MAD – Data Analysis & Biostatistics in R

Getting to Work

James R. Hunter, Ph.D.

DIPA, EPM, UNIFESP

11 September 2020

MAD

Administrative Notes

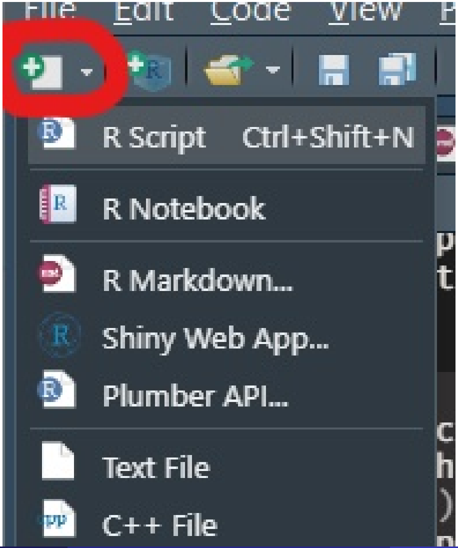
- Piazza is a **bust**. Not going to use it
  - Questions through all the traditional means
- Chapter 9 has had a major revision. If you downloaded before 5/9
  - Need a new download

Section 1

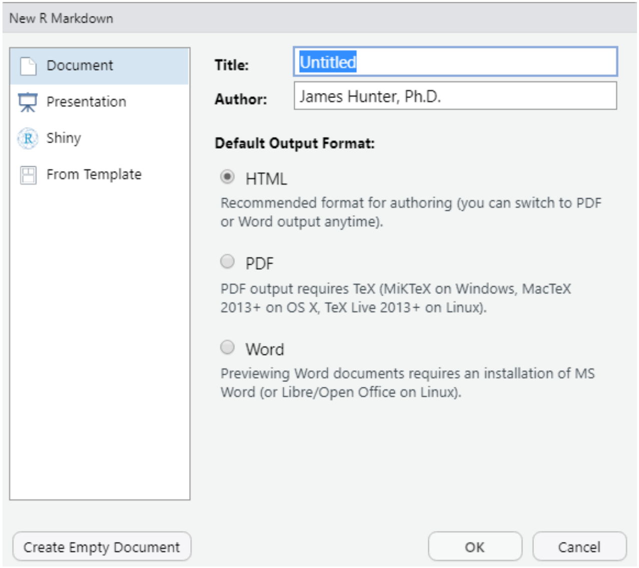
Submitting Homework and Other Documents

New R Markdown Document

- Go to upper left corner of screen (under File menu item)



Click on "R Markdown..."



Title: Untitled

Author: James Hunter, Ph.D.

Default Output Format:

[● HTML]

Recommended format for authoring (you can switch to PDF or Word output anytime).

○ PDF

PDF output requires TeX (MiKTeX on Windows, MacTeX 2013+ on OS X, TeX Live 2013+ on Linux).

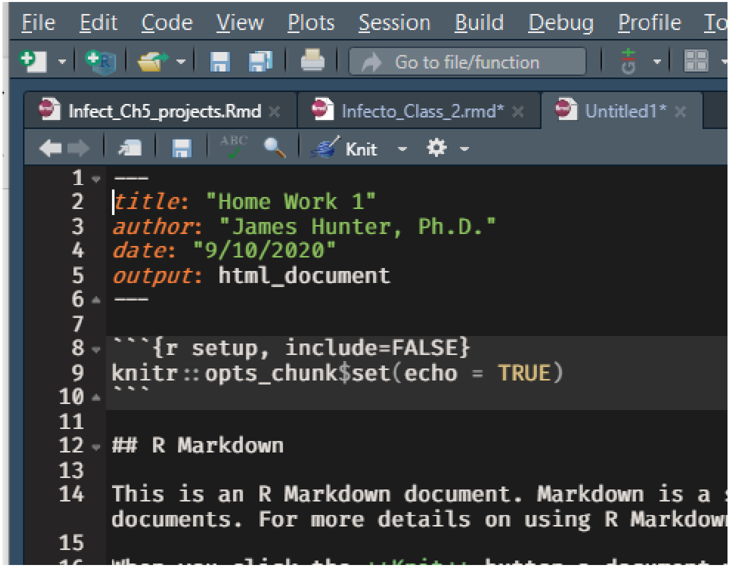
○ Word

Previewing Word documents requires an installation of MS Word (or Libre/Open Office on Linux).

Next Step

- Click on “Document” in left column
- Fill in title: “HomeWork 1”
- Fill in Author: your name
- Select HTML or PDF
  - PDF *only* if you know you have LaTeX installed on machine
- Click `OK` button

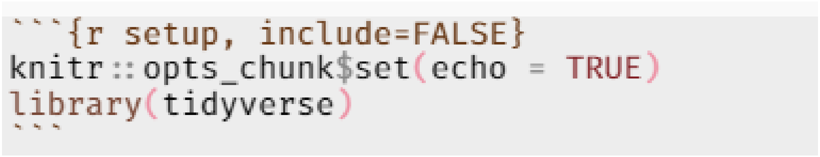
Here is your RMD Document



~~~
---
title: "Home Work 1"
author: "James Hunter, Ph.D."
date: "9/10/2020"
output: html_document
---

```{r setup, include=FALSE}
knitr::opts_chunk$set(echo = TRUE)
```

## R Markdown

This is an R Markdown document. Markdown is a
documents. For more details on using R Markdow
~~~

Working with .rmd Document

- Select and Erase all content from `## R Markdown` down to end
- Copy question 1 from homework sheet to 2 lines after the end of block
- To show solution of question 1, open code block
  - Can type CTRL-ALT-I
  - Can type format characters (see delimiters of block below)



~~~
```{r setup, include=FALSE}
knitr::opts_chunk$set(echo = TRUE)
library(tidyverse)
```
~~~

Ready to Answer Questions

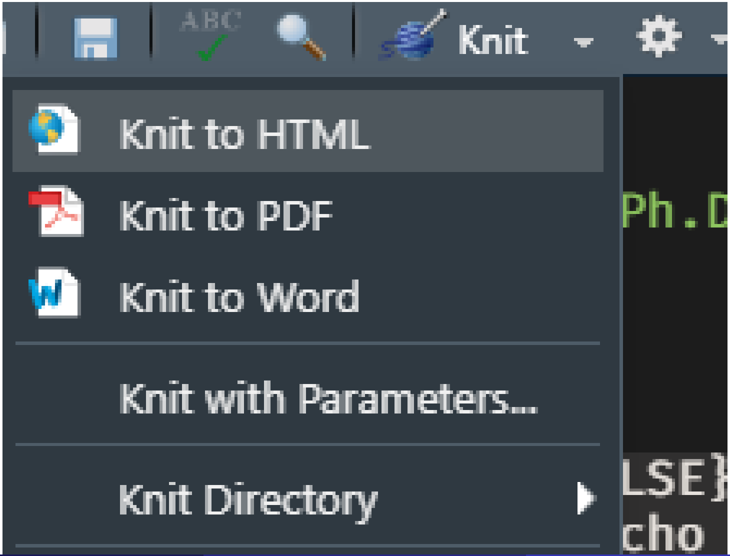
- Write code for your answer
  - To test your code, CTRL-Enter each line
    - Or hit green arrow at top right of code block
- Go on to next question through to end
- Save the file (often)

Section 2

Transforming your .rmd to HTML

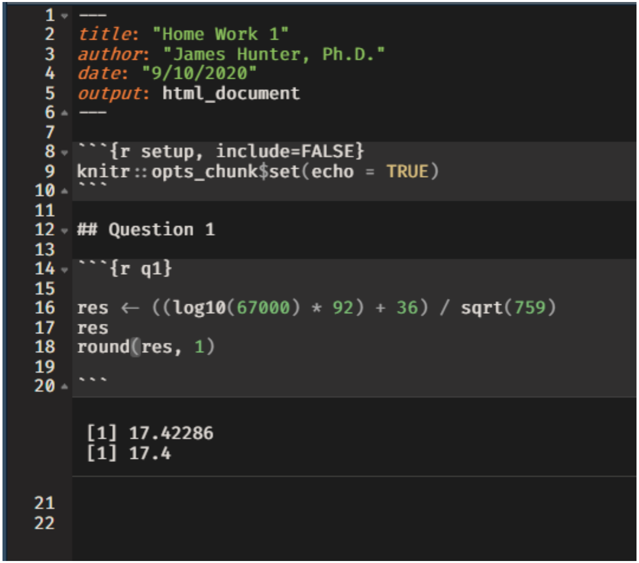
Transforming your .rmd to HTML

- Click on `Knit` above document



- Click on `Knit to HTML`
- If no major programming errors, it will do the calculations and give a result in web browser form

Raw .rmd File

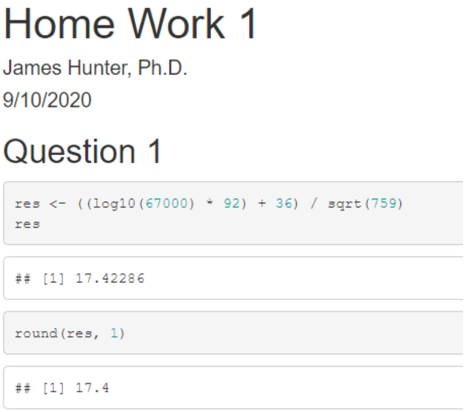


~~~
---
title: "Home Work 1"
author: "James Hunter, Ph.D."
date: "9/10/2020"
output: html_document
---

```{r setup, include=FALSE}
knitr::opts_chunk$set(echo = TRUE)
~~~

Question 1

```{r q1}

res ← ((log10(67000) * 92) + 36) / sqrt(759)
res
round(res, 1)

```
```

```
[1] 17.42286
[1] 17.4
```
```

## HTML Version



Home Work 1

James Hunter, Ph.D.

9/10/2020

Question 1

```
res <- ((log10(67000) * 92) + 36) / sqrt(759)
res
```

```
## [1] 17.42286
```

```
round(res, 1)
```

```
## [1] 17.4
```

Section 3

Homework from Week One

Question 1 - R as Calculator and PEMDAS

- Evaluate this expression in R; round to 1 decimal place

$$\frac{log_{10}(67000) * 92 + 36}{\sqrt{759}}$$

```
res <- ((log10(67000) * 92) + 36) / sqrt(759)
res
```

```
## [1] 17.42286
```

```
round(res, 1)
```

```
## [1] 17.4
```

Question 2 – Indexing a Vector

- Vector of 20 random numbers created with

```r
set.seed(42) # so everyone produces the same answer
x <- round(rnorm(20, mean = 100, sd = 10), 2) # returns v
x
```

```
##  [1] 113.71  94.35 103.63 106.33 104.04  98.94 115.12
## [11] 113.05 122.87  86.11  97.21  98.67 106.36  97.16
```

- What is the 2nd element of `x`?
  - `x[2]` = 94.35
- What is the class and type of `x`?
  - `class(x)` = numeric
  - `typeof(x)` = double
- What is the maximum value of `x`?
  - `max(x)` = 122.87

Question 3 - Information about Data Frame

- Based on data frame “einstein_soro.rds”
  - Binary stored form of data about COVID-19 sorological tests performed at Albert Einstein Hospital
  - From FAPESP Data Consortium files
- 1st Step – Load Data Frame

```
soro <- readRDS("C:/Users/james/OneDrive/Documents/MAD/MAD-Infecto-2020/einstein_so
str(soro)
```

```
## 'data.frame':    200 obs. of  10 variables:
##  $ pacid     : chr  "b6d668e4f818f7b3643ed593b8fb902bf9d2501e" "a090625661c06e9c
##  $ dt_collect: chr  "28/05/2020" "11/05/2020" "16/06/2020" "10/06/2020" ...
##  $ analysis  : chr  "IgM, COVID19" "IgG, COVID19" "IgG, COVID19" "COVID IgM Inte
##  $ result    : chr  "0.74" "0.03" "0.02" "Não reagente" ...
##  $ unit      : chr  "AU/ml" "AU/ml" "AU/ml" "NULL" ...
##  $ reference : chr  "<=0.90" "<=0.90" "<=0.90" "" ...
##  $ sex       : Factor w/ 2 levels "female","male": 2 1 1 2 1 1 1 2 1 2 ...
##  $ birth_yr  : num  1989 1975 1997 2006 1983 ...
##  $ uf        : Factor w/ 25 levels "AC","AL","AM",..: 24 9 24 24 24 24 24 24 24
##  $ city      : Factor w/ 21 levels "BARUERI","CAMPINAS",..: 19 NA 19 19 19 19 19
```

Questions about `soro`

a. How many cases are in this data base?
   - First line of `str()` tells you: `200 obs. of  10 variables`
   - Can also calculate as `nrow(soro)` = 200
b. How many variables are in this data base?
   - First line of `str()`
   - Can also calculate as `length(soro)` = 10
c. What is the class and type of `soro`?
   - `class(soro)` = data.frame
   - `typeof(soro)` = list
d. What is the date of collection (`dt_collect`) of the 3rd case?
   - `soro$dt_collect[3]` = 16/06/2020

soro Questions e and f

e. Print the 5th to the 10th birth years (`birth_yr`).

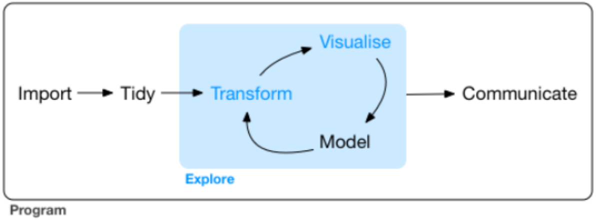
```
soro$birth_yr[5:10]
```

```
## [1] 1983 1963 1988 1971 1968 1976
```

f. How many different cities are represented in this sample?
    - `levels(soro$city)` = 21

Section 4

Data Analysis Projects


Import
Tidy
Transform
Visualise
Model
Communicate
Explore
Program

Project Workflow

- Tidyverse diagram shows principal steps in organizing computational tasks
- **However**
  - Much planning necessary before we even start with *Import*
  - Right at start of project

Initial Issues in Project Planning – Biological/Medical

- What is my research topic? - What is the large-scale question?
- What has science said on this topic previously?
  - Literature search
  - What sources to use (PubMed, Google Scholar, etc.)?
  - When will I have read enough?

Initial Issues in Project Planning – Technique

- What is the type of study I will conduct?
  - Experimental
  - Observational
  - Case-Control
  - Other type?
- What is my research hypothesis?
- What experiments/field work/data gathering support that?
- What will be my primary analytic technique?
  - What have others done?
    - Were they effective?
    - Produce interesting results?

Data for My Project

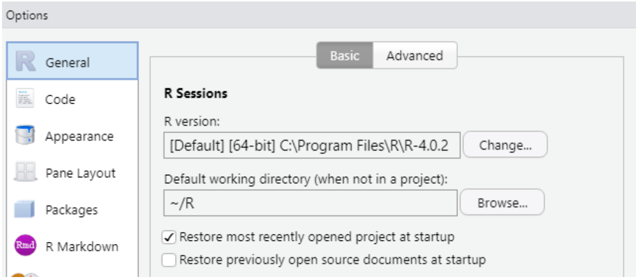
- What data do I need?
  - Categorical variables
  - Result variables
    - Numeric
    - Other Data Types
- How much data do I need?
  - Number of cases (**n**)
  - Statistical power to achieve significance

Section 5

Projects in RStudio

Active Directories

- When you start RStudio
  - Uses the default directory from your "Tools/Global Options" setting



R Sessions

R version:
[Default] [64-bit] C:\Program Files\R\R-4.0.2

Default working directory (when not in a project):
~/R

✓ Restore most recently opened project at startup
☐ Restore previously open source documents at startup

- If you want a different directory for an analysis, need to use `setwd()` to direct R there

setwd() and getwd()

- To set a new working directory, need to use command `setwd()`
  - With full path name in quotes as argument
- Example - for my thesis research in HIVAIDS
  -

    ```
    setwd(C:/Users/james/OneDrive/Documents/HIVAIDS/Drug_
    ```
- Complementary function `getwd()`
  - No argument
  - Shows what is current active directory

```
## [1] "C:/Users/james/OneDrive/Documents/MAD/MAD-Infecto
```

Why Do We Need *Projects*?

- You will work on different projects that have
  - Different databases
  - Different scripts
  - Different documents
- Throwing all your files from all projects in 1 R directory will make you crazy
  - Many projects could have upwards of 500 files
- Planning project workflow implies organization
- Build an R data analysis project around a reserved workspace

RStudio Projects Allow Group Members to Work Together Better

- Everyone has a different directory structure on their machines
  - I have projects in my R directory
  - All my HIV projects are in my HIVAIDS directory
  - All the versions of this course are in my MAD directory
- What is your directory structure?
- Projects create their own working directory (active directory)
  - Same for everyone who uses the same project structure and file
  - No more constantly having to type km-long file paths
    - And getting them wrong

If the first line of any of your R scripts is

```
setwd("C:\Users\jenny\path\that\only\I\have")
```

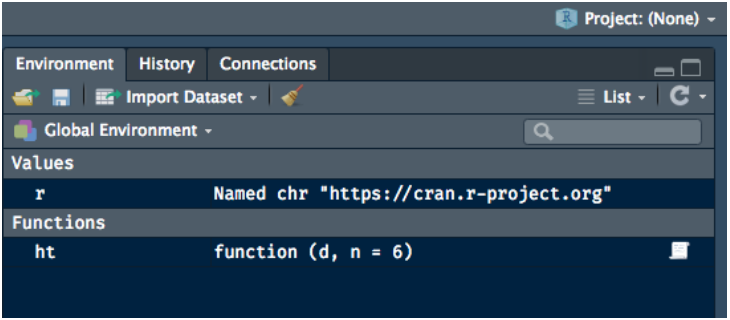
*Someone from the Tidyverse team will come to your office and **set your computer on fire***

They have promised

The best way to avoid such a consequence is create a new RStudio project for each new activity

How to Create a RStudio Project

- Find “Project:(None)” text
  - Top right of screen above Environment, etc. tab

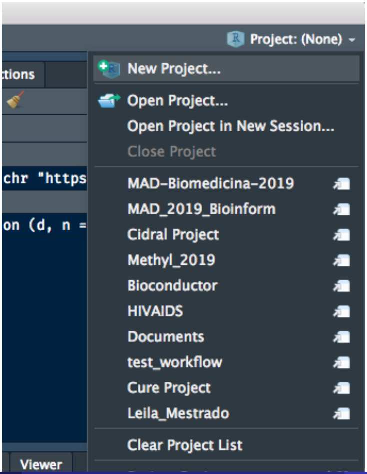


```
Values
  r          Named chr "https://cran.r-project.org"
Functions
  ht         function (d, n = 6)
```

- So far, no active project

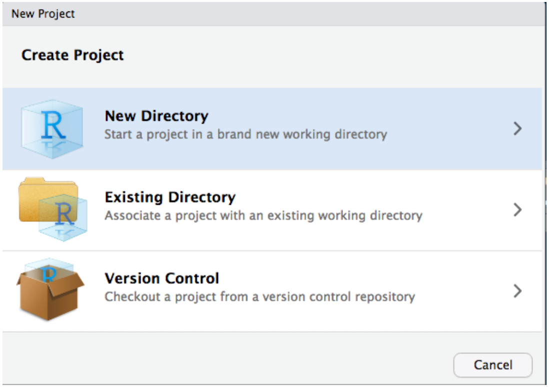
Create New Project

- Click on arrow to right of "(None)"
- Drop down menu will open
- Click "New Project . . . " option



Create Project Window

- Options to choose a new or existing directory
- Version Control option more advanced
- For this case, choose “New Directory”


New Project

Create Project

New Directory
Start a project in a brand new working directory

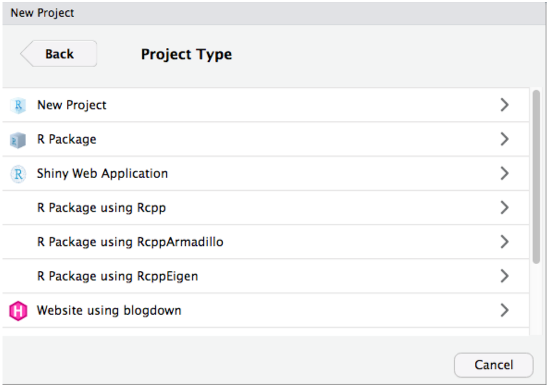
Existing Directory
Associate a project with an existing working directory

Version Control
Checkout a project from a version control repository

Cancel

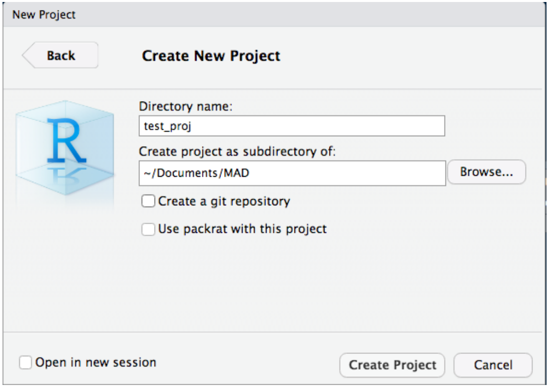
Types of New Projects

- RStudio has many types of new projects
- We want just a “New Project”: click on that



Final Creation Screen

- Give the project a name and a location
- Click `Create Project` button
- For now, ignore `git` and `packrat` boxes



Directory name: test_proj

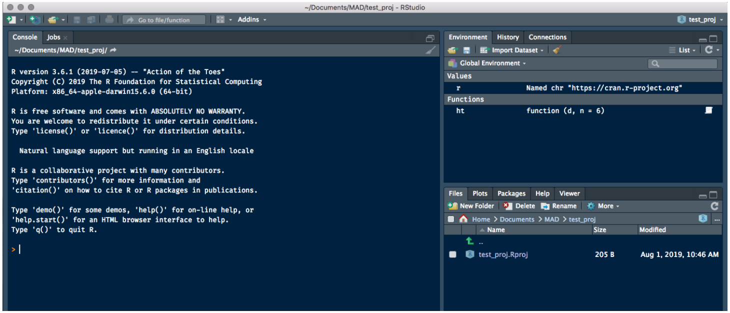
Create project as subdirectory of: ~/Documents/MAD

☐ Create a git repository

☐ Use packrat with this project

☐ Open in new session

Here Is Your Project



```
R version 3.6.1 (2019-07-05) -- "Action of the Toes"
Copyright (C) 2019 The R Foundation for Statistical Computing
Platform: x86_64-apple-darwin15.6.0 (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.

> |
```

```
Values
  r           Named chr "https://cran.r-project.org"
Functions
  ht          function (d, n = 6)
```

Name	Size	Modified
..		
test_proj.Rproj	205 B	Aug 1, 2019, 10:46 AM

Subdirectories

- Projects can become large, VERY LARGE
  - Thesis project: 1,978 files, 383 sub-directories, 425 MB
- Sub-directories for files of different types.
  - Like setting up a project
  - Helps keep files organized
- My preferred subdirectory structure for a new project
  - Data
  - Raw Data
  - Docs
  - Graphics
  - Programs/Scripts
  - Slides
- Keep *raw data* separate from any other data files
  - Cleaned, subsets, etc.
  - You may need it again!

Section 6

We're HERE!

Problem

- Various projects and various directories could have very different paths
- Path to `test_proj` on my old Mac Air:
  - "/Users/jameshunter/Documents/MAD/test_proj"
- Path to `test_proj` on my current Windows laptop:
  - "C:/Users/james/OneDrive/Documents/MAD/test_proj"
- If I wanted to use my path to go to a graphics subdirectory that both had
  - It would be a mess!

Solution: `here::here()`

- `here` package does one thing
  - Lets you know what is full path to working directory
- Simple form: no arguments

```r
here::here()
```

```
## [1] "C:/Users/james/OneDrive/Documents/MAD/MAD-Infecto-2020"
```

Go to graphics Subdirectory of Current Project

- Give the function the argument "graphics"
- Thereafter, can use variable gr to refer to full path of location

```
gr <- here::here("graphics")
gr
```

```
## [1] "C:/Users/james/OneDrive/Documents/MAD/MAD-Infecto-2020/graphics"
```

- Note new package::function notation
  - 2 colons call a function directly from a package without having to load package

Section 7

Scripts and Programming

Why Scripts? - A Review

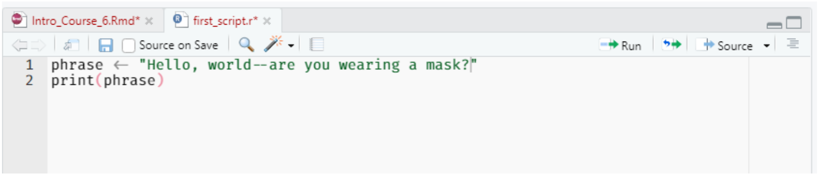
- Combine commands into coherent whole
- Store your commands
  - Reuse or recall at later date
- Facilitate correction of error and re-running program
- Scripts can be either
  - Separate documents (.r files)
  - Blocks of code in a larger .rmd document
    - ⋆ Like this one

Basic Scripts

- Open new script document (.r) with
  - CTRL-Shift-N
  - Plus icon in upper left of screen
- Editor is a basic text editor
- Example
  - Assign sentence: “Hello, world–wear a mask!” to the variable `phrase`
  - Print `phrase` to the screen

Program Text

```r
phrase <- "Hello, world--wear a mask!"
print(phrase)
```



```r
phrase ← "Hello, world--are you wearing a mask?"
print(phrase)
```

- Execute program with `Run` or `Source` buttons

Section 8

Loops

General

- Way to execute series of commands repeatedly
- Executes until encounters condition that ends it
- 2 Flavors
  - **for** loops
  - **while** loops

for Loops

- Format

```
for(var in vector) {
  code
  code
  code
}
```

- `for` starts loop
- Number of repetitions of loop inside the parentheses
  - Repetitions:
    - ★ `var` – variable name
    - ★ `vector` – values variable can assume
- {} - pair of curly braces
- Inside braces: lines of code you want to execute

Example `for` Loop

- Square whole numbers from 5 to 10
- Denote variable as `i`
- Vector
  - c(5, 6, 7, 8, 9, 10)
  - 5:10

```
for(i in 5:10){
  print(i^2)
}
```

```
## [1] 25
## [1] 36
## [1] 49
## [1] 64
## [1] 81
## [1] 100
```

What the Loop Did

- 1st Iteration
  - Assigned 5 to `i`
  - Squared `i` (`i^2`)
  - Wrote `i` to screen (25)
- 2nd Iteration
  - Assigned 6 to `i`
  - Squared `i`
  - Wrote `i` to screen (36)
- Same for 3rd to 6th
- At end of 6th iteration (i <- 10)
  - Could not find a new value to give to `i`
  - Stopped

More Realistic Example

- Take a DNA string and transform it to a vector of letters
- Length of the string unknown
- Can be calculated with `nchar()` function: gives number of characters

```r
library(tidyverse)
seq <- "CCTCAAATCACTCTTTGGCAACGACCCTTAGTCACAATAAAAGTAGGGGA"
seq_length <- nchar(seq)
seq_vector <- character(seq_length) # create vector to hold result
for (i in 1:nchar(seq)) {
  seq_vector[i] <- tolower(str_sub(seq, i, i))
}
seq_vector
```

```
##  [1] "c" "c" "t" "c" "a" "a" "a" "t" "c" "a" "c" "t" "c" "t" "t" "t" "g" "g" "c"
## [20] "a" "a" "c" "g" "a" "c" "c" "c" "t" "t" "a" "g" "t" "c" "a" "c" "a" "a" "t"
## [39] "a" "a" "a" "a" "g" "t" "a" "g" "g" "g" "g" "a"
```

`while` Loops

- Similar structure to `for` loops but work differently
- Keeps repeating until condition you put in `while` statement turns `FALSE`
- Consult text
- Used much less than `for`

Loops vs. *Vectorized* Statements

- R is vectorized
  - Interpreter will act on all elements of a vector at once
  - Don't need to loop item by item
- Should we use loops at all?
- Computer speed today reduces the speed advantage of vectorized statements
  - Except for VERY large datasets, loops are practically equivalent
  - Millions of cases, hundreds of variables
- Don't give up on loops
  - Logic is usually easier to program than many vectorized functions
  - Can save much overall time in terms of programming

Section 9

Conditional Statements (*if. . . then. . . else*)

Thought Problem

- We are reading sequences of nucleotides
  - DNA alphabet: ACGT
  - RNA alphabet: ACGU
- We could say
  - *If* the sequence is DNA *then* the alphabet is ACGT.
- More complete version
  - *If* the sequence is DNA *then* the alphabet is ACGT *else* the alphabet is ACGU.

If Statements in R

- Basic structure

```
if(condition) {
  code
  code
  code
  } else {
      code
      code
      code
}
```

Simple Example: No `else` Clause

```r
x <-  0

if (x == 0) {
  print("x equals 0")
}
```

```
## [1] "x equals 0"
```

More Complete Logical Test

```r
x <-  0

if (x == 6) {
  print("x equals 6")
} else {
    print("x does not equal 6")
  }
```

```
## [1] "x does not equal 6"
```

3 Nested Conditions – DNA/RNA Example

```r
seq_type <- "DNA"

if (seq_type == "DNA") {
  print("ACGT")
} else {
    if (seq_type == "RNA"){
      print("ACGU")
    } else {
      print("sequence neither DNA nor RNA")
    }
  }
```

```
## [1] "ACGT"
```

`ifelse()` Function

- For simple logical tests
- `ifelse(test, true, false)`
- 3 Arguments
  - `test`: logical test
  - `true`: result if TRUE
  - `false`: result if FALSE
- If result of test is TRUE, returns `true` value
- If result of test is FALSE, returns `false` value

ifelse() Example

```r
X <- 2
res <- ifelse(X > 10, "greater than", "less than or equal
paste("X is", res, "10")
```

```
## [1] "X is less than or equal to 10"
```

More Complicated if. . . then. . . else Logic

- Function in `dplyr` package: `case_when()`
- Can more comfortably handle large number of alternative cases
- More advanced

Section 10

Importing Files to R

Types We Will Work With

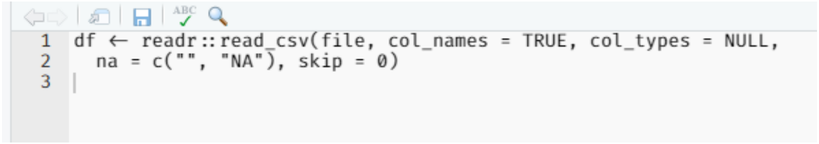
- Comma-Delimited Files (.csv)
- Excel Files (.xls or .xlsx)
- FASTA files

Section 11

Comma-Delimited Files (.csv)

Importing a .csv File

- A plain text file that is in rectangular form
- Commas separating the fields
- Interpretable by Excel
- Most common file type you will import
- Use `readr::read_csv()` to import it
  - Function reads file
  - Treats commas as Excel does: separators between columns
  - Imports it to a tibble



```r
df ← readr::read_csv(file, col_names = TRUE, col_types = NULL,
  na = c("", "NA"), skip = 0)
```

`read_csv()` Arguments

- `file` = file path to reach the file
  - Use of `here::here()`
- `col_names` = Is the first row the names of the columns or data?
  - `TRUE` means 1st row are the column names; `read_csv()` assumes so
  - `FALSE` means it has data rather than names

`col_types` = Argument

- `read_csv()` will try to guess the correct data type based on the column content
- If you want to let R guess, enter `NULL` or omit the argument
- If you want to specify column types, use a string of characters from list:
  - c = character
  - i = integer
  - n = number
  - d = double
  - l = logical
  - f = factor
  - D = date
  - T = date time
  - t = time
  - ? = guess
  - - to skip the column

Notes on `col_types =`

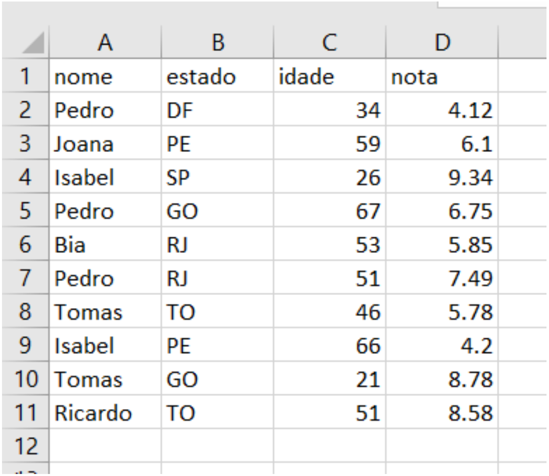
- Column string must have **exactly** the same number of characters as the data has columns
  - If you have a data set with five columns, the string must have five characters
    - Ex. `col_types = "cfn-c"`
    - Means character-factor-numeric-skip-character

Two Other Key Arguments

- Specifying missing data
  - `read_csv()` assumes that missing data will be encoded with
    - the string “NA” or a blank string
  - If data uses other codes, need to specify them
    - “99” is a common indicator of missing data in social science
    - Must specify the `na =` argument as `na = c("", "NA", "99")`
- Skip rows: `skip =`
  - Datasets with metadata, etc. in 1st few rows
  - To skip these, set a positive value for the `skip =` argument
    - Ex. `skip = 4` will start reading the database on the fifth row

Simple .csv Example

- `classe_simples.csv` records info on a small school class
  - Name, home state, age and scores on a standardized test (1 - 10)



	A	B	C	D
1	nome	estado	idade	nota
2	Pedro	DF	34	4.12
3	Joana	PE	59	6.1
4	Isabel	SP	26	9.34
5	Pedro	GO	67	6.75
6	Bia	RJ	53	5.85
7	Pedro	RJ	51	7.49
8	Tomas	TO	46	5.78
9	Isabel	PE	66	4.2
10	Tomas	GO	21	8.78
11	Ricardo	TO	51	8.58
12				

Importing `classe_simples.csv`

- Assign the function to a name for the data frame `classe1`
- Run a simplified command with no column spec
  - R will guess

```
classe1 <- read_csv(here::here("classe_simples.csv"))
classe1
```

Result

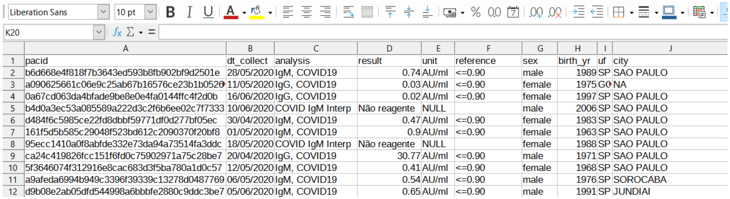
```
## Parsed with column specification:
## cols(
##   nome = col_character(),
##   estado = col_character(),
##   idade = col_double(),
##   nota = col_double()
## )

## # A tibble: 10 x 4
##    nome    estado idade  nota
##    <chr>   <chr>  <dbl> <dbl>
##  1 Pedro   DF        34  4.12
##  2 Joana   PE        59  6.1 
##  3 Isabel  SP        26  9.34
##  4 Pedro   GO        67  6.75
##  5 Bia     RJ        53  5.85
##  6 Pedro   RJ        51  7.49
##  7 Tomas   TO        46  5.78
##  8 Isabel  PE        66  4.2 
##  9 Tomas   GO        21  8.78
## 10 Ricardo TO        51  8.58
```

`classe_simples.csv` Results

- First line contained variable names; R used them
- If we had wanted to specify columns, we could have used
  - `col_types = "ccnn"`

More Complex Data Set

- Example we will use throughout data cleaning process
- 99 case version of Einstein Hospital COVID-19 sorological tests
  - `einstein_soro_tests.csv`
- First look at "messy", ie. real data



K20

	A	B	C	D	E	F	G	H	I	J
1	pacid	dt_collect	analysis	result	unit	reference	sex	birth_yr	uf	city
2	b6d668e4f818f7b3643ed593b8fb902bf9d2501e	28/05/2020	IgM, COVID19	0.74	AU/ml	<=0.90	male	1989	SP	SAO PAULO
3	a090625661c06e9c25ab67b16576ce23b1b0526	11/05/2020	IgG, COVID19	0.03	AU/ml	<=0.90	female	1975	GO	NA
4	0a67cd063da4bfade9be8e0e4fa0144ffc4f2d0b	16/06/2020	IgG, COVID19	0.02	AU/ml	<=0.90	female	1997	SP	SAO PAULO
5	b4d0a3ec53a085589a222d3c2f6b6ee02c7f7333	10/06/2020	COVID IgM Interp	Não reagente	NULL		male	2006	SP	SAO PAULO
6	d484f6c5985ce22fd8dbbf59771df0d277bf05ec	30/04/2020	IgM, COVID19	0.47	AU/ml	<=0.90	female	1983	SP	SAO PAULO
7	161f5d5b585c29048f523bd612c2090370f20bf8	01/05/2020	IgM, COVID19	0.9	AU/ml	<=0.90	female	1963	SP	SAO PAULO
8	95ecc1410a0f8abfde332e73da94a73514fa3ddc	18/05/2020	COVID IgM Interp	Não reagente	NULL		female	1988	SP	SAO PAULO
9	ca24c419826fcc151f6fd0c75902971a75c28be7	20/04/2020	IgG, COVID19	30.77	AU/ml	<=0.90	male	1971	SP	SAO PAULO
10	5f3646074f312916e8cac683d3f5ba780a1d0c57	12/05/2020	IgM, COVID19	0.41	AU/ml	<=0.90	female	1968	SP	SAO PAULO
11	a9afeda6994b949c3396f39339c13278d0487769	06/05/2020	IgM, COVID19	0.54	AU/ml	<=0.90	male	1976	SP	SOROCABA
12	d9b08e2ab05dfd544998a6bbbfe2880c9ddc3be7	05/06/2020	IgM, COVID19	0.65	AU/ml	<=0.90	male	1991	SP	JUNDIAI

Constructing the Import Command

- Variable names in first line
  - `col_names =` can be omitted
- No need to use `skip =`
- `NA` argument
  - `result` and `unit` variables use “Não reagente” and “NULL”
  - `birth_yr` has 3 `NA` values
  - Unclear if they mean 0 or could not get a meaningful result
  - Create argument: `na = c("Não reagente", "NULL", "NA")`

`col_types =` Argument

- `pacid`: awkward, but obviously a string
- `dt_collect`: Date, but in non-standard format
  - R will parse as `character` type
  - We can reformat it in tidying phase
- `analysis`: character string
- `result`: should be numeric, but has text entries
  - R will parse as `character` type
- `unit` & `reference` fields: character strings
  - `reference` could be numeric, but leave to tidying
- `sex`: `character`, but better as `factor`: only 2 values
- `birth_yr`: `numeric`
- `uf` & `city`: `character`
- Using codes, full argument: `col_types = "cccccfncc"`

Ready to Import

```r
einstein_soro <- read_csv(here::here("einstein_soro_tests.csv"),
                          col_types = "cccccfncc",
                          na = c("Não reagente", "NULL", "NA"))
glimpse(einstein_soro)
```

```
## Rows: 99
## Columns: 10
## $ pacid      <chr> "b6d668e4f818f7b3643ed593b8fb902bf9d2501e", "a090625661c...
## $ dt_collect <chr> "28/05/2020", "11/05/2020", "16/06/2020", "10/06/2020", ...
## $ analysis   <chr> "IgM, COVID19", "IgG, COVID19", "IgG, COVID19", "COVID I...
## $ result     <chr> "0.74", "0.03", "0.02", "Não reagente", "0.47", "0.9", "...
## $ unit       <chr> "AU/ml", "AU/ml", "AU/ml", NA, "AU/ml", "AU/ml", NA, "AU...
## $ reference  <chr> "<=0.90", "<=0.90", "<=0.90", "", "<=0.90", "<=0.90", ""...
## $ sex        <fct> male, female, female, male, female, female, female, male...
## $ birth_yr   <dbl> 1989, 1975, 1997, 2006, 1983, 1963, 1988, 1971, 1968, 19...
## $ uf         <chr> "SP", "GO", "SP", "SP", "SP", "SP", "SP", "SP", "SP", "S...
## $ city       <chr> "SAO PAULO", NA, "SAO PAULO", "SAO PAULO", "SAO PAULO", ...
```

Section 12

Working with Excel Files

Advice on Working *within* Excel - 1

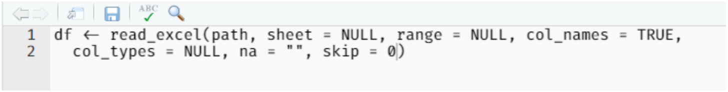
- If you are preparing *.csv* files,
    - Save different types of data in different files
- If you are working with *.xlsx* files,
    - Save different types of data in different sheets (tabs)
- Data should be placed in a block of data
    - Without any blank lines
- First line should have variable names, if you are using them
- Each column should contain **only** one class of data
    - Numeric, character, logical, . . .

Advice on Working *within* Excel - 2

- Zeros are always “0”
  - Never “-”, " " (blank space) or other character
- Missing data should always be “NA”
  - Never “0”, “99” or other text
- Start making the data conform to the tidy data rules here
  - Each column should be a **variable**
  - Each line should be a **case**
- **Never** use colors or design elements

Importing Excel Files

- Do not have to save files as ".csv" first
- `readxl` package
  - `read_excel()`
  - Works with both .xls and .xlsx formats
- Differs from `read_csv()` primarily in handling of column types
- Also has arguments for
  - Specifying tabs within workbook
  - Specifying specific region on a given sheet



```
df ← read_excel(path, sheet = NULL, range = NULL, col_names = TRUE,
  col_types = NULL, na = "", skip = 0)
```

Arguments for `read_excel()`

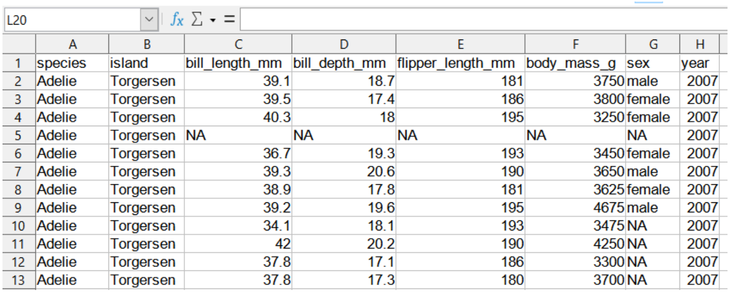
- Specifying what to download
  - Path for file
  - Sheet name
  - `range` in normal Excel format: "A1:B25"
- `col_names` = functions same as `read_csv()`
- `na` = and `skip` = function the same

Column Types in `read_excel()`

- Uses whole words instead of single letters
- If you want to make a factor, need to do so in tidying process
- Column types:
    - date
    - guess (trust R to make a good choice)
    - list
    - logical
    - numeric
    - skip
    - text

Spreadsheet Example

- R standard data set: Palmer Penguins
  - 8 characteristics of 344 penguins
  - `penguins.xlsx`



	A	B	C	D	E	F	G	H
1	species	island	bill_length_mm	bill_depth_mm	flipper_length_mm	body_mass_g	sex	year
2	Adelie	Torgersen	39.1	18.7	181	3750	male	2007
3	Adelie	Torgersen	39.5	17.4	186	3800	female	2007
4	Adelie	Torgersen	40.3	18	195	3250	female	2007
5	Adelie	Torgersen	NA	NA	NA	NA	NA	2007
6	Adelie	Torgersen	36.7	19.3	193	3450	female	2007
7	Adelie	Torgersen	39.3	20.6	190	3650	male	2007
8	Adelie	Torgersen	38.9	17.8	181	3625	female	2007
9	Adelie	Torgersen	39.2	19.6	195	4675	male	2007
10	Adelie	Torgersen	34.1	18.1	193	3475	NA	2007
11	Adelie	Torgersen	42	20.2	190	4250	NA	2007
12	Adelie	Torgersen	37.8	17.1	186	3300	NA	2007
13	Adelie	Torgersen	37.8	17.3	180	3700	NA	2007

Example Arguments

- Column types
  - Argument is unnecessary as R will parse values correctly
  - If we want to specify it
    - `col_types = c("text", "text", "numeric", "numeric", "numeric", "numeric", "text", "numeric)`
  - `NA` needed for measurement columns since missings exist
    - `na = "NA"`

Import File

```r
penguin <- readxl::read_excel(here::here("penguins.xlsx"), na = "NA")
glimpse(penguin)
```

```
## Rows: 344
## Columns: 8
## $ species           <chr> "Adelie", "Adelie", "Adelie", "Adelie", "Adelie",...
## $ island            <chr> "Torgersen", "Torgersen", "Torgersen", "Torgersen...
## $ bill_length_mm    <dbl> 39.1, 39.5, 40.3, NA, 36.7, 39.3, 38.9, 39.2, 34....
## $ bill_depth_mm     <dbl> 18.7, 17.4, 18.0, NA, 19.3, 20.6, 17.8, 19.6, 18....
## $ flipper_length_mm <dbl> 181, 186, 195, NA, 193, 190, 181, 195, 193, 190, ...
## $ body_mass_g       <dbl> 3750, 3800, 3250, NA, 3450, 3650, 3625, 4675, 347...
## $ sex               <chr> "male", "female", "female", NA, "female", "male",...
## $ year              <dbl> 2007, 2007, 2007, 2007, 2007, 2007, 2007, 2007, 2...
```

Section 13

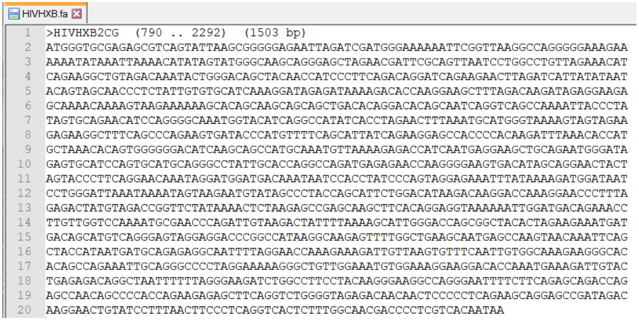
FASTA Files

Read a FASTA File

- Most common storage format for nucleotide (DNA or RNA) and amino acid sequences
- Text-based representation using single-letter codes (alphabets)
- Each sequence can be preceded by a line of sequence name and comments
- Like a .csv file: pure text (uses text editor)
- To prepare sequences for analysis
  - Use specialized programs for reading and parsing
- New package (`bioseq`) provides easy and complete functions
  - Not in original set of downloaded packages
  - Install on your machine with `install.packages("bioseq")` in Console
- `bioseq::read_fasta()` takes 2 arguments
  - file name and path
  - Type of sequence: "DNA" (default), "RNA" or "AA"

FASTA File Example

- DNA sequences of the *gag* polyprotein of the HIV-1 virus
  - Reference genome HXB-2
- Sequence stored in file `HIVHXB.fa`
- Sequence is 1,503 base pairs long.



```
>HIVHXB2CG  (790 .. 2292)  (1503 bp)
ATGGGTGCGAGAGCGTCAGTATTAAGCGGGGGAGAATTAGATCGATGGGAAAAAATTCGGTTAAGGCCAGGGGGAAAGAA
AAAATATAAATTAAAACATATAGTATGGGCAAGCAGGGAGCTAGAACGATTCGCAGTTAATCCTGGCCTGTTAGAAACAT
CAGAAGGCTGTAGACAAATACTGGGACAGCTACAACCATCCCTTCAGACAGGATCAGAAGAACTTAGATCATTATATAAT
ACAGTAGCAACCCTCTATTGTGTGCATCAAAGGATAGAGATAAAAGACACCAAGGAAGCTTTAGACAAGATAGAGGAAGA
GCAAAACAAAAGTAAGAAAAAAGCACAGCAAGCAGCAGCTGACACAGGACACAGCAATCAGGTCAGCCAAAATTACCCTA
TAGTGCAGAACATCCAGGGGCAAATGGTACATCAGGCCATATCACCTAGAACTTTAAATGCATGGGTAAAAGTAGTAGAA
GAGAAGGCTTTCAGCCCAGAAGTGATACCCATGTTTTCAGCATTATCAGAAGGAGCCACCCCACAAGATTTAAACACCAT
GCTAAACACAGTGGGGGGACATCAAGCAGCCATGCAAATGTTAAAAGAGACCATCAATGAGGAAGCTGCAGAATGGGATA
GAGTGCATCCAGTGCATGCAGGGCCTATTGCACCAGGCCAGATGAGAGAACCAAGGGGAAGTGACATAGCAGGAACTACT
AGTACCCTTCAGGAACAAATAGGATGGATGACAAATAATCCACCTATCCCAGTAGGAGAAATTTATAAAAGATGGATAAT
CCTGGGATTAAATAAAATAGTAAGAATGTATAGCCCTACCAGCATTCTGGACATAAGACAAGGACCAAAGGAACCCTTTA
GAGACTATGTAGACCGGTTCTATAAAACTCTAAGAGCCGAGCAAGCTTCACAGGAGGTAAAAAATTGGATGACAGAAACC
TTGTTGGTCCAAAATGCGAACCCAGATTGTAAGACTATTTTAAAAGCATTGGGACCAGCGGCTACACTAGAAGAAATGAT
GACAGCATGTCAGGGAGTAGGAGGACCCGGCCATAAGGCAAGAGTTTTGGCTGAAGCAATGAGCCAAGTAACAAATTCAG
CTACCATAATGATGCAGAGAGGCAATTTTAGGAACCAAAGAAAGATTGTTAAGTGTTTCAATTGTGGCAAAGAAGGGCAC
ACAGCCAGAAATTGCAGGGCCCCTAGGAAAAAGGGCTGTTGGAAATGTGGAAAGGAAGGACACCAAATGAAAGATTGTAC
TGAGAGACAGGCTAATTTTTTAGGGAAGATCTGGCCTTCCTACAAGGGAAGGCCAGGGAATTTTCTTCAGAGCAGACCAG
AGCCAACAGCCCCACCAGAAGAGAGCTTCAGGTCTGGGGTAGAGACAACAACTCCCCCTCAGAAGCAGGAGCCGATAGAC
AAGGAACTGTATCCTTTAACTTCCCTCAGGTCACTCTTTGGCAACGACCCCTCGTCACAATAA
```

Import the File

```r
seq <- bioseq::read_fasta(here::here("HIVHXB.fa"), "DNA")
stringr::str_sub(seq, 1, 60)
```

```
## [1] "ATGGGTGCGAGAGCGTCAGTATTAAGCGGGGGAGAATTAGATCGATGGGAAAAAATTCGG"
```

- Result a character string vector
- First 60 nucleotides using the subsetting function of `stringr` package.

Section 14

Go to Presentation Class_2b