CWRU DSCI351-351m-451: Rmd and For Loop Basics, (CWRU, Pitt, UCF, UTRGV)

Profs: R. H. French, L. S. Bruckman, P. Leu, K. Davis, S. Cirlos

TAs: W. Oltjen, K. Hernandez, M. Li, M. Li, D. Colvin

16 September, 2022

Contents

3.2.2.1 Reading, Homeworks, Projects, SemProjects

- Readings:
  - For Today: OIS Chapter 4
  - For Next Class: EDA 1-31
- Lab Exercises:
  - LE2 given our Tuesday Sept. 13th
  - LE2 is due Thursday Sept. 22nd
- Office Hours: (Class Canvas Calendar for Zoom Link)
  - Wednesday @ 4:00 PM to 5:00 PM, Will Oltjen
  - Saturday @ 3:00 PM to 4:00 PM, Kristen Hernandez
  - **Office Hours are on Zoom, and recorded**
- Semester Projects
  - DSCI 451 Students Biweekly Update 1 Due
  - DSCI 451 Students
    - Next **Report Out #1 is Due Friday September 30th**
  - All DSCI 351/351M/451 Students:
    - **Peer Grading of Report Out #1 is Due October 11th, 2022**
  - Exams
    - MidTerm: Tuesday October 18th, in class or remote, 11:30 - 12:45 PM
    - Final: Monday December 19, 2022, 12:00PM - 3:00PM, Nord 356 or remote

3.2.2.2 Textbooks

- Peng: R Programming for Data Science
- Peng: Exploratory Data Analysis with R
- Open Intro Stats, v3
- Wickham: R for Data Science
- Hastie: Intro to Statistical Learning with R

3.2.2.3 Syllabus

3.2.2.4 R Intro 2

```
#====================
# A pound symbol, or "Hashtag" indicates a comment in the code

# Basics of R
# Math operations
2 + 2
```

3.2.2.4.1 Basics of R

```
## [1] 4
```

```
8 / 4
```

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

```
2 * 3
```

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

Day:Date	Foundation	Practicum	Reading	Due
w01a:Tu:8/30/22	ODS Tool Chain	R, Rstudio, Git		
w01b:Th:9/1/22	Setup ODS Tool Chain	Bash, Git, Slack, Agile	PRP4-33	LE1
w02a:Tu:9/6/22	Bash-Git-Knuth-Lit.Prog.	RIntroR	PRP35-64	
w02b:Th:9/8/22	What is Data Science	OIS:Intro2R	OIS1,2	
w02Pr:Fr:9/9/22			PRP65-93	**451 Update1**
w03a:Tu:9/13/22	Data Intro	Data Analytic Style	PRP94-116	LE2 **LE1 Due**
w03b:Th:9/15/22	Rand. Var. Normal Dist.	Git, Rmds, Loops	OIS4	
w04a:Tu:9/20/22	Tidy Check Explore	Tidy GapMinder	EDA1-31	
w04b:Th:9/22/22	Inference, DSCI Process	Other Distrib. 7 ways	R4DS1-3	LE3 **LE2 Due**
w04Pr:Fr:9/23/22			EDA32-58	**451 Update2**
w05a:Tu:9/27/22	OIS4 Rand. Var.	EDA of PET Degr.	OIS5	
w05b:Th:9/29/22	OIS5 Found. of Infer.	Multivar Corr. Plot	R4DS4-6	
w05Pr:Fr:9/30/22				**451 RepOut1**
w06a:Tu:10/4/22	Pred., Algorithm, Model	Anscombe's Quartets	R4DS7-8	
w06b:Th:10/6/22	EDA stats, vis	Summ. Stats & Vis.	R4DS9-16	LE4 **LE3 Due**
w06Pr:Fr:10/7/22	Corr. Coeff. Pairs Plots			**451 Update3**
w07a:Tu:10/11/22	Confidence Intervals	Penguins	OIS6.1-2	**PeerRv1 Due**
w07b:Th:10/13/22	Midterm Rev.	Hypo.Test, Sampl. Dist.		
w08a:Tu:10/18/22	**MIDTERM**	**EXAM**		
w08b:Th:10/20/22	Programming & Coding	Coding Expect.		**LE4 Due**
w08Pr:Fr:10/21/22				**451 Update4**
Tu:10/24,25	**CWRU**	**FALL BREAK**	R4DS17-21	
w09b:Th:10/27/22	Cat. Inf. 1 & 2 propor.	Indep. Test,2-way tables	OIS6.3-4	LE5
w09Pr:Fr:10/28/22				**451 RepOut2**
w10a:Tu:11/1/22	Goodness of Fit, χ^2 test	t-tests 1&2 means	OIS7.1-4	
w10b:Th:11/3/22	Num. Infer, Cont. Tables	Stat. Power		
w10Pr:Fr:11/4/22				**451 Update5**
w11a:Tu:11/8/22	Sample & Effect Size	Stat. Power GGmap	OIS8	**PeerRv2 Due**
w11b:Th:11/10/22	Inf. 4 Regr, Test & Train	Curse of Dimen.	ISLR1,2.1,2	LE6 **LE5 Due**
w12a:Tu:11/15/22	Lin. Regr. Part 1	Residuals	OIS9	
w12b:Th:11/17/22	Lin. Regr. Part 2	Regr. Diagnostics		
w12Pr:Fr:11/18/22				**451 Update6**
w13a:Tu:11/22/22	Mult. Lin. Regr.	Var. & Mod. Selec.,	ISLR3.1	LE7 **LE6 due**
w13b:Th:11/24/22	Log. Regr.	GIS Trends	ISLR3.2	
w13Pr:Fr:11/25/22				**451 RepOut3**
w14a:Tu:11/23/22	Classificat., Sup. Lrning	Caret, Broom 4 modeling	ISLR4.1-3	
Th,Fr:11/24,25	**THANKSGIVIING**	**Vacation**		
w15a:Tu:11/29/22		Clustering		**PeerRv3 Due**
w15b:Th:12/1/22	Big Data Analytics	Dist. Comp., Hadoop		
w15SPr:Fr:12/2/22		Read Article by	Mirletz,2015	
w16a:Tu:12/6/22	Final Exam Review			
w15b:Th:12/8/22				**LE7 due**
Friday 12/12	**SemProj**	**Final Report**		**SemProj4 due**
Monday 12/19	**FINAL EXAM**	**12:00-3:00pm**	Nord 356	or remote

Figure 1: DSCI351-351M-451 Syllabus

```
3 ^ 3
```

```
## [1] 27
```

```
a <- 3
b <- 3
a + b
```

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

```
#====================
# Assignment operator is usually used instead of =
# It is directional
# = works in most cases too but may cause problems
# So we always use the <- Assignment operatore

a <- 3
b <- 6

a <- b
a
```

3.2.2.4.2 Assignment Operator

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

```
a <- 3
b <- 6

a -> b
a
```

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

```
#====================
# Object classes
# numerics
a <- 5.5
class(a)
```

3.2.2.4.3 Object Classes

```
## [1] "numeric"
```

```
# integers
b <- as.integer(42)
class(b)
```

```
## [1] "integer"
```

```
# logicals
c <- TRUE
class(c)
```

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

```r
# characters
d <- "hello world"
class(d)
```

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

```r
d
```

```
## [1] "hello world"
```

```r
# factors
e <- as.factor(c("1", "1", "a", "1", "c", "a"))
class(e)
```

```
## [1] "factor"
```

```r
e
```

```
## [1] 1 1 a 1 c a
## Levels: 1 a c
```

```r
## quick note it you want to convert a factor to a numeric
## You have to convert it to a character first, then a numeric


#====================
# Assigining Operators

## PRO TIP ## - Control enter runs the line you are on or a highlighted section

# Assign number
x <- 5

# Lets view it with function View()
View(x)
```

3.2.2.4.4 Assigning Operators

```
## Error in .External2(C_dataviewer, x, title): unable to start data viewer
```

```r
# We can also look into what view is as a function and how it works
?View() # The question command shows us the help file

# Assign a vector or array of numbers, from 1 - 5
x <- 1:5

# We can also use = instead of <- (supposeduly <- is better than =, not sure why)
# Assign a character string
B <- "hello"

# Use print function to print to console
print(B)
```

```
## [1] "hello"
```

```r
print(x)
```

```
## [1] 1 2 3 4 5
```

```r
#====================
# R Object and type of classes
# Characters, integers, numeric, complex, Logic (TRUE/FALSE)

# R Vectors
a <- c("hello", "wolrd")
A <- c("hi", 5)
c <- c(1.23452, 2.1435, 3.14)

# The function c() here is "concatenate" so it will combine the values
false <- TRUE

# These will have attributes such as dimensions, object class, and length

# Lets check all of the attributes
class(a)
```

3.2.2.4.5 Objects and Class Types

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

```r
class(A)
```

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

```r
class(x)
```

```
## [1] "integer"
```

```r
class(c)
```

```
## [1] "numeric"
```

```r
class(false)
```

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

```r
false
```

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

```r
# There are also factors - but we will dive into those later, they are categorical
# Oh and lists... probably the most dynamic object in R

# These classes are default by what R interprets them to be, but we can change them
c <- as.character(c)
class(c)
```

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

```r
# Other class changing functions
as.numeric(x)
```

```
## [1] 1 2 3 4 5
```

```r
as.integer(x)
```

```
## [1] 1 2 3 4 5
```

```r
as.POSIXct(1442866615, origin = "1970-01-01") # A Date Format
```

```
## [1] "2015-09-21 16:16:55 EDT"
```

```
as.factor(x)
```

```
## [1] 1 2 3 4 5
## Levels: 1 2 3 4 5
```

```
as.list(x)
```

```
## [[1]]
## [1] 1
##
## [[2]]
## [1] 2
##
## [[3]]
## [1] 3
##
## [[4]]
## [1] 4
##
## [[5]]
## [1] 5
```

```
y <- as.data.frame(x) # An incredibly useful object
```

```
#====================
# Lets check the lengths of the variables

length(a)
```

3.2.2.4.6 Checking the length of variables

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

```
length(A)
```

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

```
length(x)
```

```
## [1] 5
```

```
length(c)
```

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

```
length(false)
```

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

```
# Other nice functions to create objects, seq(), matrix(), vector(), array()
m <- seq(1, 24, by = 3)
m
```

```
## [1]  1  4  7 10 13 16 19 22
```

```
n <- matrix(0, nrow = 3, ncol = 3)
print(n)
```

```
##      [,1] [,2] [,3]
## [1,]    0    0    0
## [2,]    0    0    0
## [3,]    0    0    0
```

```
#====================
# Lets look at the dimensions
dim(n)
```

3.2.2.4.7 Looking at Dimensions

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

```
# Lets also reset some of the variables
# The format for pulling values is [row, column]
n[2, 2] <- 2
print(n)
```

```
##      [,1] [,2] [,3]
## [1,]    0    0    0
## [2,]    0    2    0
## [3,]    0    0    0
```

```
# We can be even more efficient, say all of column 1 is 3
n[, 1] <- 3
print(n)
```

```
##      [,1] [,2] [,3]
## [1,]    3    0    0
## [2,]    3    2    0
## [3,]    3    0    0
```

```
# Next lets turn the matrix into a dataframe - this will allow multiple classes
n = as.data.frame(n)
n[1, 2] <- "character"
n[, 3] <- TRUE
print(n)
```

```
##   V1        V2   V3
## 1  3 character TRUE
## 2  3         2 TRUE
## 3  3         0 TRUE
```

```
class(n)
```

```
## [1] "data.frame"
```

```
class(n[, 1])
```

```
## [1] "numeric"
```

```
class(n[, 2])
```

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

```
class(n[, 3])
```

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

```
#====================
# Another useful tool is to bind datasets with rbind() and cbind()
a <- 1:5
b <- 20:24
rbind(a, b)
```

3.2.2.4.8 Binding rows and columns in a dataframe

```
##   [,1] [,2] [,3] [,4] [,5]
## a    1    2    3    4    5
## b   20   21   22   23   24
```

```
cbind(a, b)
```

```
##      a  b
## [1,] 1 20
## [2,] 2 21
## [3,] 3 22
## [4,] 4 23
## [5,] 5 24
```

```
d <- cbind(a, b)

# These must be of equal length along the dimension you wish to bind
# Same rows/same columns
```

```
#====================
# Other useful functions to mention, which(), mean(), sd(), min(), max()

row <- which(d[, 1] == 2) # Note the two equal signs that are necessary to check for equivalence
print(row)
```

3.2.2.4.9 Other Useful Functions

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

```
mean(d)
```

```
## [1] 12.5
```

```
sd(d)
```

```
## [1] 10.12423
```

```
min(d)
```

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

```
max(d)
```

```
## [1] 24
```

```
#====================
# For and IF statements, basic structure and example
# structure
for (i in 1:5) {
```

```
  # How much it should loop, and the values
  # Input what you want it to do
}

if (i == 5) {
  # Let it know what to check for
  # What you want to do if condition is met
}
```

3.2.2.4.10 For and If statements

```
## NULL
```

```
# Example
length <- length(d[, 1])

for (i in 1:length) {
  d[i, 1] = 100

  if (i == 2) {
    d[i, 1] = 200
  }

}

print(d)
```

```
##        a  b
## [1,] 100 20
## [2,] 200 21
## [3,] 100 22
## [4,] 100 23
## [5,] 100 24
```

```
#====================
# Writing and Reading data files

# Working directories
# Lets find where you are
getwd()
```

3.2.2.4.11 Writing and Reading data files

```
## [1] "/mnt/rstor/CSE_MSE_RXF131/cradle-members/sdle/lsh41/22f-dsci351-451-prof/2-class"
```

```
# Now we can set the directory we want, you must have R point to the directory
# that you either want to save to or read from
#setwd("path")

# PRO TIP the path . gives you the current, and .. gives you the one before
# Say I need to go two folders back and up one to data, I would enter
# setwd("../../data)
# Or if data was the next folder in from where I was setwd("./data)

# Set the working directory where you want
```

```r
# setwd("./2-class")


# Lets write a .csv
write.csv(d, "d.csv")

# Now lets read it back in
e = read.csv("d.csv")

# Notice there are now rownames in the object
# You can avoid this with

write.csv(d, "d.csv", row.names = FALSE)
e = read.csv("./d.csv")

# Other ways to read in data
read.table("./d.csv", sep = ",")
```

```
##    V1 V2
## 1   a  b
## 2 100 20
## 3 200 21
## 4 100 22
## 5 100 23
## 6 100 24
```

```r
#  read.delim("blah.txt", header = TRUE, sep = "\t") # For tab delimited files
```

```r
#====================
# Packages
# One of the coolest things in R is that it is open sourced and you can
# download new analysis techniques from many authors on the fly and incorporate
# it into your work

# There are two steps, install and library
#  install.packages("ggplot2")

library("ggplot2")

? ggplot2  # This allows us to check out the vignettes, or long form documentation
```

3.2.2.4.12 R Packages

```r
#====================
# R uses brackets to reference a data index
# data["row","column"]
# Standard organization for data set has variables as columns and observations as rows
# Keep in mind that R indexing starts from 1, not 0
# Load a test data set
data("iris")

iris[1, 2]
```

3.2.2.4.13 Using brackets to reference a data index

```
## [1] 3.5
```

```
# Leaving a row or column input blank puts all values
# First column
iris[, 1]
```

```
##   [1] 5.1 4.9 4.7 4.6 5.0 5.4 4.6 5.0 4.4 4.9 5.4 4.8 4.8 4.3 5.8 5.7 5.4 5.1
##  [19] 5.7 5.1 5.4 5.1 4.6 5.1 4.8 5.0 5.0 5.2 5.2 4.7 4.8 5.4 5.2 5.5 4.9 5.0
##  [37] 5.5 4.9 4.4 5.1 5.0 4.5 4.4 5.0 5.1 4.8 5.1 4.6 5.3 5.0 7.0 6.4 6.9 5.5
##  [55] 6.5 5.7 6.3 4.9 6.6 5.2 5.0 5.9 6.0 6.1 5.6 6.7 5.6 5.8 6.2 5.6 5.9 6.1
##  [73] 6.3 6.1 6.4 6.6 6.8 6.7 6.0 5.7 5.5 5.5 5.8 6.0 5.4 6.0 6.7 6.3 5.6 5.5
##  [91] 5.5 6.1 5.8 5.0 5.6 5.7 5.7 6.2 5.1 5.7 6.3 5.8 7.1 6.3 6.5 7.6 4.9 7.3
## [109] 6.7 7.2 6.5 6.4 6.8 5.7 5.8 6.4 6.5 7.7 7.7 6.0 6.9 5.6 7.7 6.3 6.7 7.2
## [127] 6.2 6.1 6.4 7.2 7.4 7.9 6.4 6.3 6.1 7.7 6.3 6.4 6.0 6.9 6.7 6.9 5.8 6.8
## [145] 6.7 6.7 6.3 6.5 6.2 5.9
```

```
# First row
iris[1, ]
```

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1          5.1         3.5          1.4         0.2  setosa
```

```
# Data frames have associated column names
colnames(iris)
```

```
## [1] "Sepal.Length" "Sepal.Width"  "Petal.Length" "Petal.Width"  "Species"
```

```
# Columns can be called by name using $
# Rstudio features tab completion for thing like column names
iris$Sepal.Length
```

```
##   [1] 5.1 4.9 4.7 4.6 5.0 5.4 4.6 5.0 4.4 4.9 5.4 4.8 4.8 4.3 5.8 5.7 5.4 5.1
##  [19] 5.7 5.1 5.4 5.1 4.6 5.1 4.8 5.0 5.0 5.2 5.2 4.7 4.8 5.4 5.2 5.5 4.9 5.0
##  [37] 5.5 4.9 4.4 5.1 5.0 4.5 4.4 5.0 5.1 4.8 5.1 4.6 5.3 5.0 7.0 6.4 6.9 5.5
##  [55] 6.5 5.7 6.3 4.9 6.6 5.2 5.0 5.9 6.0 6.1 5.6 6.7 5.6 5.8 6.2 5.6 5.9 6.1
##  [73] 6.3 6.1 6.4 6.6 6.8 6.7 6.0 5.7 5.5 5.5 5.8 6.0 5.4 6.0 6.7 6.3 5.6 5.5
##  [91] 5.5 6.1 5.8 5.0 5.6 5.7 5.7 6.2 5.1 5.7 6.3 5.8 7.1 6.3 6.5 7.6 4.9 7.3
## [109] 6.7 7.2 6.5 6.4 6.8 5.7 5.8 6.4 6.5 7.7 7.7 6.0 6.9 5.6 7.7 6.3 6.7 7.2
## [127] 6.2 6.1 6.4 7.2 7.4 7.9 6.4 6.3 6.1 7.7 6.3 6.4 6.0 6.9 6.7 6.9 5.8 6.8
## [145] 6.7 6.7 6.3 6.5 6.2 5.9
```

```
#====================
# Functions are processes that take an input and give an output
# Rstudio has tab completion for function inputs
max(iris$Petal.Length)
```

3.2.2.4.14 Functions

```
## [1] 6.9
```

```
mean(iris$Sepal.Width)
```

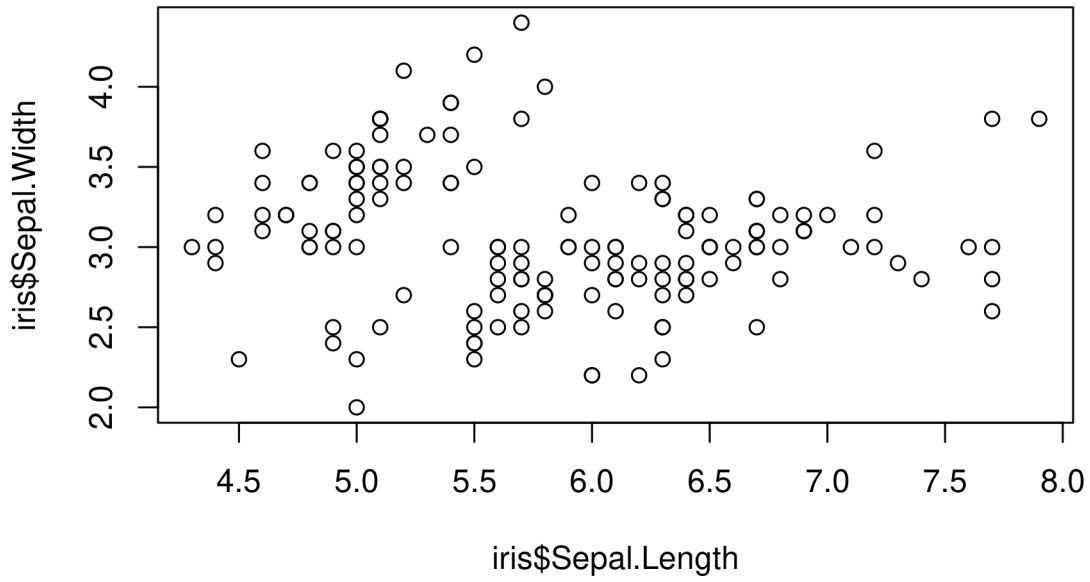
```
## [1] 3.057333
```

```
sd(iris$Petal.Width)
```

```
## [1] 0.7622377
```

```r
# Functions can take multiple inputs, they can be named in the call or placed in order
plot(x = iris$Sepal.Length, y = iris$Sepal.Width)

# x and y can be specified with x = ... in any order or the inputs can be given in order
# This plot is the same as the previous
plot(iris$Sepal.Length, iris$Sepal.Width)
```




```r
#====================
# Matrix operations
mat <- matrix(data = 1:9,
              nrow = 3,
              ncol = 3)
mat
```

3.2.2.4.15 Matrix Operations

```
##      [,1] [,2] [,3]
## [1,]    1    4    7
## [2,]    2    5    8
## [3,]    3    6    9
```

```r
# Element multiplication
mat * mat
```

```
##      [,1] [,2] [,3]
## [1,]    1   16   49
## [2,]    4   25   64
## [3,]    9   36   81
```

```r
# Matrix multiplication
mat %*% mat
```

```
##      [,1] [,2] [,3]
## [1,]   30   66  102
## [2,]   36   81  126
## [3,]   42   96  150
```

```r
# t() function is for transposing
t(mat)
```

```
##      [,1] [,2] [,3]
## [1,]    1    2    3
## [2,]    4    5    6
## [3,]    7    8    9
```

```r
mat %*% t(mat)
```

```
##      [,1] [,2] [,3]
## [1,]   66   78   90
## [2,]   78   93  108
## [3,]   90  108  126
```

```r
# Inverse matrix
mat[2, 3] <- 18
solve(mat)
```

```
##       [,1] [,2]        [,3]
## [1,] -1.05  0.1  0.61666667
## [2,]  0.60 -0.2 -0.06666667
## [3,] -0.05  0.1 -0.05000000
```

```r
solve(mat) %*% mat
```

```
##      [,1]          [,2]         [,3]
## [1,]    1  0.000000e+00 8.881784e-16
## [2,]    0  1.000000e+00 2.220446e-16
## [3,]    0 -5.551115e-17 1.000000e+00
```

```r
#====================
# Structures in R
# for loops
for (i in 1:5) {
  print(i)
}
```

3.2.2.4.16 Looping structures in R: For and While loops

```
## [1] 1
## [1] 2
## [1] 3
## [1] 4
## [1] 5
```

```r
# While loops
i <- 10
while (i > 5) {
  i <- i - 1
  print(i)
}
```

```
## [1] 9
## [1] 8
## [1] 7
## [1] 6
```

```
## [1] 5
```

```
# if statments
dave <- TRUE

# if (dave) {} also works
if (dave == TRUE) {
  print("good morning dave")
}
```

```
## [1] "good morning dave"
```

```
# User defined functions
math <- function(a, b) {
  c <- a + b * 2
  # return defines what the output of the function is
  return(c)
}
math(2, 6)
```

```
## [1] 14
```

```

#====================
# In the past few years there has been much progress in developing
# The Tidyverse,  which focuses on Tidy Data Frames
# The packages involved include magrittr and dplyr
# which have streamlined call and code pipelines
# that specify and subset dataframes on the fly
```

3.2.2.4.17 The Tidyverse

3.2.2.5 Some Simple Rmd Items

3.2.2.5.1 Rmd for Exploratory Data Analysis

- EDA is a foundation of Data Science
- Identify sources of data for your problem
- Need to acquire, assemble, clean, and explore your data
- An environment for Exploratory Data Analysis (EDA)

3.2.2.5.2 R markdown is tool for Open Science

- Reproducible data analysis
- Incorporating Data, Code, Presentation and Reporting
- Good coding practices are essential
- Comment your code, describe your data frames
- Make your data analyses a presentation and report.

3.2.2.5.3 R Code in Rmd is delineated

- three backticks for code blocks
- one tick for inline code

3.2.2.5.4 YAML settings and commenting

- This is the top block of the Rmd file

  - set off by three dashes —
- In the YAML Header, you can comment lines with ' #### '
  - But in the body, ' #### ' is a second level header!

3.2.2.5.5 Commenting in the Rmd body

- To comment in the Rmd body, use html comment form
  - < ! - - Comment - - >

  - but with no spaces between the characters
    *
    * i.e.' ''' ends a comment block

–>

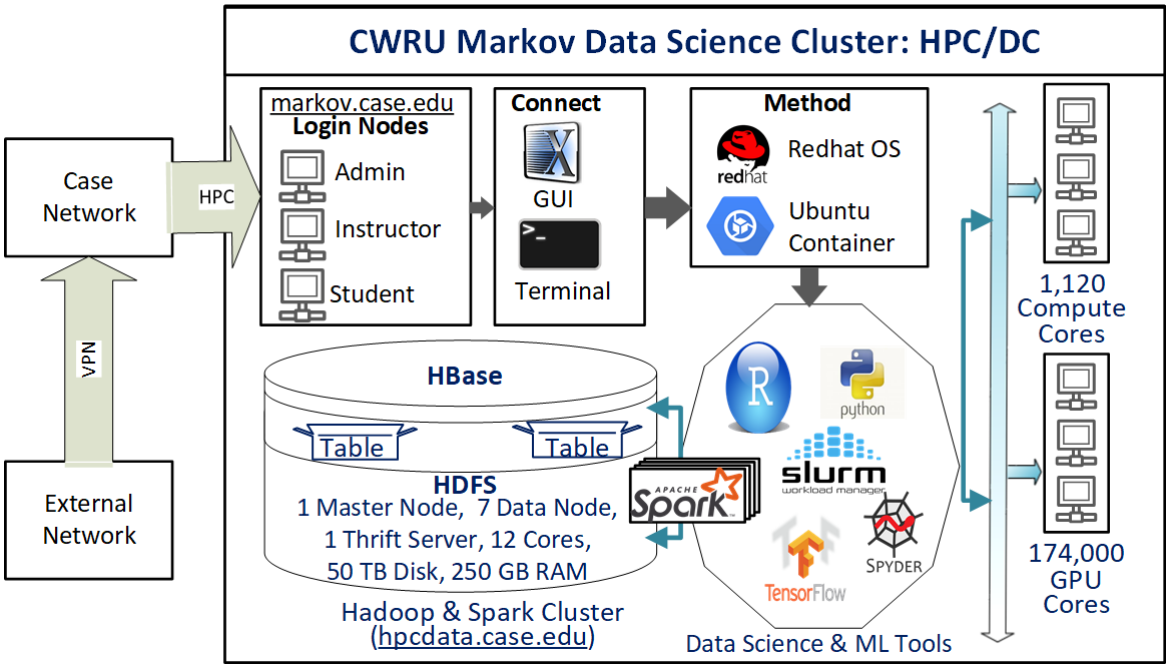


Figure 2: Caption

3.2.2.5.6 Inserting Figs, 2 ways.

3.2.2.6 Filenames and Paths

- Essential to use relative file paths
- Essential to use Posix compatible paths

3.2.2.6.1 Filenames

- No Spaces
- No characters other than letters, numbers, - and underscore
- Better not to capitalize
- Or if you must, use CamelBacking

3.2.2.6.2 Paths

- Windows is not Posix compatible
  - \ is not understood, must be typed \\
    - but should be /,
  - / always works on Linux, Mac, Windows
- Relative Paths
  - . i.e. dot, is the current folder
  - .. i.e. dot dot is the folder one above your current area
- setwd (setting working directory) is bad to rely on.

3.2.2.7 R Coding Training: For Loops

We will try to not use For Loops

- Using the tidyverse
- An dpipes %>% is much more effective
- An produces more readable code

3.2.2.7.1 For loop basics

- For loops are an important part for almost any coding problem
- They work by applying an iterator that changes every time
  - i is the standard iterator over a code block
  - but the iterator can be named anything)

```
# print out numbers upto a given num
num <- 5

for (i in 1:num) {
  print(i)
}
```

3.2.2.7.2 Common example, using a counter

```
## [1] 1
## [1] 2
## [1] 3
## [1] 4
## [1] 5
```

3.2.2.7.3 Vectors or columns can also be iterated over

This can improve clarity in many cases

- if a counter is not needed

```
letters <- c('a','b','c')

for (i in letters) {
  print(i)
}
```

```
## [1] "a"
## [1] "b"
## [1] "c"
```

3.2.2.7.4 Collecting for loop outputs

Lets say we want to calculate the square root of every value in a vector

- using a for loop,
- what is the problem with the code below?

```
num <- c(4, 8, 15, 16, 23, 42)

for (i in num) {
  result <- sqrt(i)
}

result
```

```
## [1] 6.480741
```

We only get 1 number when we wanted 6

- Every time the loop iterates it overwrites the 'result' variable,
  - leaving us with only the last value
- There are multiple was to save out results,
  - depending on what analysis you're running
- I've found this to be one of the most straight forward ways

```
num <- c(4, 8, 15, 16, 23, 42)
# define a NULL variable to write into
all_results <- NULL

for (i in num) {
  # calculate the square root of i
  result <- sqrt(i)
  # concatinate the ith result onto the total result vector
  # rbind() is also useful if the results have multiple variables (columns)
  all_results <- c(all_results, result)
}

all_results
```

- This gives us the answer we wanted

3.2.2.7.5 For loop drawbacks

For loops are highly fundamental

- But they have some problems
- As seen in the example above,
  - organizing results can be messy,
  - especially with complicated results
- They only run one process at a time,
  - making them slow and
  - unable to run parallel process
- Later on we will look at ways to avoid for loops
  - to improve code clarity and increase speed,
  - as well as allow for parallel processing

Dplyr, Pipes and the Tidyverse

- Help avoid the slow performance of For loops
- And streamline/clarify the code

3.2.2.8 Links

http://www.r-project.org

http://rmarkdown.rstudio.com/

<!– # Keep a complete change log history at bottom of file. # Complete Change Log History # v0.00.00 - 1405-07 - Nick Wheeler made the blank script ##########