HuGen2071 book

Daniel E. Weeks

August 9, 2022

Table of contents

Preface

This is a Quarto book.

To learn more about Quarto books visit https://quarto.org/docs/books.

1 Preparation

The first part of our HuGen 2071 course aims to teach you R in the context of applied data wrangling in a genetic context. In our experience, if you have never programmed before, it moves kind of fast. As such, it would be useful to review these sources below.

1.1 Basic programming ideas

1.1.1 Introduction to Coding

This web page and two short videos discusses how computer programming is very similar to writing a recipe - you have to break a complex project down into precise smaller individual steps.

https://subjectguides.york.ac.uk/coding/introduction

1.2 R

1.2.1 PhD Training Workshop: Statistics in R

This online book has a nice introduction to the concepts of programming, RStudio, and R

https://bookdown.org/animestina/R_Manchester/

See Chapters 1, 2, and 3

1.3 R and RStudio

1.3.1 R for the Rest of Us

Acquaint or refresh yourself with R and RStudio — including installing them on your computer with this "R for the Rest of Us course" (24 min of videos + exercises):

https://rfortherestofus.com/courses/getting-started/

Slides: https://rfortherestofus.github.io/getting-started/slides/slides.html

1.4 GitHub

To introduce yourself to GitHub:

https://guides.github.com/introduction/git-handbook/

https://guides.github.com/activities/hello-world/

1.5 R Markdown

To introduce yourself or refresh yourself on R Markdown:

https://rmarkdown.rstudio.com/ (click on Get Started)

1.6 Unix

And finally, to introduce yourself or refresh yourself with Unix (well, Linux in this case, but close enough), try Lessons 1–11 here:

https://www.webminal.org/

2 Introduction

This is a book created from markdown and executable code using Quarto within RStudio.

Book source code: https://github.com/DanielEWeeks/HuGen2071

Created by Daniel E. Weeks

Website: https://www.publichealth.pitt.edu/home/directory/daniel-e-weeks

3 Logistics

3.1 GitHub: Set up an account

Please go to https://github.com and set up a GitHub account.

Choose your GitHub user name carefully, as you may end up using it later in a professional context.

3.2 GitHub Classroom

As GitHub Classroom will be used to distribute course materials and to submit assignments, it would be best if you get git working on your own computer. The easiest way to do this is to install RStudio, R, and git on your computer.

Please follow the detailed instructions in https://github.com/jfiksel/github-classroom-for-students

In particular, see Step 5 re generating an ssh key so you don't need to login every time.

4 R Basics Group Exercise

4.1 Set up the data frame `a`

```r
a <- data.frame(n = 1:4)
dim(a)
```

```
[1] 4 1
```

```r
a
```

```
  n
1 1
2 2
3 3
4 4
```

4.2 Exercise 1: recycling

This exercise should help answer this question: 'In what type of situations would "recycling" be useful?'

Use recycling to insert into the data frame `a` a column named `rowNum1` that contains a 1 in even rows and a 2 in odd rows.

Tip

The R command

```r
a$rowNum1 <- NA
```

would insert a new row into the data frame `a` full of `NA` values.

Expand to see the answer

```r
a$rowNum1 <- c(1,2)
a
```

```
  n rowNum1
1 1       1
2 2       2
3 3       1
4 4       2
```

4.3 Exercise 2: vector addition

Use vector addition to construct a vector of length 4 that contains a 1 in even rows and a 2 in odd rows. Then insert this vector into the data frame `a` into a column named `rowNum6`.

Tip

What vector could you add to this vector so the sum is the vector (1, 2, 1, 2)?

```r
rep(1, 4)
```

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

Expand to see the answer

```r
r1 <- rep(1, times = 4)
r2 <- rep(c(0,1), times = 2)
r1
```

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

```r
r2
```

```
[1] 0 1 0 1
```

```r
r1 + r2
```

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

```
a$rowNum6 <- r1 + r2
a
```

```
  n rowNum1 rowNum6
1 1       1       1
2 2       2       2
3 3       1       1
4 4       2       2
```

4.4 Exercise 3: `for` loops

Loops allow you to repeat actions on each item from a vector of items.

Here is an example `for` loop, iterating through the values of `i` from 1 to 3:

```
for (i in 1:3) {
  print(paste("i =",i))
}
```

```
[1] "i = 1"
[1] "i = 2"
[1] "i = 3"
```

This does the same thing as this repetitive code:

```
i.vector <- c(1,2,3)
i <- i.vector[1]
print(paste("i =",i))
```

```
[1] "i = 1"
```

```
i <- i.vector[2]
print(paste("i =",i))
```

```
[1] "i = 2"
```

```r
i <- i.vector[3]
print(paste("i =",i))
```

```
[1] "i = 3"
```

Use a `for` loop to insert into the data frame `a` a column named `rowNum2` that contains a 1 in even rows and a 2 in odd rows.

Tip

Think about how as `i` increments from 1 to `nrow(a)`, how could we map that sequence (e.g. 1, 2, 3, 4) to the desired sequence of 1, 2, 1, 2.

Expand to see the answer

```r
# Set value that we want to iterate 1, 2, 1, 2, ...
j <- 1
# Initialize rowNum2 to all missing values
a$rowNum2 <- NA
# Start the for loop, looping over the number of rows in a
for (i in c(1:nrow(a))) {
   # Assign value j to row i
   a$rowNum2[i] <- j
   # Increment j
   j <- j + 1
   # If j is greater than 2, set it back to 1
   if (j > 2) {
     j <- 1
   }
}
a
```

```
  n rowNum1 rowNum6 rowNum2
1 1       1       1       1
2 2       2       2       2
3 3       1       1       1
4 4       2       2       2
```

4.5 Exercise 4: `while` loops

Here's an example `while` loop:

```r
i <- 1
while (i < 4) {
  print(paste("i =",i))
  i <- i + 1
}
```

```
[1] "i = 1"
[1] "i = 2"
[1] "i = 3"
```

Use a `while` loop to insert into the data frame `a` a column named `rowNum3` that contains a 1 in even rows and a 2 in odd rows.

Expand to see the answer

```r
a$rowNum3 = NA
i <- 1 #set index
while(i <= nrow(a)){ #set conditions for while loop

  if ((i %% 2)) { #if statement for when "i" is odd
    a$rowNum3[i] <- 1
  }
  else #else statement for when "i" is even
    a$rowNum3[i] <- 2

  i <- i + 1 #counter for "i", increments by 1 with each loop iteration
}
a
```

```
  n rowNum1 rowNum6 rowNum2 rowNum3
1 1       1       1       1       1
2 2       2       2       2       2
3 3       1       1       1       1
4 4       2       2       2       2
```

4.6 Exercise 5: `repeat` loops

Here's an example `repeat` loop:

```
i <- 1
repeat {
  print(paste("i =",i))
  i <- i + 1
  if (i > 3) break
}
```

```
[1] "i = 1"
[1] "i = 2"
[1] "i = 3"
```

Use a `repeat` loop to insert into the data frame `a` a column named `rowNum4` that contains a 1 in even rows and a 2 in odd rows.

Expand to see the answer

```
a$rowNum4 <- NA
i <- 1 #set index
repeat {

  if ((i %% 2)) { #if statement for when "i" is odd
    a$rowNum4[i] <- 1
  }
  else #else statement for when "i" is even
    a$rowNum4[i] <- 2

  i <- i + 1 #counter for "i", increments by 1 with each loop iteration
  if (i > nrow(a)) {
    break
  }
}
a
```

```
  n rowNum1 rowNum6 rowNum2 rowNum3 rowNum4
1 1       1       1       1       1       1
2 2       2       2       2       2       2
3 3       1       1       1       1       1
```

```
4 4       2       2       2       2       2
```

4.7 Exercise 6: using the `rep` function

Use the `rep` command to insert into the data frame `a` a column named `rowNum5` that contains a 1 in even rows and a 2 in odd rows.

Expand to see the answer

```
# This will only work correctly if nrow(a) is even
a$rowNum5 <- rep(c(1,2), nrow(a)/2)
a
```

```
  n rowNum1 rowNum6 rowNum2 rowNum3 rowNum4 rowNum5
1 1       1       1       1       1       1       1
2 2       2       2       2       2       2       2
3 3       1       1       1       1       1       1
4 4       2       2       2       2       2       2
```

4.8 Exercise 7

List all even rows of the data frame `a`.

List rows 3 and 4 of the data frame `a`.

Expand to see the answer

```
# All even rows
a[a$rowNum1==2,]
```

```
  n rowNum1 rowNum6 rowNum2 rowNum3 rowNum4 rowNum5
2 2       2       2       2       2       2       2
4 4       2       2       2       2       2       2
```

```
# All odd rows
a[a$rowNum1==1,]
```

```
  n rowNum1 rowNum6 rowNum2 rowNum3 rowNum4 rowNum5
```

```
1 1        1        1        1        1        1        1
3 3        1        1        1        1        1        1
```

4.9 Exercise 8

Note

Learning objective: Learn how to alter the options of an R command to achieve your goals.

This exercise should help answer this question: “When reading a file, will missing data be automatically represented as NA values, or does that need to be coded/manually curated?”

The tab-delimited file in `testdata.txt` contains the following data:

```
1        1        1
2        2        2
3        NA       99
4        4        4
```

Your collaborator who gave you these data informed you that in this file `99` stands for a missing value, as does `NA`.

However if we use the `read.table` command with its default options to read this in, we fail to accomplish the desired task, as `99` is not reading as a missing value:

```r
infile <- "data/testdata.txt"
# Adjust the read.table options to read the file correctly as desired.
b <- read.table(infile)
b
```

```
  V1 V2 V3
1  1  1  1
2  2  2  2
3  3 NA 99
4  4  4  4
```

```r
str(b)
```

```
'data.frame':	4 obs. of  3 variables:
 $ V1: int  1 2 3 4
 $ V2: int  1 2 NA 4
 $ V3: int  1 2 99 4
```

Use the `read.table` command to read this file in while automatically setting both the 'NA" and the `99` to `NA`. This can be done by adjusting the various options of the `read.table` command.

Tip

Read the help page for the `read.table` command

Expand to see the answer

To read this in properly, we have to let 'read.table' know that there is no header and that which values should be mapped to the missing NA value:

```r
b <- read.table(infile, header = FALSE, na.strings = c("NA","99"))
b
```

```
  V1 V2 V3
1  1  1  1
2  2  2  2
3  3 NA NA
4  4  4  4
```

```r
str(b)
```

```
'data.frame':	4 obs. of  3 variables:
 $ V1: int  1 2 3 4
 $ V2: int  1 2 NA 4
 $ V3: int  1 2 NA 4
```

```r
summary(b)
```

```
       V1              V2               V3       
 Min.   :1.00   Min.   :1.000   Min.   :1.000  
 1st Qu.:1.75   1st Qu.:1.500   1st Qu.:1.500  
 Median :2.50   Median :2.000   Median :2.000  
 Mean   :2.50   Mean   :2.333   Mean   :2.333  
```

```
3rd Qu.:3.25   3rd Qu.:3.000   3rd Qu.:3.000
Max.   :4.00   Max.   :4.000   Max.   :4.000
               NA's   :1       NA's   :1
```

5 R Character Exercise

6 Load Libraries

```r
library(tidyverse)
# library(tidylog)
library(knitr)
```

7 Useful RStudio cheatsheet

See the “String manipulation with stringr cheatsheet” at

https://www.rstudio.com/resources/cheatsheets/

8 Scenario 1

You are working with three different sets of collaborators: 1) the clinical group that did the field work and generated the anthropometric measurements; 2) the medical laboratory that measured blood pressure in a controlled environment; and 3) the molecular laboratory that generated the genotypes.

```r
clin <- read.table(file = "data/clinical_data.txt", header=TRUE)
kable(clin)
```

ID	height
1	152
104	172
2112	180
2543	163

```r
lab <- read.table(file = "data/lab_data.txt", header = TRUE)
kable(lab)
```

ID	SBP
SG0001	120
SG0104	111
SG2112	125
SG2543	119

```r
geno <- read.table(file = "data/genotype_data.txt", header = TRUE)
kable(geno)
```

Sample	rs1212
TaqMan-SG0001-190601	G/C
TaqMan-SG0104-190602	G/G
TaqMan-SG2112-190603	C/C

Sample	rs1212
TaqMan-Sg2543-190603	C/G

9 Discussion Questions

9.1 Question 1

The clinical group, which measured height, used integer IDs, but the medical group, which measured the blood pressure, decided to prefix the integer IDs with the string 'SG' (so as to distinguish them from other studies that were also using integer IDs). So ID '1' was mapped to ID 'SG0001'.

Discuss how, using R commands, you would reformat the integer IDs to be in the format "SGXXXX". Write down your ideas in the next section, and, if you have time, try them out within an R chunk.

Table 9.1: The `clin` data frame

ID	height
1	152
104	172
2112	180
2543	163

Hint: Use the `formatC` function.

9.2 Answer 1

Expand to see solution

```r
clin$SUBJECT_ID <- paste0("SG", formatC(clin$ID, width = 4, flag = "0000"))
kable(clin)
```

ID	height	SUBJECT_ID
1	152	SG0001

104	172	SG0104
2112	180	SG2112
2543	163	SG2543

9.3 Question 2

Discuss how, using R commands, you would reformat the "SGXXXX" IDs to be integer IDs. Write down your ideas in the next section, and, if you have time, try them out within an R chunk.

Table 9.3: The `lab` data frame

ID	SBP
SG0001	120
SG0104	111
SG2112	125
SG2543	119

Hint: Use either the `gsub` command or the `str_replace_all` command.

9.4 Answer 2

Expand to see solution

```
lab$ID2 <- as.numeric(gsub("SG","",lab$ID))
kable(lab)
```

ID	SBP	ID2
SG0001	120	1
SG0104	111	104
SG2112	125	2112
SG2543	119	2543

```
lab$ID2 <- NA
lab$ID2 <- str_replace_all(lab$ID, pattern = "SG", replacement = "") %>% as.numeric()
kable(lab)
```

ID	SBP	ID2
SG0001	120	1
SG0104	111	104
SG2112	125	2112
SG2543	119	2543

9.5 Question 3

The genotype group used IDs in the style “TaqMan-SG0001-190601”, where the first string is “TaqMan” and the ending string is the date of the genotyping experiment.

Discuss how, using R commands, you would extract an “SGXXXX” style ID from the “TaqMan-SG0001-190601” style IDs. Write down your ideas in the next section, and, if you have time, try them out within an R chunk.

Note that one of the IDs has a lower case ‘g’ in it - how would you correct this, using R commands?

Table 9.6: The `geno` data frame

Sample	rs1212
TaqMan-SG0001-190601	G/C
TaqMan-SG0104-190602	G/G
TaqMan-SG2112-190603	C/C
TaqMan-Sg2543-190603	C/G

Hint: Use either the `str_split_fixed` function or the `separate` function.

9.6 Answer 3

Expand to see solution

```r
a <- str_split_fixed(geno$Sample, pattern = "-",n=3)
a
```

```
     [,1]     [,2]     [,3]    
[1,] "TaqMan" "SG0001" "190601"
[2,] "TaqMan" "SG0104" "190602"
[3,] "TaqMan" "SG2112" "190603"
[4,] "TaqMan" "Sg2543" "190603"
```

```r
geno$ID <- toupper(a[,2])
kable(geno)
```

Sample	rs1212	ID
TaqMan-SG0001-190601	G/C	SG0001
TaqMan-SG0104-190602	G/G	SG0104
TaqMan-SG2112-190603	C/C	SG2112
TaqMan-Sg2543-190603	C/G	SG2543

The `separate` function from the `tidyr` package is also useful:

```r
geno %>% separate(Sample, into=c("Tech","ID","Suffix"), sep="-")
```

```
    Tech     ID Suffix rs1212
1 TaqMan SG0001 190601    G/C
2 TaqMan SG0104 190602    G/G
3 TaqMan SG2112 190603    C/C
4 TaqMan Sg2543 190603    C/G
```

10 Scenario 2

A replication sample has been measured, and that is using IDs in the style "RP5XXX".

```r
joint <- read.table(file = "data/joint_data.txt", header = TRUE)
kable(joint)
```

ID	SBP
SG0001	120
SG0104	111
SG2112	125
SG2543	119
RP5002	121
RP5012	118
RP5113	112
RP5213	142

10.1 Question 4

Discuss how you would use R commands to split the 'joint' data frame into an 'SG' and 'RP' specific piece? Write down your ideas in the next section, and, if you have time, try them out within an R chunk.

Table 10.2: The `joint` data frame

ID	SBP
SG0001	120
SG0104	111
SG2112	125
SG2543	119
RP5002	121
RP5012	118
RP5113	112
RP5213	142

10.2 Answer 4

Expand to see solution

```
grep(pattern = "SG",joint$ID)
```

```
[1] 1 2 3 4
```

```
grep(pattern = "RP", joint$ID)
```

```
[1] 5 6 7 8
```

```
joint.SG <- joint[grep(pattern = "SG",joint$ID), ]
joint.RP <- joint[grep(pattern = "RP", joint$ID), ]
kable(joint.SG)
```

ID	SBP
SG0001	120
SG0104	111
SG2112	125
SG2543	119

```r
kable(joint.RP)
```

	ID	SBP
5	RP5002	121
6	RP5012	118
7	RP5113	112
8	RP5213	142

```r
# Reset row names
rownames(joint.RP) <- NULL
kable(joint.RP)
```

ID	SBP
RP5002	121
RP5012	118
RP5113	112
RP5213	142

11 R Functions Excercise

12 Load Libraries

```r
 library(tidyverse)
# library(tidylog)
```

13 Location

This file `exercise1_solution.Qmd` is in the “HuGen2071_book” sub-folder of the “hugen2071” folder of our Lectures repository.

```
paste0(basename(dirname(getwd())),"/",basename(getwd()))
```

```
[1] "hugen2071/HuGen2071_book"
```

14 Data set creation code

```
i <- 6
for (i in 1:10) {
fl <- data.frame(name=rep(paste0("name",i),26))
b <- data.frame(name = rep(NA, 26))
b$name <- paste0(fl$name,"_",letters)
b$trait <- rnorm(26)
write_tsv(b,paste0("data/dataset",i,".txt"))
}
```

15 Example

Here we have been sent three data sets in the files that contain the trait quantitative values for each person in the data set:

"dataset1.txt" "dataset2.txt" "dataset3.txt"

And we've been asked to make a table that gives, for each dataset, the sample size (N), the mean of the trait, the median, and the variance.

We could do this by reading in each data set, one by one, as follows:

```
results <- data.frame(dataset=rep(NA,3),N=NA, mean=NA, median=NA, var=NA)
fl1 <- read.table("data/dataset1.txt",sep="\t",header=TRUE)
results$dataset[1] <- "dataset1"
results$N <- nrow(fl1)
results$mean[1] <- mean(fl1$trait)
results$median[1] <- median(fl1$trait)
results$var[1] <- var(fl1$trait)
results
```

```
   dataset  N       mean    median       var
1 dataset1 26 0.09762111 0.2198957 0.5974116
2     <NA> 26         NA        NA        NA
3     <NA> 26         NA        NA        NA
```

```
fl2 <- read.table("data/dataset2.txt",sep="\t",header=TRUE)
results$dataset[2] <- "dataset2"
results$N <- nrow(fl2)
results$mean[2] <- mean(fl2$trait)
results$median[2] <- median(fl2$trait)
results$var[2] <- var(fl2$trait)
results
```

```
   dataset  N       mean    median       var
1 dataset1 26 0.09762111 0.2198957 0.5974116
```

```
2 dataset2 26 0.43486401 0.3558736 1.0936651
3     <NA> 26         NA        NA        NA
```

```
fl3 <- read.table("data/dataset3.txt",sep="\t",header=TRUE)
results$dataset[3] <- "dataset3"
results$N <- nrow(fl3)
results$mean[3] <- mean(fl3$trait)
results$median[3] <- median(fl3$trait)
results$var[3] <- var(fl3$trait)
results
```

```
   dataset  N       mean    median       var
1 dataset1 26 0.09762111 0.2198957 0.5974116
2 dataset2 26 0.43486401 0.3558736 1.0936651
3 dataset3 26 0.07508335 0.0445614 0.7950574
```

Your colleague initially sent you the three data sets above, but now your colleague has sent you three more data sets and asked you to update the 'results' table.

As you can see, the code above is very repetitive. So let's automate this by writing a function that loops through a list of data set files named "dataset1.txt", "dataset2.txt", "dataset3.txt", etc., building up the results table as above.

15.1 Question: How could we construct a list of file names?

How could we construct a list of file names?

Expand to see solution

Hint: the `list.files` command provides a handy way to get a list of the input files:

```
fls <- list.files(path="data",pattern="dataset*")
fls
```

```
[1] "dataset1.txt" "dataset2.txt" "dataset3.txt" "dataset4.txt" "dataset5.txt"
[6] "dataset6.txt"
```

15.2 Question: Outline a possible algorithm

Outline a possible algorithm that loops through a list of input data set files named “dataset1.txt”, “dataset2.txt”, “dataset3.txt”, etc., building up the results table as above.

Expand to see solution

- Read in the input file names into a list
- Set up an empty results table
- For each file in our file name list
  - Read the file
  - Compute the statistics
  - Insert the information into the results table
  - Return the filled-in results table

15.3 Question: Construct a more detailed step-by-step algorithm.

Construct a more detailed step-by-step algorithm.

Expand to see solution

- Input the path to the folder containing the data files
- Read in the input file names into a list `fls`
- Count the number of input files `N`
- Set up an empty results table with `N` rows
- For each file in our file name list `fls`
  - Read the file
  - Compute the statistics
  - Insert the information into the correct row of the results table
- Return the filled-in results table

15.4 Task: Write a `read_data_file` function.

Write a `read_data_file` function to accomplish the required steps for a single input data file.

1. Make the number in the data file name an argument.

Expand to see solution

Here we make the number in the data file name an argument

```
results <- data.frame(dataset=rep(NA,6),N=NA, mean=NA, median=NA, var=NA)
read_data_file <- function(n=1, results) {
  fl1 <- read.table(paste0("data/dataset",n,".txt"),sep="\t",header=TRUE)
  results$dataset[n] <- paste0("dataset",n,".txt")
  results$N <- nrow(fl1)
  results$mean[n] <- mean(fl1$trait)
  results$median[n] <- median(fl1$trait)
  results$var[n] <- var(fl1$trait)
  invisible(results)
}
```

2. Make the path to the input file an argument to your `read_data_file` function.

Expand to see solution

Here we make the path to the input file an argument.

```
read_data_file_v2 <- function(flnm, results) {
  fl1 <- read.table(paste0("data/",flnm),sep="\t",header=TRUE)
  results$dataset[n] <- flnm
  results$N <- nrow(fl1)
  results$mean[n] <- mean(fl1$trait)
  results$median[n] <- median(fl1$trait)
  results$var[n] <- var(fl1$trait)
  invisible(results)
}
```

15.5 Question: What does the above code assume?

What does the above code assume?

Expand to see solution

Assumes a file naming style of 'dataset*.txt' where the asterisk represents 1, 2, 3, ...
Assumes the files are in the "data" folder.

15.6 Question: Extend your function to process all of the files

The above function `read_data_file` processes one file at a time. How would you write a function to loop this over to process all of our files?

Expand to see solution

```
fls <- list.files(path="data",pattern="dataset*")

loop_over_dataset <- function(fls) {
  # Input: the list of file names
  # Output: the 'results table
  # Count the number of data set file names in fls
  n_datasets <- length(fls)
  # Set up a results dataframe with n_datasets rows
  results <- data.frame(dataset=rep(NA,n_datasets),N=NA, mean=NA, median=NA, var=NA)
  for (n in 1:n_datasets) {
    results <- read_data_file(n=n, results=results)
  }
  return(results)
}
```

```
loop_over_dataset(fls = fls)
```

```
      dataset  N        mean      median       var
1 dataset1.txt 26  0.09762111  0.21989574 0.5974116
2 dataset2.txt 26  0.43486401  0.35587359 1.0936651
3 dataset3.txt 26  0.07508335  0.04456140 0.7950574
4 dataset4.txt 26  0.06259720  0.04813915 0.9186042
5 dataset5.txt 26 -0.09288522 -0.19155759 0.9978161
6 dataset6.txt 26 -0.20266667 -0.23845426 1.5605823
```

15.7 Bonus question

Can you find a subtle mistake in the `read_data_file` function?

```
results <- data.frame(dataset=rep(NA,6),N=NA, mean=NA, median=NA, var=NA)
read_data_file <- function(n=1, results) {
  fl1 <- read.table(paste0("data/dataset",n,".txt"),sep="\t",header=TRUE)
  results$dataset[n] <- paste0("dataset",n,".txt")
  results$N <- nrow(fl1)
  results$mean[n] <- mean(fl1$trait)
  results$median[n] <- median(fl1$trait)
  results$var[n] <- var(fl1$trait)
  invisible(results)
}
```

Expand to see solution

If `N` varies across the data sets, then this line will not do the right thing:

```
results$N <- nrow(fl1)
```

```
results <- data.frame(dataset=rep(NA,6),N=NA, mean=NA, median=NA, var=NA)
read_data_file <- function(n=1, results) {
  fl1 <- read.table(paste0("data/dataset",n,".txt"),sep="\t",header=TRUE)
  results$dataset[n] <- paste0("dataset",n,".txt")
  results$N[n] <- nrow(fl1)
  results$mean[n] <- mean(fl1$trait)
  results$median[n] <- median(fl1$trait)
  results$var[n] <- var(fl1$trait)
  invisible(results)
}
```

16 R Tidyverse Exercise

17 Load Libraries

Load the `tidyverse` packages

```
 library(tidyverse)
# library(tidylog)
```

18 Untidy data

Let's use the World Health Organization TB data set from the `tidyr` package

```
who <- tidyr::who
dim(who)
```

```
[1] 7240   60
```

```
head(who[,1:6] %>% filter(!is.na(new_sp_m014)))
```

```
# A tibble: 6 x 6
  country     iso2  iso3   year new_sp_m014 new_sp_m1524
  <chr>       <chr> <chr> <int>       <int>        <int>
1 Afghanistan AF    AFG    1997           0           10
2 Afghanistan AF    AFG    1998          30          129
3 Afghanistan AF    AFG    1999           8           55
4 Afghanistan AF    AFG    2000          52          228
5 Afghanistan AF    AFG    2001         129          379
6 Afghanistan AF    AFG    2002          90          476
```

See the help page for `who` for more information about this data set.

In particular, note this description:

"The data uses the original codes given by the World Health Organization. The column names for columns five through 60 are made by combining new_ to a code for method of diagnosis (rel = relapse, sn = negative pulmonary smear, sp = positive pulmonary smear, ep = extrapulmonary) to a code for gender (f = female, m = male) to a code for age group (014 = 0-14 yrs of age, 1524 = 15-24 years of age, 2534 = 25 to 34 years of age, 3544 = 35 to 44 years of age, 4554 = 45 to 54 years of age, 5564 = 55 to 64 years of age, 65 = 65 years of age or older)."

So `new_sp_m014` represents the counts of new TB cases detected by a positive pulmonary smear in males in the 0-14 age group.

19 Tidy data

Tidy data: Have each variable in a column.

Question: Are these data tidy?

Expand to see solution

No these data are not tidy because aspects of the data that should be variables are encoded in the name of the variables.
These aspects are

1. test type.
2. sex of the subjects.
3. age range of the subjects.

Question: How would we make these data tidy?

Consider this portion of the data:

```
head(who[,1:5] %>% filter(!is.na(new_sp_m014) & new_sp_m014>0), 1)
```

```
# A tibble: 1 x 5
  country     iso2  iso3   year new_sp_m014
  <chr>       <chr> <chr> <int>       <int>
1 Afghanistan AF    AFG    1998          30
```

Expand to see solution

We would replace the `new_sp_m014` with the following four columns:

```
type  sex   age  n
sp    m     014  30
```

This would place each variable in its own column.

20 Gather

```
stocks <- tibble(
  time = as.Date('2009-01-01') + 0:9,
  X = rnorm(10, 0, 1),
  Y = rnorm(10, 0, 2),
  Z = rnorm(10, 0, 4)
)

head(stocks)
```

```
# A tibble: 6 x 4
  time             X       Y     Z
  <date>       <dbl>   <dbl> <dbl>
1 2009-01-01 -0.0999 -0.643  -3.73
2 2009-01-02 -0.314  -0.0374  4.30
3 2009-01-03  0.900   1.67   -8.03
4 2009-01-04 -0.671   0.506  -1.69
5 2009-01-05  0.483   0.490   3.79
6 2009-01-06  0.907   5.59    9.98
```

```
stocks %>% gather("stock", "price", -time) %>% head()
```

```
# A tibble: 6 x 3
  time       stock   price
  <date>     <chr>   <dbl>
1 2009-01-01 X     -0.0999
2 2009-01-02 X     -0.314
3 2009-01-03 X      0.900
4 2009-01-04 X     -0.671
5 2009-01-05 X      0.483
6 2009-01-06 X      0.907
```

21 Pivot_longer

```
stocks %>% pivot_longer(c(X,Y,Z), names_to= "stock", values_to = "price") %>%
  head()
```

```
# A tibble: 6 x 3
  time       stock   price
  <date>     <chr>   <dbl>
1 2009-01-01 X     -0.0999
2 2009-01-01 Y     -0.643
3 2009-01-01 Z     -3.73
4 2009-01-02 X     -0.314
5 2009-01-02 Y     -0.0374
6 2009-01-02 Z      4.30
```

22 WHO TB data

Question: How would we convert this to tidy form?

```r
head(who[,1:6] %>% filter(!is.na(new_sp_m014)))
```

```
# A tibble: 6 x 6
  country     iso2  iso3   year new_sp_m014 new_sp_m1524
  <chr>       <chr> <chr> <int>       <int>        <int>
1 Afghanistan AF    AFG    1997           0           10
2 Afghanistan AF    AFG    1998          30          129
3 Afghanistan AF    AFG    1999           8           55
4 Afghanistan AF    AFG    2000          52          228
5 Afghanistan AF    AFG    2001         129          379
6 Afghanistan AF    AFG    2002          90          476
```

Expand to see solution

```r
who.long <- who %>% pivot_longer(starts_with("new"), names_to = "demo", values_to = "n")
head(who.long)
```

```
# A tibble: 6 x 6
  country     iso2  iso3   year demo             n
  <chr>       <chr> <chr> <int> <chr>        <int>
1 Afghanistan AF    AFG    1997 new_sp_m014      0
2 Afghanistan AF    AFG    1997 new_sp_m1524    10
3 Afghanistan AF    AFG    1997 new_sp_m2534     6
4 Afghanistan AF    AFG    1997 new_sp_m3544     3
5 Afghanistan AF    AFG    1997 new_sp_m4554     5
6 Afghanistan AF    AFG    1997 new_sp_m5564     2
```

Question: How would we split `demo` into variables?

```r
head(who.long)
```

```
# A tibble: 6 x 6
  country     iso2  iso3   year demo             n
  <chr>       <chr> <chr> <int> <chr>        <int>
1 Afghanistan AF    AFG    1997 new_sp_m014      0
2 Afghanistan AF    AFG    1997 new_sp_m1524    10
3 Afghanistan AF    AFG    1997 new_sp_m2534     6
4 Afghanistan AF    AFG    1997 new_sp_m3544     3
5 Afghanistan AF    AFG    1997 new_sp_m4554     5
6 Afghanistan AF    AFG    1997 new_sp_m5564     2
```

Look at the variable naming scheme:

```
names(who) %>% grep("m014",., value=TRUE)
```

```
[1] "new_sp_m014" "new_sn_m014" "new_ep_m014" "newrel_m014"
```

Question: How should we adjust the `demo` strings so as to be able to easily split all of them into the desired variables?

Expand to see solution

```
who.long <- who.long %>%
  mutate(demo = str_replace(demo, "newrel", "new_rel"))
grep("m014",who.long$demo, value=TRUE) %>%  unique()
```

```
[1] "new_sp_m014"  "new_sn_m014"  "new_ep_m014"  "new_rel_m014"
```

Question: After adjusting the `demo` strings, how would we then separate them into the desired variables?

Expand to see solution

```
who.long <- who.long %>%
  separate(demo, into = c("new", "type", "sexagerange"), sep="_") %>%
  separate(sexagerange, into=c("sex","age_range"), sep=1) %>%
  select(-new)
head(who.long)
```

```
# A tibble: 6 x 8
  country     iso2  iso3   year type  sex   age_range     n
  <chr>       <chr> <chr> <int> <chr> <chr> <chr>     <int>
1 Afghanistan AF    AFG    1997 sp    m     014           0
2 Afghanistan AF    AFG    1997 sp    m     1524         10
3 Afghanistan AF    AFG    1997 sp    m     2534          6
4 Afghanistan AF    AFG    1997 sp    m     3544          3
5 Afghanistan AF    AFG    1997 sp    m     4554          5
6 Afghanistan AF    AFG    1997 sp    m     5564          2
```

23 Conclusion

Now our untidy data are tidy.

```
head(who.long)
```

```
# A tibble: 6 x 8
  country     iso2  iso3   year type  sex   age_range     n
  <chr>       <chr> <chr> <int> <chr> <chr> <chr>     <int>
1 Afghanistan AF    AFG    1997 sp    m     014           0
2 Afghanistan AF    AFG    1997 sp    m     1524         10
3 Afghanistan AF    AFG    1997 sp    m     2534          6
4 Afghanistan AF    AFG    1997 sp    m     3544          3
5 Afghanistan AF    AFG    1997 sp    m     4554          5
6 Afghanistan AF    AFG    1997 sp    m     5564          2
```

24 Acknowledgment

This exercise was modeled, in part, on this exercise:

https://people.duke.edu/~ccc14/cfar-data-workshop-2018/CFAR_R_Workshop_2018_Exercisees.html

25 R Recoding Reshaping Exercise

26 Load Libraries

```r
library(tidyverse)
# library(tidylog)
```

27 Project 1 Data

In the `ds` data frame we have the synthetic yet realistic data we will be using in Project 1.

In the `dd` data frame we have the corresponding data dictionary.

```
load("data/exercise.RData", verbose = TRUE)
```

```
Loading objects:
  ds
  dd
  DictPer
```

```
dim(ds)
```

```
[1] 191  24
```

```
names(ds)
```

```
 [1] "sample_id"                "Sample_trimester"
 [3] "Gestationalage_sample"    "subject_id"
 [5] "strata"                   "race"
 [7] "maternal_age_delivery"    "case_control_status"
 [9] "prepregnancy_weight"      "height"
[11] "prepregnancy_BMI"         "gravidity"
[13] "parity"                   "gestationalage_delivery"
[15] "average_SBP_lt20weeks"    "average_DBP_lt20weeks"
[17] "average_SBP_labor"        "average_DBP_labor"
[19] "smoke_lifetime"           "baby_birthweight"
[21] "baby_sex"                 "baby_birthweight_centile"
[23] "baby_SGA"                 "placental_pathology"
```

```
dim(dd)
```

```
[1] 27  5
```

```
names(dd)
```

```
[1] "Original.Variable.Name" "R21.Variable.Name"      "Description"
[4] "Variable.Units"         "Variable.Coding"
```

28 Exercise 1

Skill: Checking for duplicated IDs

```
ds %>%
    select(subject_id, sample_id, height) %>%
    head(n = 10)
```

```
   subject_id sample_id height
1      SUBJ48   SAMP149   64.6
2      SUBJ46   SAMP037   65.7
3      SUBJ28   SAMP120   63.3
4      SUBJ26   SAMP187   61.1
5      SUBJ49   SAMP082   67.6
6      SUBJ48   SAMP149   64.6
7      SUBJ19   SAMP074   66.1
8      SUBJ07   SAMP063   64.4
9      SUBJ28   SAMP053   63.3
10     SUBJ43   SAMP085   65.7
```

Check if there are any duplicated `sample_id`'s using the `duplicated` command.

Expand to see solution

```
sum(duplicated(ds$sample_id))
```

```
[1] 72
```

Construct a table of the number of times each `sample_id` is duplicated:

Expand to see solution

```
table(table(ds$sample_id))
```

```
 1  2  3  4  5 
67 35 13  2  1 
```

```
# But?
sum(duplicated(ds$sample_id))
```

```
[1] 72
```

```
35 + 13 * 2 + 2 * 3 + 1 * 4
```

```
[1] 71
```

```
sum(duplicated(ds$sample_id, incomparables = NA))
```

```
[1] 71
```

```
table(table(ds$sample_id, useNA = "always"))
```

```

 1  2  3  4  5 
67 36 13  2  1 
```

```
36 + 13 * 2 + 2 * 3 + 1 * 4
```

```
[1] 72
```

Check if there are any duplicated `subject_ids`

Expand to see solution

```
sum(duplicated(ds$subject_id))
```

```
[1] 137
```

29 Checking for duplicates

How do we return every row that contains a duplicate?

```r
f <- data.frame(ID = c(1, 1, 2), c2 = c(1, 2, 3))
f
```

```
  ID c2
1  1  1
2  1  2
3  2  3
```

```r
f[duplicated(f$ID), ]
```

```
  ID c2
2  1  2
```

30 Counting the number of occurences of the ID

```
f %>%
    group_by(ID) %>%
    summarise(n = n())
```

```
# A tibble: 2 x 2
     ID     n
  <dbl> <int>
1     1     2
2     2     1
```

31 Count `sample_id` duplicates

Using Tidyverse commands, count how many times each `sample_id` occcurs in the `ds` data frame, reporting the counts in descending order, from highest to lowest.

Expand to see solution

```
ds %>%
    group_by(sample_id) %>%
    summarise(n = n()) %>%
    filter(n > 1) %>%
    arrange(desc(n)) %>%
    head()
```

```
# A tibble: 6 x 2
  sample_id     n
  <chr>     <int>
1 SAMP100       5
2 SAMP125       4
3 SAMP155       4
4 SAMP017       3
5 SAMP048       3
6 SAMP058       3
```

```
ds %>%
    group_by(sample_id) %>%
    summarise(n = n()) %>%
    filter(n > 1) %>%
    arrange(desc(n)) %>%
    pull(n) %>%
    table()
```

```
.
 2  3  4  5
36 13  2  1
```

32 Checking for duplicates

Here we list all of the rows containing a duplicated 'ID' value using functions from the 'tidyverse' package:

```
f %>%
    group_by(ID) %>%
    filter(n() > 1)
```

```
# A tibble: 2 x 2
# Groups:   ID [1]
     ID    c2
  <dbl> <dbl>
1     1     1
2     1     2
```

32.1 How to list all duplicates

Use Tidyverse commands to list all duplicates for `sample_id` and for `subject_id`. Sort the results by the ID.

Expand to see solution

32.2 Sample ID

```
ds %>%
    group_by(sample_id) %>%
    filter(n() > 1) %>%
    select(sample_id, subject_id, Sample_trimester, Gestationalage_sample) %>%
    arrange(sample_id, Sample_trimester, Gestationalage_sample) %>%
    head()
```

```
# A tibble: 6 x 4
# Groups:   sample_id [3]
```

```
  sample_id subject_id Sample_trimester Gestationalage_sample
  <chr>     <chr>                 <dbl>                 <dbl>
1 SAMP002   SUBJ20                    2                 19.3
2 SAMP002   SUBJ20                    2                 19.7
3 SAMP003   SUBJ12                    1                  8.25
4 SAMP003   SUBJ12                    1                  8.35
5 SAMP004   SUBJ35                    2                 20.4
6 SAMP004   SUBJ35                    2                 20.9
```

32.3 Subject ID

```
ds %>%
    group_by(subject_id) %>%
    filter(n() > 1) %>%
    select(subject_id, sample_id, Sample_trimester, Gestationalage_sample) %>%
    arrange(subject_id, sample_id, Sample_trimester, Gestationalage_sample) %>%
    head(10)
```

```
# A tibble: 10 x 4
# Groups:   subject_id [2]
   subject_id sample_id Sample_trimester Gestationalage_sample
   <chr>      <chr>                <dbl>                 <dbl>
 1 SUBJ01     SAMP011                  1                  9.00
 2 SUBJ01     SAMP034                  3                 39.8
 3 SUBJ01     SAMP034                  3                 42.1
 4 SUBJ01     SAMP103                  2                 19.9
 5 SUBJ01     SAMP103                  2                 20.0
 6 SUBJ01     SAMP155                  3                 40.0
 7 SUBJ01     SAMP155                  3                 40.5
 8 SUBJ01     SAMP155                  3                 40.7
 9 SUBJ01     SAMP155                  3                 41.6
10 SUBJ02     SAMP113                  3                 38.6
```

33 Exercise 2

Skill: Reshaping data

Select only three columns "sample_id", "Sample_trimester", "Gestationalage_sample", and then reshape from 'long' format to 'wide' format using `pivot_wider`, taking time as the "Sample_trimester".

Expand to see solution

```
b <- ds %>%
    select(sample_id, Sample_trimester, Gestationalage_sample)

b2 <- b %>%
    pivot_wider(id_cols = sample_id, names_from = Sample_trimester, values_from = Gestat
```

```
Warning: Values from `Gestationalage_sample` are not uniquely identified; output will conta
* Use `values_fn = list` to suppress this warning.
* Use `values_fn = {summary_fun}` to summarise duplicates.
* Use the following dplyr code to identify duplicates.
  {data} %>%
    dplyr::group_by(sample_id, Sample_trimester) %>%
    dplyr::summarise(n = dplyr::n(), .groups = "drop") %>%
    dplyr::filter(n > 1L)
```

```
head(b2)
```

```
# A tibble: 6 x 5
  sample_id `1`       `3`       `2`       `NA`
  <chr>     <list>    <list>    <list>    <list>
1 SAMP149   <dbl [3]> <NULL>    <NULL>    <NULL>
2 SAMP037   <dbl [2]> <NULL>    <NULL>    <NULL>
3 SAMP120   <dbl [3]> <NULL>    <NULL>    <NULL>
4 SAMP187   <NULL>    <dbl [1]> <NULL>    <NULL>
5 SAMP082   <NULL>    <NULL>    <dbl [1]> <NULL>
6 SAMP074   <NULL>    <NULL>    <dbl [3]> <NULL>
```

33.1 Comment

View `b2` via the `View(b2)` command in RStudio - it nicely put all the different gestational age observations into one list for each `sample_id` x `Sample_trimester` combination.

34 Exercise 3

Skill: Aggregating data

Make a table showing the proportion of blacks and whites that are controls and cases.

Expand to see solution

```r
prop.table(table(ds$case_control_status, ds$race), margin = 2)
```

```
           B         W     White
0 0.5396825 0.5156250 0.0000000
1 0.4603175 0.4843750 1.0000000
```

34.1 Comment:

The `margin` parameter of the `prop.table` command has to be specified in order to get the desired answer: “1 indicates rows, 2 indicates columns.

```r
prop.table(table(ds$case_control_status, ds$race), margin = 1)
```

```
            B          W      White
0 0.67326733 0.32673267 0.00000000
1 0.64444444 0.34444444 0.01111111
```

```r
prop.table(table(ds$case_control_status, ds$race))
```

```
             B           W       White
0 0.356020942 0.172774869 0.000000000
1 0.303664921 0.162303665 0.005235602
```

Construct more readable tables with labels using `xtabs`

Expand to see solution

34.2 xtabs table with labels

```
prop.table(xtabs(~case_control_status + race, data = ds), margin = 1)
```

```
                   race
case_control_status          B          W      White
                  0 0.67326733 0.32673267 0.00000000
                  1 0.64444444 0.34444444 0.01111111
```

Create a count cross table using Tidyverse commands

Expand to see solution

```
ds %>%
    group_by(case_control_status, race) %>%
    summarize(n = n()) %>%
    spread(race, n)
```

```
`summarise()` has grouped output by 'case_control_status'. You can override
using the `.groups` argument.

# A tibble: 2 x 4
# Groups:   case_control_status [2]
  case_control_status     B     W White
                <dbl> <int> <int> <int>
1                   0    68    33    NA
2                   1    58    31     1
```

```
addmargins(xtabs(~case_control_status + race, data = ds))
```

```
                   race
case_control_status   B   W White Sum
                0    68  33     0 101
                1    58  31     1  90
                Sum 126  64     1 191
```

Create a proportion cross table using Tidyverse commands

Expand to see solution

```
ds %>%
    group_by(case_control_status, race) %>%
    summarize(n = n()) %>%
    mutate(prop = n/sum(n)) %>%
    select(-n) %>%
    spread(race, prop)
```

```
`summarise()` has grouped output by 'case_control_status'. You can override
using the `.groups` argument.

# A tibble: 2 x 4
# Groups:   case_control_status [2]
  case_control_status     B     W   White
                <dbl> <dbl> <dbl>   <dbl>
1                   0 0.673 0.327 NA
2                   1 0.644 0.344  0.0111
```

35 Exercise 4

Skill: Summarizing within groups

Apply the `summary` command to the "Gestationalage_sample" within each "Sample_trimester" group.

Expand to see solution

```r
f <- split(ds[, "Gestationalage_sample"], ds$Sample_trimester)
sapply(f, summary)
```

```
                1        2        3
Min.     4.934325 16.53800 31.44880
1st Qu.  7.838825 18.45761 35.16305
Median   8.565282 19.72388 37.71093
Mean     8.616799 19.83310 37.37827
3rd Qu.  9.193104 20.69576 39.11360
Max.    13.026958 24.60659 42.09340
```

```r
# Or 'tapply' can be used:
tapply(ds$Gestationalage_sample, ds$Sample_trimester, summary)
```

```
$`1`
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
  4.934   7.839   8.565   8.617   9.193  13.027 

$`2`
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
  16.54   18.46   19.72   19.83   20.70   24.61 

$`3`
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
  31.45   35.16   37.71   37.38   39.11   42.09 
```

Note: With `split(x, f)`, any missing values in `f` are dropped together with the corresponding values of `x`.

36 Exercise 5: Recoding data

Approach 1

- Implement our dictionaries using look-up tables
    - Use a named vector.

Skill:: Recoding IDs using a dictionary

Create a new subject ID column named "subjectID" where you have used the `DictPer` named vector to recode the original "subject_id" IDs into integer IDs.

```
head(DictPer)
```

```
SUBJ48 SUBJ46 SUBJ28 SUBJ26 SUBJ49 SUBJ19
    40      2     23     38     10     27
```

Expand to see solution

```
a5 <- ds
a5$ID <- DictPer[a5$subject_id]
a5 %>%
    select(subject_id, ID) %>%
    head
```

```
  subject_id ID
1     SUBJ48 40
2     SUBJ46  2
3     SUBJ28 23
4     SUBJ26 38
5     SUBJ49 10
6     SUBJ48 40
```

```
head(DictPer)
```

```
SUBJ48 SUBJ46 SUBJ28 SUBJ26 SUBJ49 SUBJ19
    40      2     23     38     10     27
```

37 Recoding data

Approach 2

- Implement our dictionaries using left joins

37.1 Comment

I usually prefer to use a merge command like `left_join` to merge in the new IDs into my data frame.

Expand to see solution

```r
key <- data.frame(SubjID = names(DictPer), ID = DictPer)
head(key)
```

```
       SubjID ID
SUBJ48 SUBJ48 40
SUBJ46 SUBJ46  2
SUBJ28 SUBJ28 23
SUBJ26 SUBJ26 38
SUBJ49 SUBJ49 10
SUBJ19 SUBJ19 27
```

```r
b5 <- left_join(ds, key, by = c(subject_id = "SubjID"))
b5 %>%
    select(subject_id, ID) %>%
    head()
```

```
  subject_id ID
1     SUBJ48 40
2     SUBJ46  2
3     SUBJ28 23
4     SUBJ26 38
5     SUBJ49 10
```

```
6      SUBJ48 40
```

38 Exercise 6

Skill: Filtering rows.

Create a data frame `tri1` containing the records for Trimester 1, and a second data frame `tri2` containing the records for Trimester 2.

Expand to see solution

```
tri1 <- ds %>%
    filter(Sample_trimester == 1)
tri1 %>%
    select(subject_id, sample_id, Sample_trimester) %>%
    head()
```

```
  subject_id sample_id Sample_trimester
1     SUBJ48   SAMP149                1
2     SUBJ46   SAMP037                1
3     SUBJ28   SAMP120                1
4     SUBJ48   SAMP149                1
5     SUBJ07   SAMP063                1
6     SUBJ28   SAMP053                1
```

```
tri2 <- ds %>%
    filter(Sample_trimester == 2)
tri2 %>%
    select(subject_id, sample_id, Sample_trimester) %>%
    head()
```

```
  subject_id sample_id Sample_trimester
1     SUBJ49   SAMP082                2
2     SUBJ19   SAMP074                2
3     SUBJ10   SAMP121                2
4     SUBJ22   SAMP184                2
5     SUBJ29   SAMP100                2
6     SUBJ19   SAMP074                2
```

39 Exercise 7

Skill: Selecting columns

Update `tri1` and `tri2` to only contain the three columns "sample_id", "Sample_trimester", "Gestationalage_sample"

Expand to see solution

```
tri1 <- tri1 %>%
    select(sample_id, Sample_trimester, Gestationalage_sample)
head(tri1)
```

```
  sample_id Sample_trimester Gestationalage_sample
1   SAMP149                1              8.094299
2   SAMP037                1              7.146034
3   SAMP120                1              7.122495
4   SAMP149                1              8.473876
5   SAMP063                1              7.510132
6   SAMP053                1              7.446434
```

```
tri2 <- tri2 %>%
    select(sample_id, Sample_trimester, Gestationalage_sample)
head(tri2)
```

```
  sample_id Sample_trimester Gestationalage_sample
1   SAMP082                2              21.89337
2   SAMP074                2              21.26259
3   SAMP121                2              18.29106
4   SAMP184                2              18.76825
5   SAMP100                2              24.48074
6   SAMP074                2              21.24652
```

40 Summary

In summary, this book is a work in progress.

References