freak Out"
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.

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.

> knitr::opts_chunk$set(echo = TRUE)
> 
```

Name	Size	Modified
..		
.Rhistory	0 B	Sep 9, 2020, 12:3
Images		
project.Rproj	205 B	Dec 2, 2020, 12:0
RZero.pdf	130.8 KB	Dec 2, 2020, 2:21
RZero.Rmd	670 B	Dec 2, 2020, 2:32
Setup.docx	1.1 MB	Dec 2, 2020, 2:28
Setup.html	2.1 MB	Dec 2, 2020, 2:29
Setup.Rmd	5.8 KB	Dec 2, 2020, 2:29

Figure 1: RStudio user-interface. Note: the positioning of some windows may differ from that shown here and can be changed to suit the user.

The following windows will appear as part of the RStudio user-interface:

- **Console window** - The window on the right in Figure 1 is the standard R console window (the only one you would see if you used only R and not R via RStudio. This is the window where you type in commands and the results are returned.

- **Workspace / History** - This window (left side in Figure 1) shows all the objects that you have created in the current R session (Workspace tab) and the commands you have used in the current R session (History tab).
- **Files / Plots / Packages / Help** - This window (bottom right in Figure 1) is primarily used for displaying plots (graphs) and for using the help system.

R is a command driven program, which means that you type in code to the Console window and it returns the result. The advantage of this is that the software is very powerful, but the disadvantage is that to start with the software requires a steep learning curve!

You can either type code directly into the Console window, or you can type it into a text file, and then run the code you have written in the Console window. The advantage of the latter is that you can save the code, so that to run it a second time at a later date, simply open the text file and run the code, you do not need to type it in a second time! To create a text file to write your code in select

File -> New File -> R Script

and the RStudio window changes to:

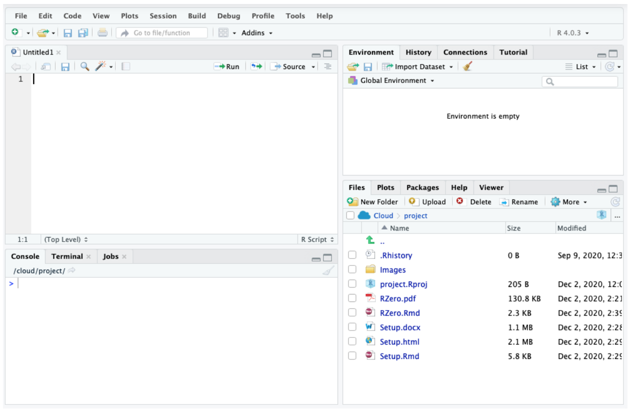

File Edit Code View Plots Session Build Debug Profile Tools Help
Untitled1
Run
Source
Environment History Connections Tutorial
Import Dataset
Global Environment
Environment is empty
Console Terminal Jobs
/cloud/project/
Files Plots Packages Help Viewer
New Folder Upload Delete Rename More
Cloud project
Name Size Modified

Figure 2: RStudio user-interface with a new script named Untitled1 for writing R code open.

To save the R code (once you've written some!), go to

File -> Save

and a dialog box will appear asking where (on which drive) you would like to save the code. Once saved, you can use your code again, so no need to type it in a second time. If you close RStudio and then open it again you can read in your code by going to

File -> Open File. . .

or navigate to your file in the "Files/Plots/Pkgs/Help" window (see image below) and find the file from the location where you saved it.

For a comprehensive one-page guide to the RStudio IDE, check out this 'cheatsheet'

Key Elements of R Programming Language

Now let's start to explore what R can do for us.

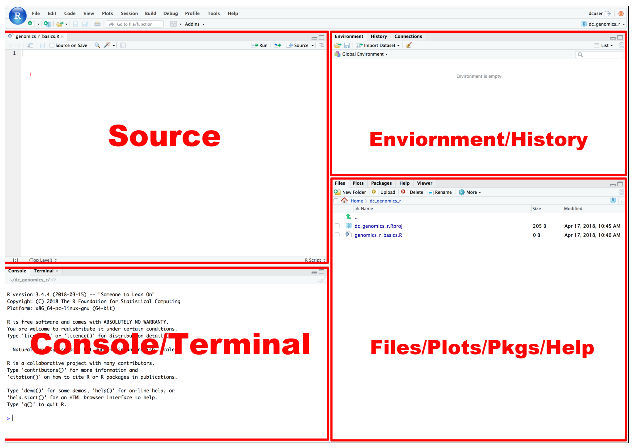


Figure 3: Typical 4 pane layout of RStudio IDE

Functions

A function in R (or any computing language) is a short program that takes some input and returns some output. Functions may seem like an advanced topic (and they are), but you probably have already used many functions if you've used Microsoft Excel (think sum() or mean() - both of which are tiny programs that running specific actions on data).

####**Exercise 1**

Try the following functions by writing them in your script. See if you can guess what they do, and make sure to add comments to your script about your assumed purpose.

```
dir()
```

```
## [1] "Data"                 "Images"               "R Basics Solutions"
## [4] "RZero.Rmd"            "Setup.pdf"            "Setup.Rmd"
```

```
sessionInfo()
```

```
## R version 4.1.1 (2021-08-10)
## Platform: x86_64-pc-linux-gnu (64-bit)
## Running under: Red Hat Enterprise Linux 8.6 (Ootpa)
##
## Matrix products: default
## BLAS/LAPACK: /usr/lib64/libopenblasp-r0.3.15.so
##
## locale:
##  [1] LC_CTYPE=en_US.UTF-8       LC_NUMERIC=C
##  [3] LC_TIME=en_US.UTF-8        LC_COLLATE=en_US.UTF-8
##  [5] LC_MONETARY=en_US.UTF-8    LC_MESSAGES=en_US.UTF-8
##  [7] LC_PAPER=en_US.UTF-8       LC_NAME=C
##  [9] LC_ADDRESS=C               LC_TELEPHONE=C
## [11] LC_MEASUREMENT=en_US.UTF-8 LC_IDENTIFICATION=C
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base
##
```

```
## loaded via a namespace (and not attached):
##  [1] compiler_4.1.1    magrittr_2.0.1    tools_4.1.1       htmltools_0.5.1.1
##  [5] yaml_2.2.1        stringi_1.7.3     rmarkdown_2.10    knitr_1.33
##  [9] stringr_1.4.0     xfun_0.25         digest_0.6.27     rlang_0.4.11
## [13] evaluate_0.14
```

```
date()
```

```
## [1] "Wed Jun 15 12:12:26 2022"
```

```
Sys.time()
```

```
## [1] "2022-06-15 12:12:26 EDT"
```

####**Exercise 1 solutions**

`dir()` # Lists files in the working directory `sessionInfo()` # Gives the version of R and additional info including on attached packages `date()` # Gives the current date `Sys.time()` # Gives the current time

An R Function has three key properties:

- Functions have a name (e.g. `dir`); note that functions are case sensitive
- Function names are followed by a pair of `()`
- Inside the parentheses, a function may take 0 or more **arguments**

An **argument** may be a specific input for your function and/or may modify the function's behavior. For example the function `round()` will round a number with a decimal:

```
# This will round a number to the nearest integer
round(3.14159)
```

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

Getting help with function arguments ?

What if you wanted to round to one significant digit? `round()` can do this, but you may first need to read the help to find out how. To see the help (In R sometimes also called a "vignette") enter a `?` in front of the function name:

```
?round()
```

The "Help" tab will show you information (often, too much information). You will slowly learn how to read and make sense of help files. Checking the "Usage" or "Examples" headings is often a good place to look first. If you look under "Arguments," we also see what arguments we can pass to this function to modify its behavior.

args

You can also see a function's argument using the `args()` function:

```r
args(round)
```

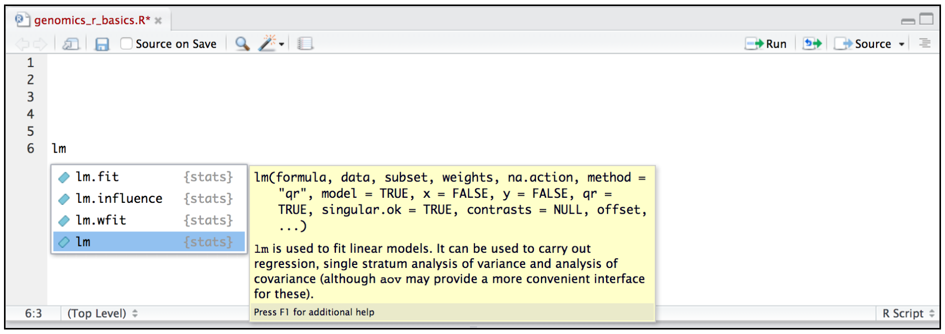
```
## function (x, digits = 0)
## NULL
```

```r
#add code below to round 3.14159 to 3.14
round(3.14159, digits = 2)
```

```
## [1] 3.14
```

Contentual Help

Here is one last bonus we will mention about RStudio. It's difficult to remember all of the arguments and definitions associated with a given function. When you start typing the name of a function and hit the `Tab` key, RStudio will display functions and associated help:



```
lm
lm.fit        {stats}
lm.influence  {stats}
lm.wfit       {stats}
lm            {stats}
```

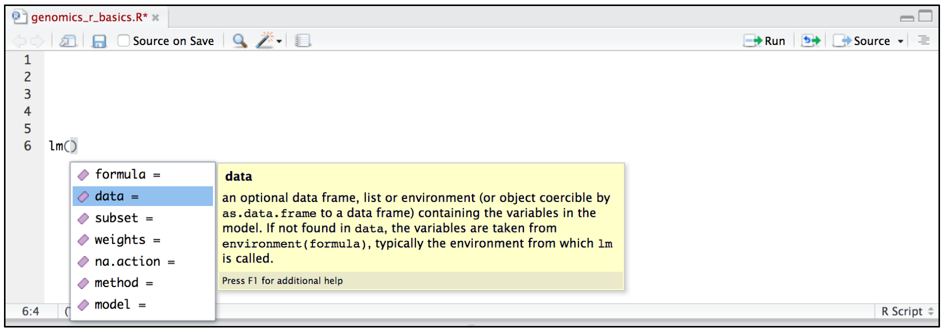
```
lm(formula, data, subset, weights, na.action, method =
    "qr", model = TRUE, x = FALSE, y = FALSE, qr =
    TRUE, singular.ok = TRUE, contrasts = NULL, offset,
    ...)
```

`lm` is used to fit linear models. It can be used to carry out regression, single stratum analysis of variance and analysis of covariance (although `aov` may provide a more convenient interface for these).

Press F1 for additional help

Figure 4: Contentual Help 1

Once you type a function, hitting the `Tab` inside the parentheses will show you the function's arguments and provide additional help for each of these arguments.



```
lm()
formula =
data =
subset =
weights =
na.action =
method =
model =
```

data

an optional data frame, list or environment (or object coercible by `as.data.frame` to a data frame) containing the variables in the model. If not found in `data`, the variables are taken from `environment(formula)`, typically the environment from which `lm` is called.

Press F1 for additional help

Figure 5: Contentual Help 1

derived in part from https://datacarpentry.org/genomics-r-intro/01-introduction/index.html

Packages

Built into R is a wide range of functions that'll assist you in analyzing and visualizing your data. However, the true power of R comes from the millions of additional functions you can load into your coding environment.

There is an enormous development community that is continually updating and creating new packages that enhance the functionality of R.

To install packages use the `install.package()` function and to load a package into your current environment use the `library()` function. For each of these functions, pass the name of the package inside quotations marks (e.g. `install.packages("cowsay")` and `library("cowsay")`

```
#type in the the code to install and load the cowsay package below
```

Objects

What might be called a variable in many languages is called an object in R.

To create an object you need:

- a name (e.g. 'a')
- a value (e.g. '1')
- the assignment operator ('<-')

Nifty keyboard shortcut alert! To make a <- : hold `Alt` (or `option` on a Mac) and press the `-` (minus sign) key.

```
# this line creates the object 'a' and assigns it the value '1'

a <- 1
```

Next, run this line of code in your script. You can run a line of code by hitting the Run button that is just above the first line of your script in the header of the Source pane or you can use the appropriate shortcut:

- Windows execution shortcut: `Ctrl+Enter`
- Mac execution shortcut: `Cmd()+Enter`

To run multiple lines of code, you can highlight all the line you wish to run and then hit Run or use the shortcut key combo listed above.

```
#####Create the following objects; give each object an appropriate name (your best guess at what name t

#Create an object that has the value of number of pairs of human chromosomes


#Create an object that has a value of your favorite gene name


#Create an object that has this URL as its value: "ftp://ftp.ensemblgenomes.org/pub/bacteria/release-39

#Create an object that has the value of your birthday
```

####*Reassigning object names or deleting objects

Once an object has a value, you can change that value by overwriting it. R will not give you a warning or error if you overwriting an object, which may or may not be a good thing depending on how you look at it.

```
a <- 4
```

You can also remove an object from R's memory entirely. The `rm()` function will delete the object.

```
# delete the object 'gene_name'
rm(gene_name)
```

```
## Warning in rm(gene_name): object 'gene_name' not found
```

Operators

Once an object exists (which by definition also means it has a mode), R can appropriately manipulate that object. For example, objects of the numeric modes can be added, multiplied, divided, etc. R provides several mathematical (arithmetic) operators including:

Operator	Description
+	Addition
-	Subtraction
*	Multiplication
/	Division
^ or **	Exponentiation
a%?%b	Integer Division (where remainder is discarded)
a%%b	modulus (returns remainder after division)

These can be used with literal numbers:

```
(1+ (5**0.05))/2
```

```
## [1] 1.041899
```

and importantly, can also be used on any object that evaluates to (i.e. interpreted by R) a numeric object:

```
number_of_presidents <- 45

as_of_jan202021_number_of_presidents <- number_of_presidents + 1

as_of_jan202021_number_of_presidents
```

```
## [1] 46
```

Vectors

Vectors are probably the most used commonly used object type in R. *A vector is a collection of values that are all of the same type (numbers, characters, etc.).*

One of the most common ways to create a vector is to use the `c()` function - the "concatenate" or "combine" function. Inside the function you may enter one or more values; for multiple values, separate each value with a comma:

```
## Create a list of your favorite foods
fav_foods <- c("Tacos","Burritos","Hamburgers")

## Check the mode, length, and structure of 'fav_foods'

# Mode
mode(fav_foods)
```

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

```
# Length
length(fav_foods)
```

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

```
# Structure
str(fav_foods)
```

```
##  chr [1:3] "Tacos" "Burritos" "Hamburgers"
```

Data frame

A data frame is the standard way in R to store tabular data. A data fame could also be thought of as a collection of vectors, all of which have the same length.

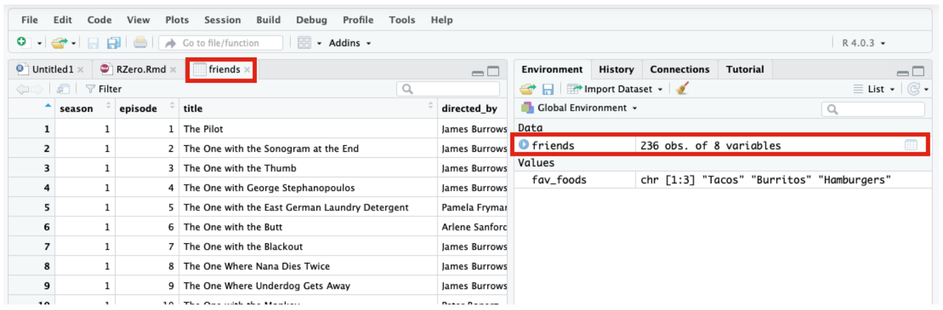
Importing tablular data into R

There are several ways to import data into R. For our purpose here, we will focus on using the tools every R installation comes with (so called "base" R) to import a comma-delimited file containing the results of our variant calling workflow. We will need to load the sheet using a function called read.csv().

```
# Before using the read.csv() function, use R's help feature to answer the following questions.

#Hint: Entering '?' before the function name and then running that line will bring up the help document

#A) What is the default parameter for 'header' in the read.csv() function?

#B) What argument would you have to change to read a file that was delimited by semicolons (;) rather t

#C) What argument would you have to change to read file in which numbers used commas for decimal separa

#D) What argument would you have to change to read in only the first 10,000 rows of a very large file?
```

Now, let's read the file `friends_info.csv` which is located in `/Cloud/project/Data/`. Assign this data frame to an objected called `friends`. The first argument to ass to our `read.csv()` function is the file path for our data. The file path must be in quotes and now is a good time to remember to use tab autocompletion. **If you use tab autocompletion you avoid typos and errors in file paths. Use it!**

```
## read in a CSV file and save it as 'friends'

friends <- read.csv("~/TransitionToR/1 - R Basics/Data/friends_info.csv")
```



	season	episode	title	directed_by
1	1	1	The Pilot	James Burrows
2	1	2	The One with the Sonogram at the End	James Burrows
3	1	3	The One with the Thumb	James Burrows
4	1	4	The One with George Stephanopoulos	James Burrows
5	1	5	The One with the East German Laundry Detergent	Pamela Frymai
6	1	6	The One with the Butt	Arlene Sanforc
7	1	7	The One with the Blackout	James Burrows
8	1	8	The One Where Nana Dies Twice	James Burrows
9	1	9	The One Where Underdog Gets Away	James Burrows

```
Data
friends      236 obs. of 8 variables
Values
  fav_foods  chr [1:3] "Tacos" "Burritos" "Hamburgers"
```

Figure 6: View our new data frame

One of the first things you should notice is that in the Environment window, you have the newly created `friends` object. listed with 236 obs. (observations/rows) of 29 variables (columns). Double-clicking on the name of the object will open a view of the data in a new tab.

Summarizing and determining the structure of a data frame.

Using only two functions, we can learn a lot about a data frame including some summary statistics as well as well as the "structure" of the data frame. Let's examine what each of these functions can tell us:

```r
## get summary statistics on a data frame

summary(friends)
```

```
##      season          episode          title            directed_by       
##  Min.   : 1.000   Min.   : 1.00   Length:236         Length:236        
##  1st Qu.: 3.000   1st Qu.: 6.00   Class :character   Class :character  
##  Median : 5.000   Median :12.00   Mode  :character   Mode  :character  
##  Mean   : 5.377   Mean   :12.38                                        
##  3rd Qu.: 8.000   3rd Qu.:18.00                                        
##  Max.   :10.000   Max.   :25.00                                        
##   written_by          air_date         us_views_millions  imdb_rating   
##  Length:236         Length:236         Min.   :15.65     Min.   :7.200  
##  Class :character   Class :character   1st Qu.:22.30     1st Qu.:8.200  
##  Mode  :character   Mode  :character   Median :24.48     Median :8.400  
##                                        Mean   :25.37     Mean   :8.461  
##                                        3rd Qu.:27.73     3rd Qu.:8.700  
##                                        Max.   :52.90     Max.   :9.700  
```

Our data frame had 8 variables, so we get 8 fields that summarize data. The `imdb_rating`, `us_views_millions`, `episode`, and `season` variables are numerical data and so you get summary statistics on the min/max values, as well as mean, median, and interquartile ranges. Variables like `air_date` and `title` are treated as categorical data.

Before we analyze the data further, we also need to know a little more about the data frame structure to do that we use the `str()` function:

```r
## get the structure of a data frame

str(friends)
```

```
## 'data.frame':    236 obs. of  8 variables:
```

```
##  $ season           : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ episode          : int  1 2 3 4 5 6 7 8 9 10 ...
##  $ title            : chr  "The Pilot" "The One with the Sonogram at the End" "The One with the Thuml
##  $ directed_by      : chr  "James Burrows" "James Burrows" "James Burrows" "James Burrows" ...
##  $ written_by       : chr  "David Crane & Marta Kauffman" "David Crane & Marta Kauffman" "Jeffrey As
##  $ air_date         : chr  "1994-09-22" "1994-09-29" "1994-10-06" "1994-10-13" ...
##  $ us_views_millions: num  21.5 20.2 19.5 19.7 18.6 18.2 23.5 21.1 23.1 19.9 ...
##  $ imdb_rating      : num  8.3 8.1 8.2 8.1 8.5 8.1 9 8.1 8.2 8.1 ...
```

Ok, that's a lot up unpack! Some things to notice.

- the object type data.frame is displayed in the first row along with its dimensions, in this case 236 observations (rows) and 8 variables (columns)
- Each variable (column) has a name (e.g. title). This is followed by the object mode (e.g. factor, int, num, etc.). Notice that before each variable name there is a $ - this will be important later.

Exercise: Import explore new data set

```
## Import the friends_emotions.csv found in the /Data folder and assign the data frame to a new object

## Explore the structure of emotions. How many variables and observations does your data frame have? Wha

## Run summary statistics on your data frame? Is there anything interesting from this summary?
```

Your first code with COWSAY

For your next grant, you'd really like to include a figure what has a cow asking (politely) for a strong score. Let's figure out how to do that!

You are in luck, someone kind soul in this world created a package called "cowsay" that draws animals with dialogs boxes that you can populate!

1. In your first two lines of code, install and load the cowsay package
2. In line 4 of your code, use one of the help methods we learned to ascertain more about the cowsay package.
3. Now that we know the cowsay package contains a single function, say(), determine what arguments you can pass to the function.
4. In line 5 of your code, called the say() function and pass it the phrase/string "Me need $"
5. In line 6 of your code, let's create an object that contains your phrase
6. In line 7 of your code, pass your object to the say().
7. Cats are great but we really want a different animal to convey our feelings. Determine what argument you need to call to change your animal.
8. Red text might be a little aggressive for your grant, determine what argument you need to call to change the color of your phrase. While you are at it, color the color of the animal as well.
9. Some packages have built in data. In COWSAY, there is a "rms" data set that pulls a random 'quote 'fact' about Richard Stallman from rmsfact::rmsfact() package. Instead of having your animal say your phrase, have it say a random quote from Richard.