HuGen2071 book

Daniel E. Weeks

November 1, 2022

Table of contents

Preface

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

Book web site: https://danieleweeks.github.io/HuGen2071/

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

Created by Daniel E. Weeks

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

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 web site: https://danieleweeks.github.io/HuGen2071/

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

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

```
[1] 4 1
```

```
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
>
> `a$rowNum1 <- NA`
>
> would insert a new row into the data frame `a` full of `NA` values.

Expand to see the answer

```
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)?

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

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

Expand to see the answer

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

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

```
r2
```

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

```
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:

```
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

```
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:

```
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
```

```
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:

```
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
```

```
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
```

```
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

5.1 Load Libraries

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

5.2 Useful RStudio cheatsheet

See the "String manipulation with stringr cheatsheet" at

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

5.3 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.

```
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
TaqMan-Sg2543-190603	C/G

5.4 Discussion Questions

5.4.1 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 5.4: The `clin` data frame

ID	height
1	152
104	172
2112	180

ID	height
2543	163

Hint: Use the `formatC` function.

5.4.2 Answer 1

Expand to see solution

```
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

5.4.3 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 5.6: 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.

5.4.4 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

5.4.5 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 5.9: 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.

5.4.6 Answer 3

Expand to see solution

```
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"
```

```
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:

```
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
```

5.5 Scenario 2

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

```
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

5.5.1 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 5.12: The `joint` data frame

ID	SBP
SG0001	120
SG0104	111

ID	SBP
SG2112	125
SG2543	119
RP5002	121
RP5012	118
RP5113	112
RP5213	142

5.5.2 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

6 R Functions Excercise

6.1 Load Libraries

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

6.2 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"
```

6.3 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"))
}
```

6.4 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)
f1 <- read.table("data/dataset1.txt",sep="\t",header=TRUE)
results$dataset[1] <- "dataset1"
results$N <- nrow(f1)
results$mean[1] <- mean(f1$trait)
results$median[1] <- median(f1$trait)
results$var[1] <- var(f1$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
```

```
f2 <- read.table("data/dataset2.txt",sep="\t",header=TRUE)
results$dataset[2] <- "dataset2"
results$N <- nrow(f2)
results$mean[2] <- mean(f2$trait)
results$median[2] <- median(f2$trait)
results$var[2] <- var(f2$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
```

```
f3 <- 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.

6.4.1 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"
```

6.4.2 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

6.4.3 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

6.4.4 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)
}
```

6.4.5 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.

6.4.6 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
```

6.4.7 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)
  }
```

7 R Tidyverse Exercise

7.1 Load Libraries

Load the `tidyverse` packages

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

7.2 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.

7.3 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.

7.4 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  1.61   3.79  -0.855
2 2009-01-02  1.91  -1.58  -6.94
3 2009-01-03  0.481  1.80  -1.84
4 2009-01-04 -0.382  1.65  -3.17
5 2009-01-05  0.279  1.30  -4.66
6 2009-01-06  0.854 -0.471 -2.73
```

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

```
# A tibble: 6 x 3
  time       stock  price
  <date>     <chr>  <dbl>
1 2009-01-01 X      1.61
2 2009-01-02 X      1.91
3 2009-01-03 X      0.481
4 2009-01-04 X     -0.382
```

```
5 2009-01-05 X      0.279
6 2009-01-06 X      0.854
```

7.5 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      1.61
2 2009-01-01 Y      3.79
3 2009-01-01 Z     -0.855
4 2009-01-02 X      1.91
5 2009-01-02 Y     -1.58
6 2009-01-02 Z     -6.94
```

7.6 WHO TB data

Question: How would we convert this to tidy form?

```
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

```
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?

```
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
```

7.7 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
```

7.8 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

8 R Recoding Reshaping Exercise

8.1 Load Libraries

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

8.2 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"
```

8.3 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

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

```
[1] 72
```

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

Expand to see solution

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

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

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

```
[1] 72
```

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

```
[1] 71
```

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

```
[1] 71
```

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

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

```r
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
```

8.4 Checking for duplicates

How do we return every row that contains a duplicate?

```
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
```

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

```
  ID c2
2  1  2
```

8.5 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
```

8.6 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
```

8.7 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
```

8.7.1 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

8.7.2 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
```

8.7.3 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
```

8.8 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>
```

8.8.1 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.

8.9 Exercise 3

Skill: Aggregating data

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

Expand to see solution

```
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
```

8.9.1 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.

```
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
```

```
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

8.9.2 `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
```

8.10 Exercise 4

Skill: Summarizing within groups

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

Expand to see solution

```
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
```

```
# 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`.

8.11 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
```

8.12 Recoding data

Approach 2

- Implement our dictionaries using left joins

8.12.1 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

```
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
```

```
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
```

8.13 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
```

8.14 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
```

9 R Merging Exercise

9.1 Load Libraries

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

9.2 Input data

Let's load the synthetic simulated Project 1 data and associated data dictionary:

```
load("data/project1.RData", verbose = TRUE)
```

```
Loading objects:
  ds
  dd
```

9.3 Select a subset of subject-level fields

Set up a data frame 'a' that has these subject-level fields: "subject_id" "maternal_age_delivery" "case_control_status" "prepregnancy_BMI"

```
a <- ds %>%
    select("subject_id", "maternal_age_delivery", "case_control_status", "prepregnancy_BMI
    arrange(subject_id)
```

```
select: dropped 20 variables (sample_id, Sample_trimester, Gestationalage_sample, strata, ra
```

```
head(a, 10)
```

```
   subject_id maternal_age_delivery case_control_status prepregnancy_BMI
1      SUBJ01              20.01345                   0         45.29379
2      SUBJ01              20.01345                   0         45.29379
3      SUBJ01              20.01345                   0         45.29379
4      SUBJ01              20.01345                   0         45.29379
5      SUBJ01              20.01345                   0         45.29379
6      SUBJ01              20.01345                   0         45.29379
7      SUBJ01              20.01345                   0         45.29379
8      SUBJ01              20.01345                   0         45.29379
9      SUBJ01              20.01345                   0         45.29379
10     SUBJ02              22.22541                   1         41.63679
```

```
tail(a)
```

```
    subject_id maternal_age_delivery case_control_status prepregnancy_BMI
186     SUBJ55              23.79660                   0         30.73757
187     SUBJ56              20.77767                   1         32.30103
188     SUBJ56              20.77767                   1         32.30103
189     SUBJ56              20.77767                   1         32.30103
190     SUBJ56              20.77767                   1         32.30103
191     SUBJ56              20.77767                   1         32.30103
```

9.4 Unique records

The data were given to us in a way that repeated subject-level information, once for each sample from each individual subject.

From your data frame 'a' select only the unique records, creating data frame `b`.

Expand to see solution

```
b <- unique(a)
head(b)
```

```
   subject_id maternal_age_delivery case_control_status prepregnancy_BMI
1      SUBJ01              20.01345                   0         45.29379
10     SUBJ02              22.22541                   1         41.63679
13     SUBJ03              20.80036                   1         32.55473
15     SUBJ04              21.94422                   0         35.09978
19     SUBJ05              20.18760                   0         41.85877
```

```
22     SUBJ06              25.70581                    0          38.02936
```

```
  b1 <- a %>%
      distinct()
```

```
distinct: removed 137 rows (72%), 54 rows remaining
```

```
  all.equal(b, b1)
```

```
[1] "Attributes: < Component \"row.names\": Mean relative difference: 0.7157633 >"
```

```
  all.equal(b, b1, check.attributes = FALSE)
```

```
[1] TRUE
```

```
  head(rownames(b))
```

```
[1] "1"  "10" "13" "15" "19" "22"
```

```
  head(rownames(b1))
```

```
[1] "1" "2" "3" "4" "5" "6"
```

```
  # Reset row names
  rownames(b) <- NULL
  rownames(b1) <- NULL
  all.equal(b, b1)
```

```
[1] TRUE
```

9.4.1 Comment

It is better to apply `unique` to the whole data frame, not just to the `subject_id` column, as that ensures that you are selecting whole records that are unique across all of their columns.

```
  (ex1 <- data.frame(ID = c(1, 1, 1, 2), trait = c(10, 9, 9, 11)))
```

```
  ID trait
1  1    10
2  1     9
3  1     9
4  2    11
```

```
unique(ex1)
```

```
  ID trait
1  1    10
2  1     9
4  2    11
```

```
ex1 %>%
    distinct()
```

```
distinct: removed one row (25%), 3 rows remaining
```

```
  ID trait
1  1    10
2  1     9
3  2    11
```

9.5 Check that the `subject_id`'s are now not duplicated

Are the `subject_id`'s unique?

Expand to see solution

```
sum(duplicated(b$subject_id))
```

```
[1] 0
```

```
b %>%
    group_by(subject_id) %>%
    filter(n() > 1)
```

```
group_by: one grouping variable (subject_id)
```

```
filter (grouped): removed all rows (100%)

# A tibble: 0 x 4
# Groups:   subject_id [0]
# ... with 4 variables: subject_id <chr>, maternal_age_delivery <dbl>,
#   case_control_status <dbl>, prepregnancy_BMI <dbl>
```

9.6 Create random integer IDs

Create a new column `ID` containing randomly chosen integer IDs; this is necessary to de-identify the data. To do this, use the `sample` command, sampling integers from 1 to the number of rows in data frame `b`.

Expand to see solution

```
set.seed(10234)
b$ID <- sample(c(1:nrow(b)), replace = FALSE)
head(b %>%
    select(subject_id, ID))
```

```
select: dropped 3 variables (maternal_age_delivery, case_control_status, prepregnancy_BMI)

  subject_id ID
1     SUBJ01  6
2     SUBJ02  4
3     SUBJ03 41
4     SUBJ04 14
5     SUBJ05 51
6     SUBJ06 37
```

```
sum(duplicated(b$ID))
```

```
[1] 0
```

9.7 Merge in new phenotype information

The PI has sent you new trait data for your subjects.

```
new <- read_tsv("data/newtrait.tsv")
```

```
Rows: 54 Columns: 2
-- Column specification --------------------------------------------------------
Delimiter: "\t"
chr (1): subject_id
dbl (1): trait

i Use `spec()` to retrieve the full column specification for this data.
i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
head(new)
```

```
# A tibble: 6 x 2
  subject_id trait
  <chr>      <dbl>
1 SUBJ48      130.
2 SUBJ46      104.
3 SUBJ28      125.
4 SUBJ26      108.
5 SUBJ49      129.
6 SUBJ19      117.
```

```
dim(new)
```

```
[1] 54  2
```

```
dim(b)
```

```
[1] 54  5
```

Carefully merge this in using tidyverse commands.

If you notice any problems with this merge, prepare a report for the PI detailing what you noticed and what you'd like to ask the PI about.

9.8 Always be careful when merging.

- *Always check for duplicated IDs* before doing the merge.
- Always check that your ID columns do not contain any missing values.
- Check that the values in the ID columns (e.g., the keys) match.
  - Can use an 'anti_join' to check this.
  - Inconsistencies in the values of the keys can be hard to fix.
- *Always check the dimensions* to make sure the merged object has the expected number of rows and columns.
- Always explicitly name the keys you are merging on.
  - If you don't name them, then the join command will use all variables in common across `x` and `y`.

9.9 Merge in new phenotype information

Carefully merge in the new data in using tidyverse commands.

If you notice any problems with this merge, prepare a report for the PI detailing what you noticed and what you'd like to ask the PI about.

Expand to see solution

```
# Check for duplicated IDs
sum(duplicated(b$subject_id))
```

```
[1] 0
```

```
sum(duplicated(new$subject_id))
```

```
[1] 1
```

```
# Which one is duplicated
new %>%
    group_by(subject_id) %>%
    mutate(n = n()) %>%
    filter(n > 1)
```

```
group_by: one grouping variable (subject_id)
```

```
mutate (grouped): new variable 'n' (integer) with 2 unique values and 0% NA

filter (grouped): removed 52 rows (96%), 2 rows remaining

# A tibble: 2 x 3
# Groups:   subject_id [1]
  subject_id trait     n
  <chr>      <dbl> <int>
1 SUBJ09      115.     2
2 SUBJ09      115.     2
```

```
# Check for missing IDs
sum(is.na(b$subject_id))
```

```
[1] 0
```

```
sum(is.na(new$subject_id))
```

```
[1] 0
```

```
# Check the dimensions
dim(b)
```

```
[1] 54  5
```

```
dim(new)
```

```
[1] 54  2
```

```
b2 <- left_join(b, new, by = "subject_id")
```

```
left_join: added one column (trait)

           > rows only in x    2

           > rows only in y  ( 1)

           > matched rows     53    (includes duplicates)

           >                 ====
```

```
           > rows total       55
```

```
head(b2)
```

```
  subject_id maternal_age_delivery case_control_status prepregnancy_BMI ID
1     SUBJ01              20.01345                    0         45.29379  6
2     SUBJ02              22.22541                    1         41.63679  4
3     SUBJ03              20.80036                    1         32.55473 41
4     SUBJ04              21.94422                    0         35.09978 14
5     SUBJ05              20.18760                    0         41.85877 51
6     SUBJ06              25.70581                    0         38.02936 37
     trait
1 138.1346
2 113.3822
3 116.0071
4 127.5887
5 113.8754
6 110.7376
```

```
dim(b2)
```

```
[1] 55  6
```

```
b3 <- full_join(b, new, by = "subject_id")
```

```
full_join: added one column (trait)

           > rows only in x    2

           > rows only in y    1

           > matched rows     53    (includes duplicates)

           >                 ====

           > rows total       56
```

```
dim(b3)
```

```
[1] 56  6
```

anti_join() return all rows from x without a match in y.

```
  anti_join(b, new, by = "subject_id")
```

```
anti_join: added no columns

           > rows only in x    2

           > rows only in y  ( 1)

           > matched rows    (52)

           >                 ====

           > rows total        2
```

```
  subject_id maternal_age_delivery case_control_status prepregnancy_BMI ID
1     SUBJ18              27.54075                    1         44.14983 18
2     SUBJ24              21.93645                    0         31.73762  9
```

```
  anti_join(new, b, by = "subject_id")
```

```
anti_join: added no columns

           > rows only in x    1

           > rows only in y  ( 2)

           > matched rows    (53)

           >                 ====

           > rows total        1
```

```
# A tibble: 1 x 2
  subject_id trait
  <chr>      <dbl>
1 SUBJ00      127.
```

10 R Graphics Exercise

10.1 Load Libraries

```
library(tidyverse)
library(ggforce)
# library(tidylog)
# Set the default font to be a bit larger:
theme_set(theme_gray(base_size = 18))
```

10.2 Exercise 1

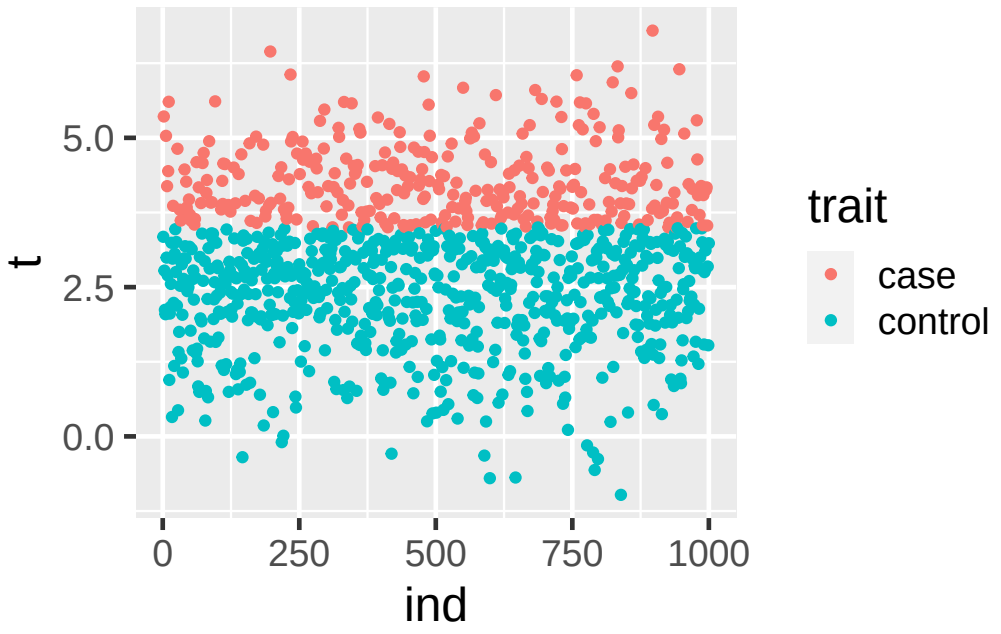
Read in and set up the data set `b`, a cleaned version of our simulated data set:

```
a <- read.csv("data/study1.csv")
a$ind <- seq_along(a$t)
b <- a[-c(1001:1004),]
b$g.f <- factor(b$g)
b$geno <- paste(b$all1,b$all2,sep="/")
```

Using ggplot and data set `b`, plot `ind` vs. `t`, coloring by case-control status (`trait`). What do you observe about the data?

Expand to see solution

```
ggplot(data=b, aes(x=ind, y=t, color=trait)) +
   geom_point()
```

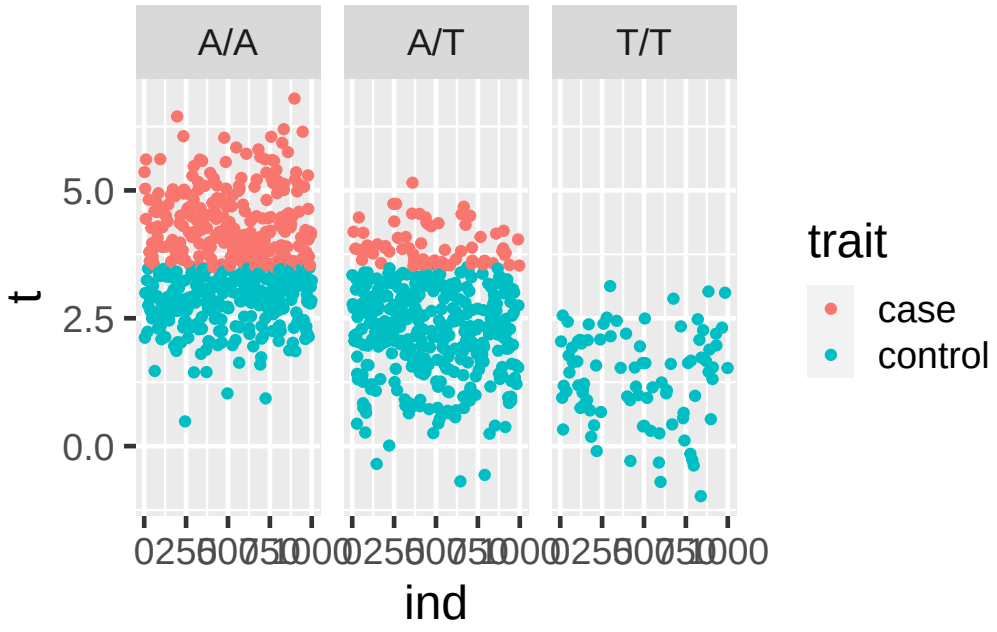


10.3 Exercise 2

Using ggplot, plot `ind` vs. `t`, coloring by case-control status (`trait`) and faceting by `geno`. What do you observe about the data?

Expand to see solution

```
ggplot(data=b, aes(x=ind, y=t, color=trait)) +
  geom_point() +
  facet_grid(~ geno)
```



10.4 Always plot your data

```
library(tidyverse)
d <- read_tsv("data/example.tsv")
```

```
New names:
Rows: 142 Columns: 26
-- Column specification
-------------------------------------------------------- Delimiter: "\t" dbl
(26): x...1, y...2, x...3, y...4, x...5, y...6, x...7, y...8, x...9, y.....
i Use `spec()` to retrieve the full column specification for this data. i
Specify the column types or set `show_col_types = FALSE` to quiet this message.
* `x` -> `x...1`
* `y` -> `y...2`
* `x` -> `x...3`
* `y` -> `y...4`
* `x` -> `x...5`
* `y` -> `y...6`
* `x` -> `x...7`
* `y` -> `y...8`
* `x` -> `x...9`
```

```
* `y` -> `y...10`
* `x` -> `x...11`
* `y` -> `y...12`
* `x` -> `x...13`
* `y` -> `y...14`
* `x` -> `x...15`
* `y` -> `y...16`
* `x` -> `x...17`
* `y` -> `y...18`
* `x` -> `x...19`
* `y` -> `y...20`
* `x` -> `x...21`
* `y` -> `y...22`
* `x` -> `x...23`
* `y` -> `y...24`
* `x` -> `x...25`
* `y` -> `y...26`
```

```
n1 <- rep(c("x","y"), 13)
n2 <- c("","",rep("_",24))
n3 <- c("", "", c(sort(rep(c(1:12), 2))))
names(d) <- paste0(n1,n2,n3)
names(d)
```

```
 [1] "x"    "y"    "x_1"  "y_1"  "x_2"  "y_2"  "x_3"  "y_3"  "x_4"  "y_4"
[11] "x_5"  "y_5"  "x_6"  "y_6"  "x_7"  "y_7"  "x_8"  "y_8"  "x_9"  "y_9"
[21] "x_10" "y_10" "x_11" "y_11" "x_12" "y_12"
```

10.5 Similar regression lines

These three data sets have very similar regression lines:

```
summary(lm(x ~ y, data=d)) %>%  coef()
```

```
             Estimate Std. Error    t value     Pr(>|t|)
(Intercept) 56.17563819 2.87986960 19.5063131 9.435087e-42
y           -0.03991951 0.05250204 -0.7603419 4.483288e-01
```

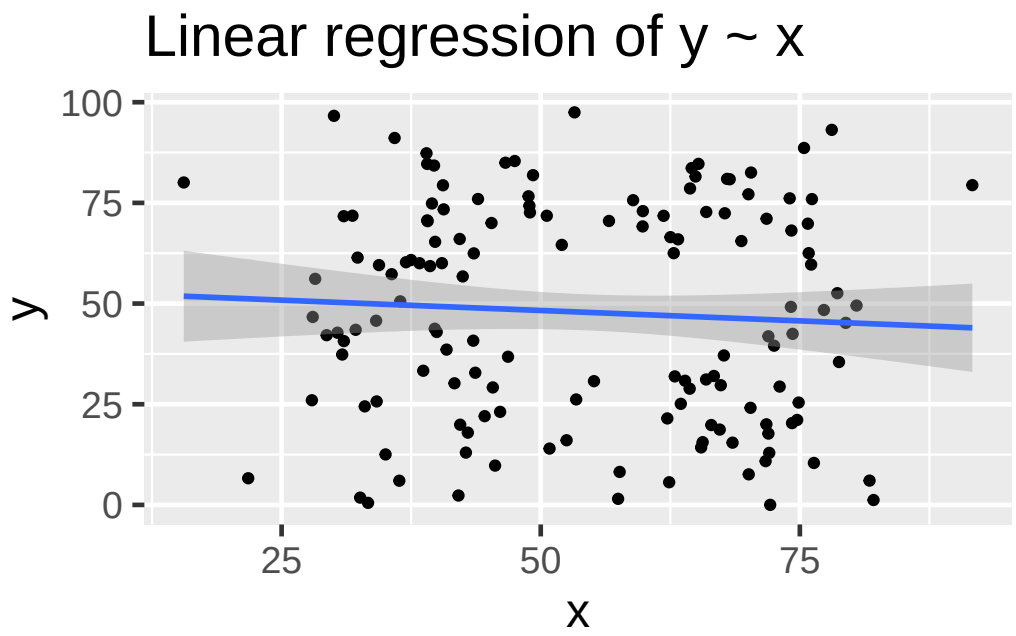
```
  summary(lm(x_1 ~ y_1, data=d)) %>%  coef()
```

```
               Estimate Std. Error    t value     Pr(>|t|)
(Intercept) 56.31108156 2.87906158 19.5588319 7.158847e-42
y_1         -0.04269949 0.05249244 -0.8134407 4.173467e-01
```

```
  summary(lm(x_3 ~ y_3, data=d)) %>%  coef()
```

```
               Estimate Std. Error    t value     Pr(>|t|)
(Intercept) 56.18271411 2.87924135 19.5130270 9.107718e-42
y_3         -0.04012859 0.05249468 -0.7644316 4.458966e-01
```

```
  ggplot(d,aes(x=x,y=y)) + geom_point() +
    geom_smooth(method="lm") + ggtitle("Linear regression of y ~ x")
```



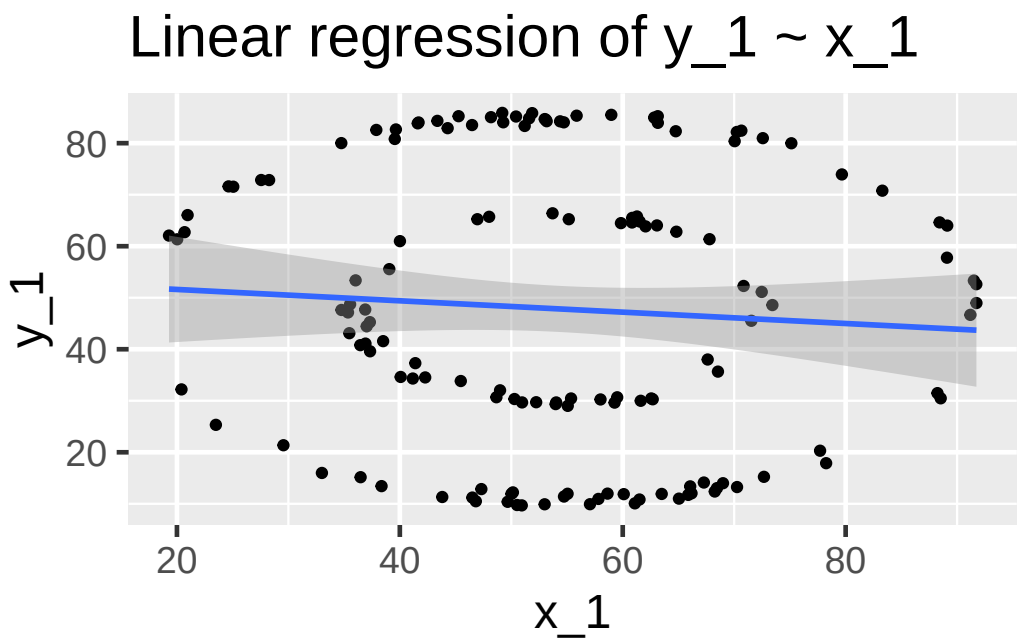
Now try this:

```
ggplot(d,aes(x=x_1,y=y_1)) + geom_point() +
  geom_smooth(method="lm")
```

Expand to see solution

```
ggplot(d,aes(x=x_1,y=y_1)) + geom_point() +
  geom_smooth(method="lm") + ggtitle("Linear regression of y_1 ~ x_1")
```

```
`geom_smooth()` using formula 'y ~ x'
```



And now try this:

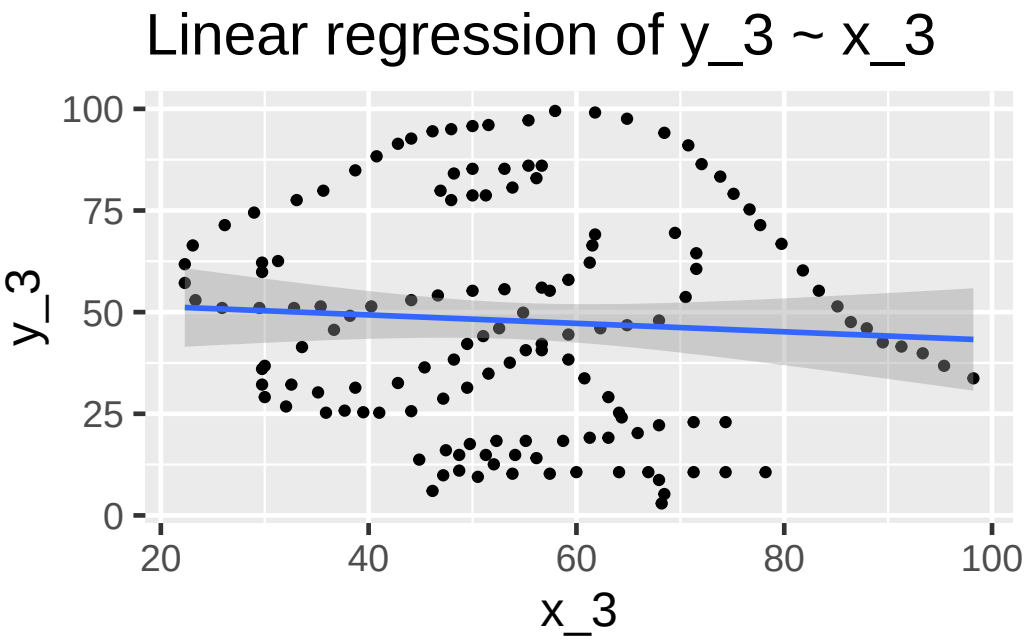
```
ggplot(d,aes(x=x_3,y=y_3)) + geom_point() +
  geom_smooth(method="lm")
```

Expand to see solution

10.5.1 Always plot your data!

```
ggplot(d,aes(x=x_3,y=y_3)) + geom_point() +
  geom_smooth(method="lm") + ggtitle("Linear regression of y_3 ~ x_3")
```

```
`geom_smooth()` using formula 'y ~ x'
```



10.6 Always plot your data

```
f <- read_tsv("data/BoxPlots.tsv")
# Delete the first column
f <- f[,-1]
head(f)
```

```
# A tibble: 6 x 5
   left lines normal right split
  <dbl> <dbl>  <dbl> <dbl> <dbl>
1 -9.77 -9.77  -9.76 -9.76 -9.77
2 -9.76 -9.74  -9.72 -9.05 -9.77
3 -9.75 -9.77  -9.68 -8.51 -9.77
4 -9.77 -9.77  -9.64 -8.24 -9.77
5 -9.76 -9.77  -9.6  -8.82 -9.77
6 -9.77 -9.76  -9.56 -8.07 -9.76
```

Stacking vectors concatenates multiple vectors into a single vector along with a factor indicating where each observation originated.

```
head(stack(f),2)
```

```
     values  ind
1 -9.769107 left
2 -9.763145 left
```

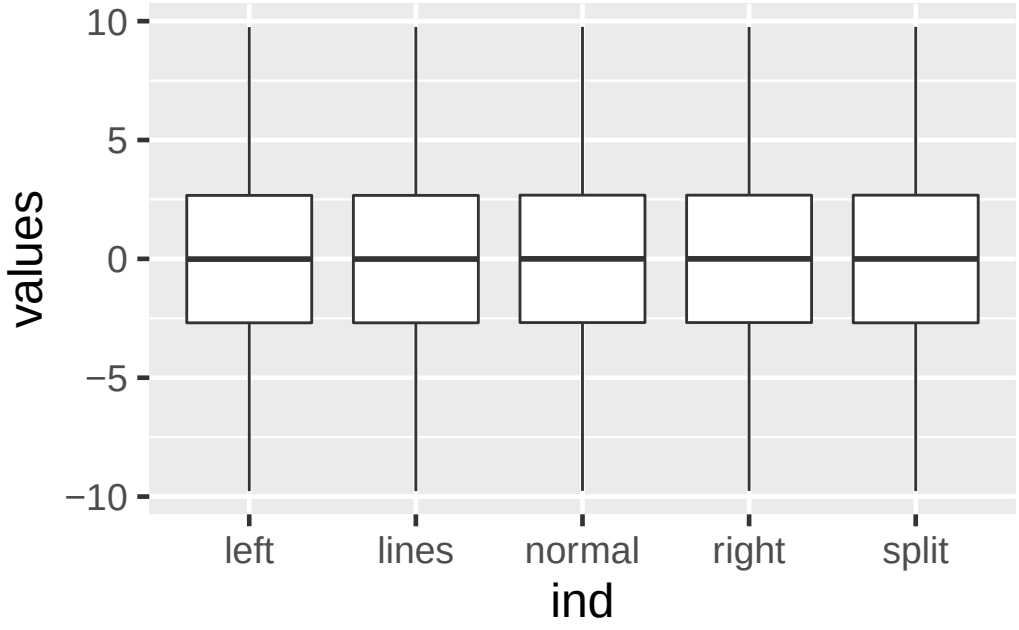
Now try this:

```
ggplot(stack(f), aes(x = ind, y = values)) +
  geom_boxplot()
```

Expand to see solution

10.7 Identical box plots

These data have essentially identical box plots.



10.8 Boxplots

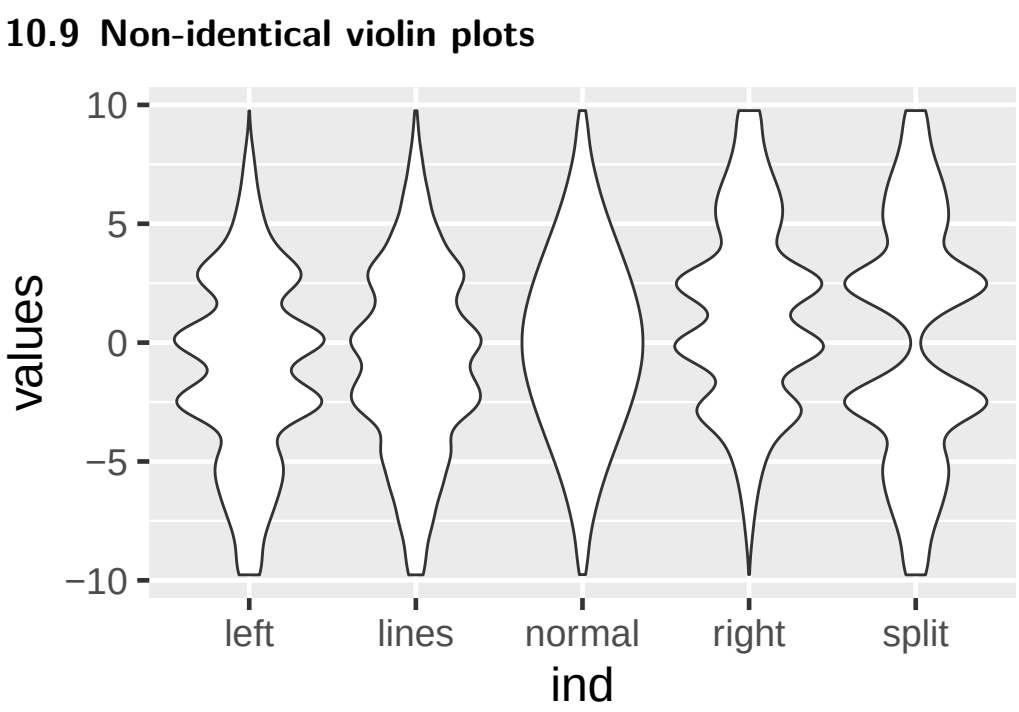
While the box plots are identical, box plots may not tell the whole story.

Let's try violin plots instead:

```
ggplot(stack(f), aes(x = ind, y = values)) +
  geom_violin()
```

A violin plot is a mirrored density plot.

Expand to see solution



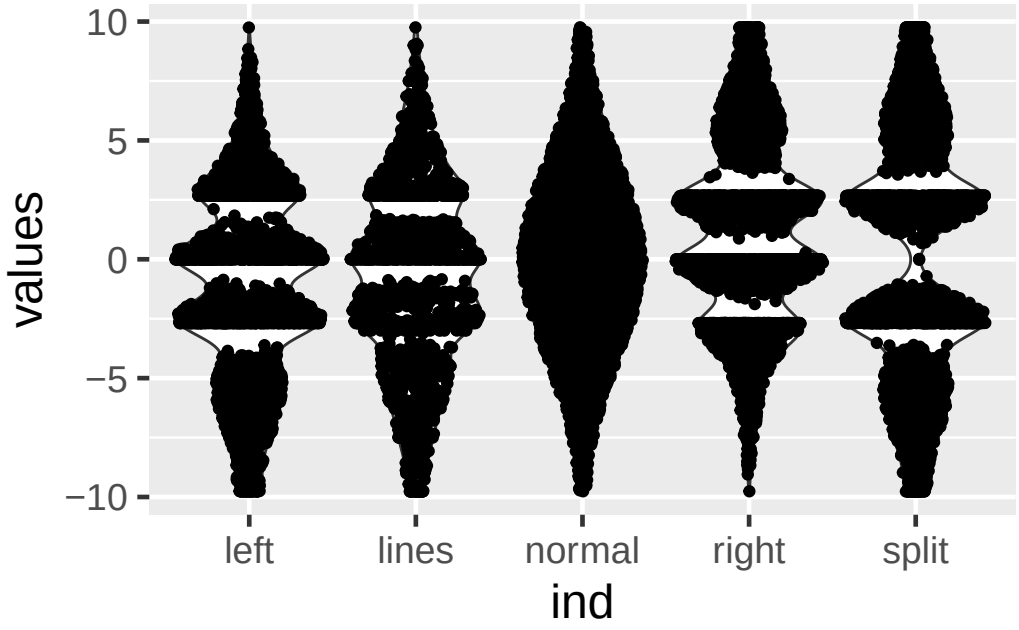
10.10 Sina plots

Sidiropoulos, N., Sohi, S.H., Rapin, N., and Bagger, F.O. (2015). SinaPlot: an enhanced chart for simple and truthful representation of single observations over multiple classes. bioRxiv 28191. https://www.biorxiv.org/content/early/2015/10/02/028191

```
library(ggforce)
ggplot(stack(f), aes(x = ind, y = values)) +
   geom_violin() + geom_sina()
```

Expand to see solution

10.11 Sina plots



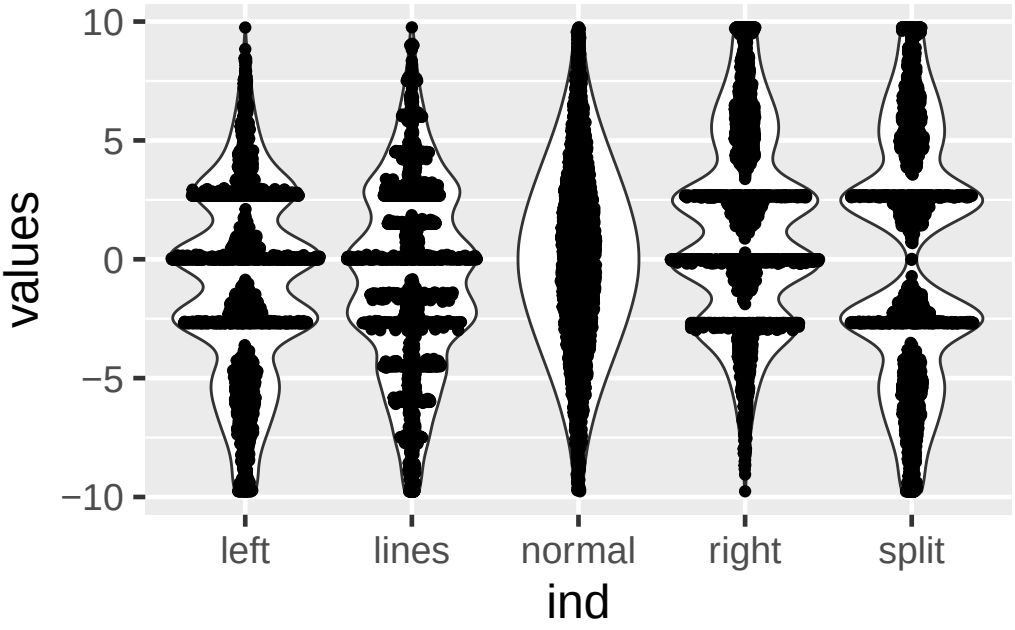
10.12 Sina plots

`method == "counts"`: The borders are defined by the number of samples that occupy the same bin.

```
ggplot(stack(f), aes(x = ind, y = values)) +
   geom_violin() + geom_sina(method="count")
```

Expand to see solution

10.13 Sina plots

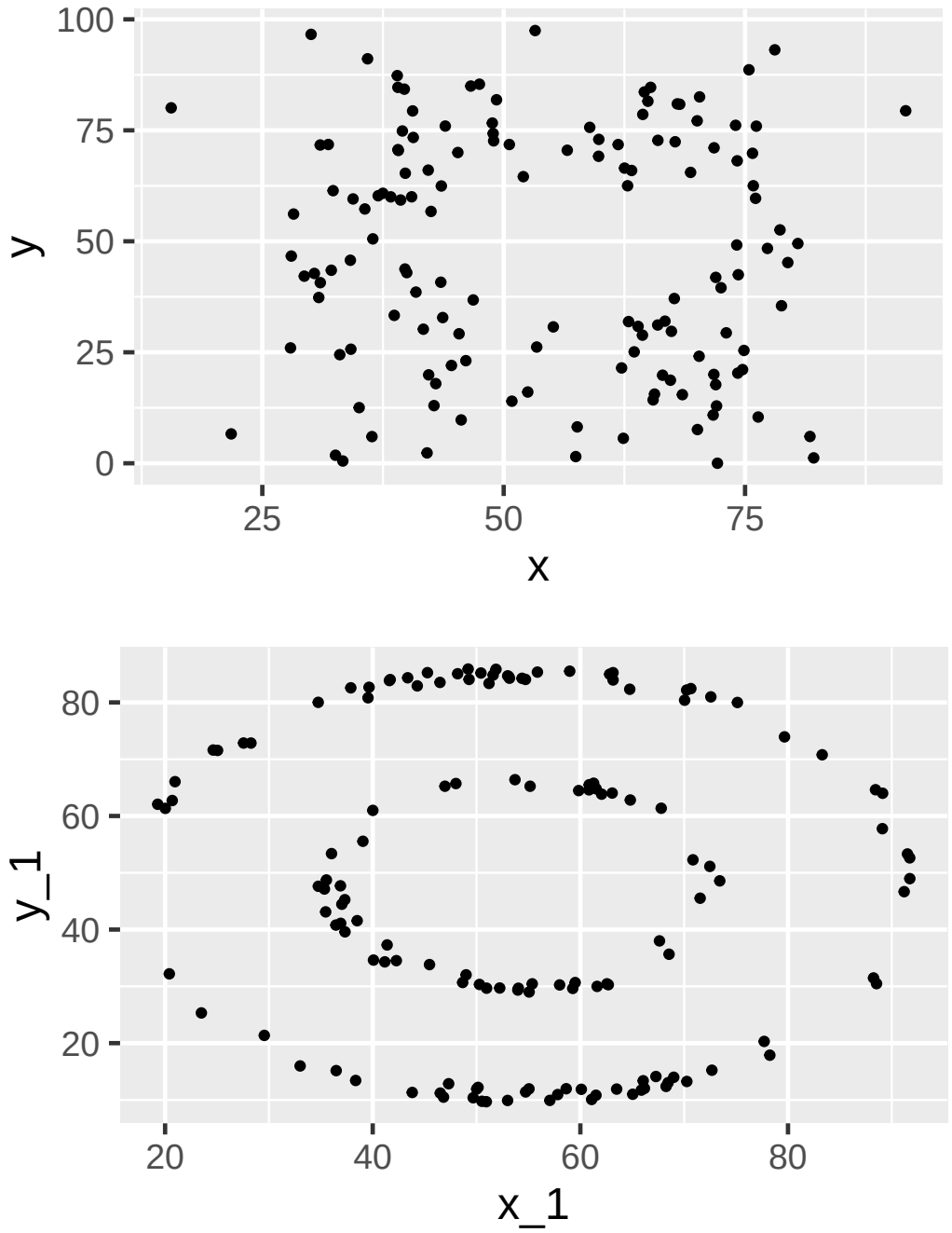


10.14 Drawing multiple graphs

Sometimes we'd like to draw multiple plots, looping across variables. Doing this within an R Markdown or Quarto Markdown document using `ggplot2` is tricky. See https://dplyr.tidyverse.org/articles/programming.html and https://r4ds.hadley.nz/functions.html#plot-functions for details.

Here's one way to do this - this example code will generate two scatter plots:

```
x.names <- c("x","x_1")
y.names <- c("y", "y_1")
for (i in 1:2) {
  x.nam <- sym(x.names[i])
  y.nam <- sym(y.names[i])
  print(ggplot(data=d, aes(x = {{ x.nam }},
                           y = {{ y.nam }})) +
          geom_point())
}
```



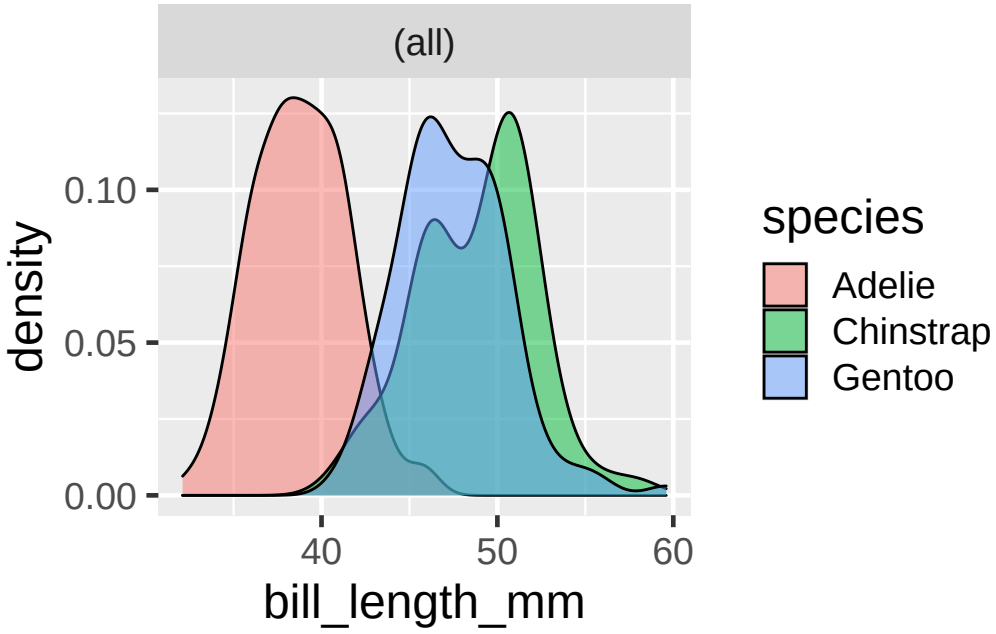
10.15 Writing ggplot functions

See https://r4ds.hadley.nz/functions.html#plot-functions

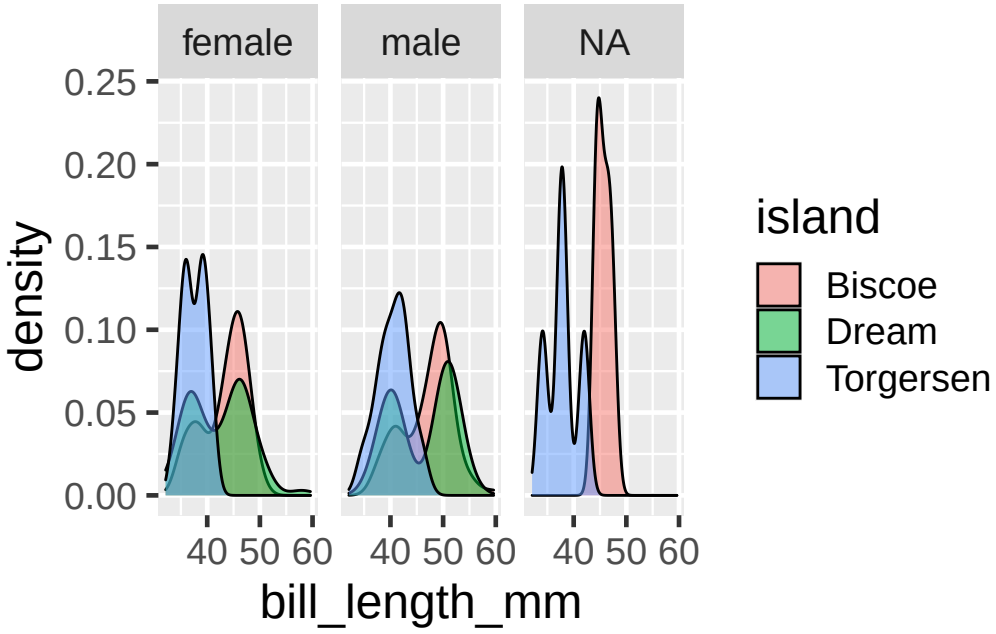
```
library(palmerpenguins)

PlDensity <- function(fill, ...) {
  ggplot(penguins %>% filter(!is.na(bill_length_mm)),
         aes(bill_length_mm, fill = {{ fill }})) +
    geom_density(alpha = 0.5) +
    facet_wrap(vars(...))
}
```

Example from: https://twitter.com/yutannihilat_en/status/1574387230025875457?s=20&t=FLbwErwEKQKWtKIGufDLIQ

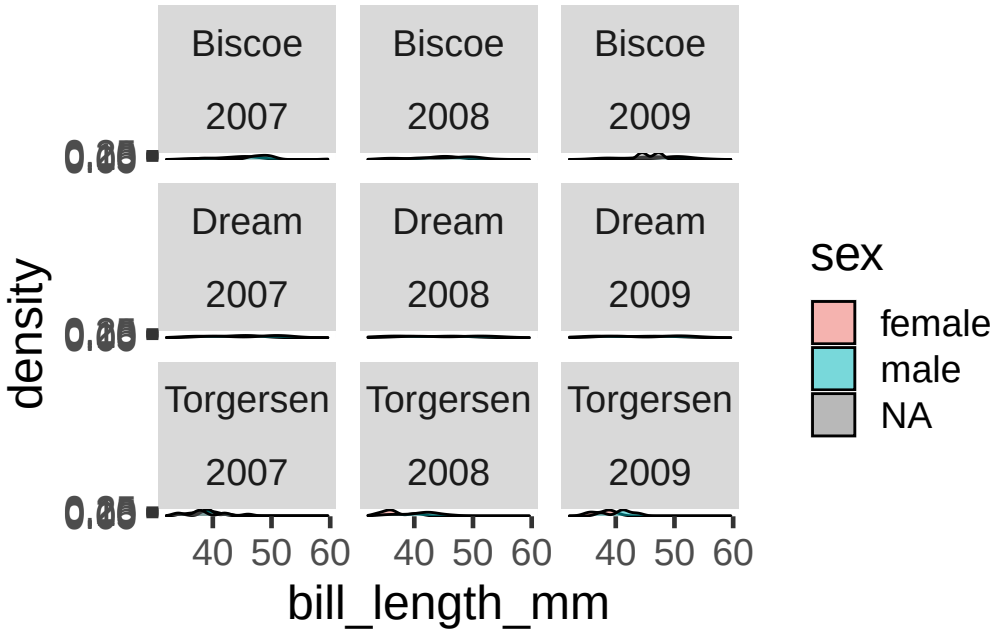
```
PlDensity(species)
```




```
PlDensity(island, sex) %>% print() %>% suppressWarnings()
```




```
PlDensity(sex, island, year) %>% print() %>% suppressWarnings()
```



10.16 Exercise 3

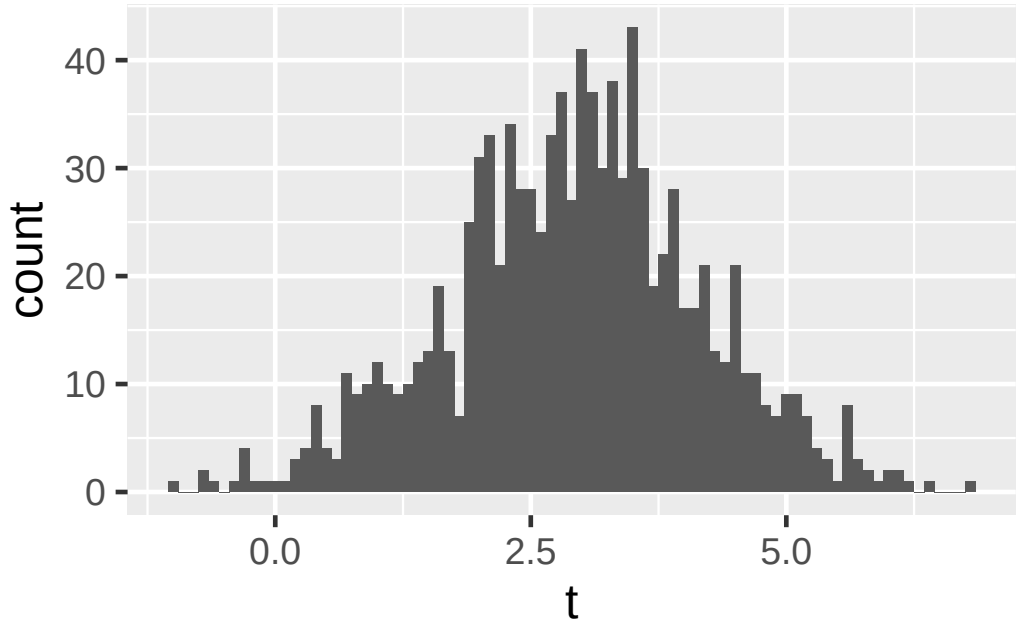
Consider this example code:

```
histogram <- function(df, var, binwidth) {
  df |>
    ggplot(aes({{ var }})) +
    geom_histogram(binwidth = binwidth)
}
```

From: https://twitter.com/hadleywickham/status/1574373127349575680?s=20&t=FLbwErwEKQKWtKIGufDLIQ

When applied to the quantitative trait `t` from the data frame `b`, this generates this histogram:

```
histogram(b, t, 0.1)
```



10.16.1 Exercise

After reading the example above, extend the `histogram` function to allow facetting and use it to draw a histogram of the quantitative trait `t` facetted by `geno` using the data set `b` that we set up above.

Hints

- See https://r4ds.hadley.nz/functions.html#facetting
- Use the `vars()` function

Expand to see solution

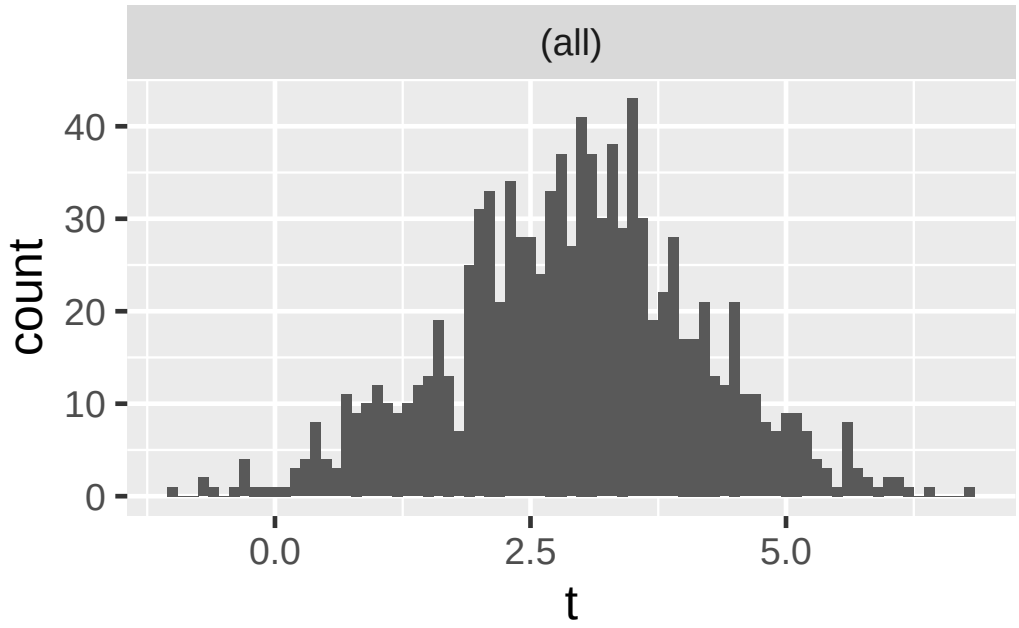
Hadley Wickham states:
You have to use the vars() syntax

```
foo <- function(x) {
  ggplot(mtcars) +
    aes(x = mpg, y = disp) +
    geom_point() +
    facet_wrap(vars({{ x }}))
}
```

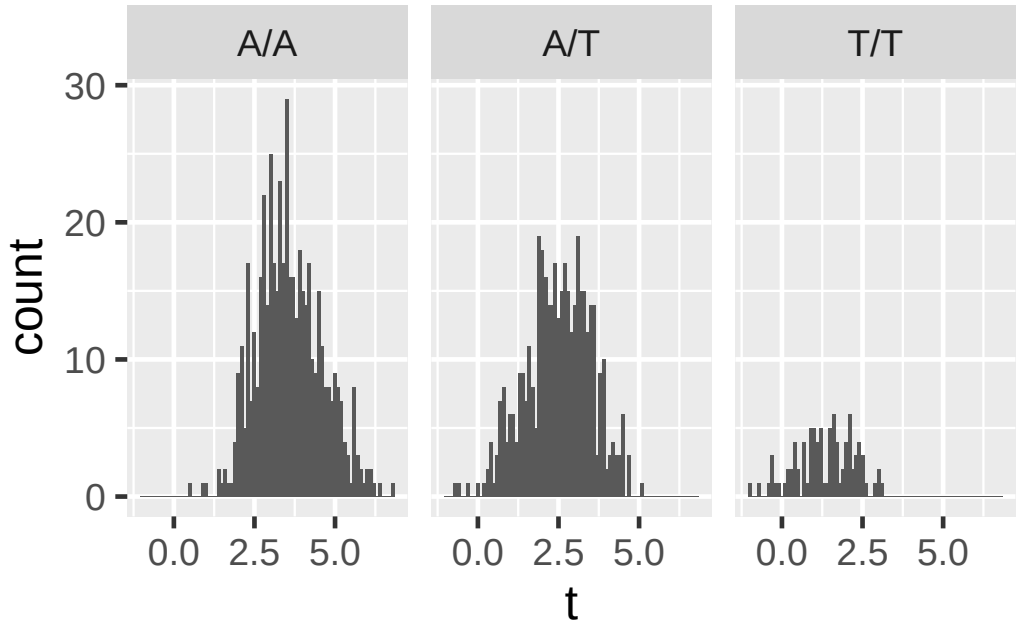
Tweet: https://twitter.com/hadleywickham/status/1574380137524887554?s=20&t=FLbwErwEKQKWtKIGufDLIQ

```
histogram <- function(df, var, binwidth, grp) {
  df |>
    ggplot(aes({{ var }})) +
    geom_histogram(binwidth = binwidth) +
    facet_wrap(vars({{ grp }}))
}
```

```
histogram(b, t, 0.1)
```




```
histogram(b, t, 0.1, geno)
```



10.17 Source of data

Illustrative data sets from https://www.autodeskresearch.com/publications/samestats

11 R Reordering Exercise

11.1 Load Libraries

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

11.2 Create some example data

Here we set up a data dictionary `dd` and some corresponding data `ds`. However, it is better if the order of the rows in the data dictionary `dd` match the order of the columns in the data `ds`.

```
set.seed(1562345)
# Set up a data dictionary
dd <- data.frame(VARNAME = sample(letters, 26), TYPE = "numeric")
# Set up data
ds <- as.data.frame(t(dd %>%
    arrange(VARNAME)))
names(ds) <- letters
rownames(ds) <- NULL
ds[1, ] <- rnorm(26)
ds[2, ] <- runif(26)
ds$ID <- c(1, 2)
ds <- ds %>%
    select(ID, everything())
```

```
select: columns reordered (ID, a, b, c, d, …)
```

```
# Randomly rearrange the columns
idx <- sample(letters, 26)
idx <- c("ID", idx)
ds <- ds %>%
```

```
    select(all_of(idx))
```

```
select: columns reordered (ID, b, z, a, p, …)
```

```
dd <- bind_rows(dd, data.frame(VARNAME = "ID", TYPE = "string"))
dim(dd)
```

```
[1] 27  2
```

```
head(dd)
```

```
  VARNAME    TYPE
1       c numeric
2       m numeric
3       f numeric
4       e numeric
5       a numeric
6       d numeric
```

```
dim(ds)
```

```
[1]  2 27
```

```
head(ds[1:3])
```

```
  ID                 b                 z
1  1  1.02333343074042  0.47956883003516
2  2 0.858655267162248 0.136965574463829
```

```
names(ds)
```

```
 [1] "ID" "b"  "z"  "a"  "p"  "f"  "u"  "m"  "q"  "n"  "d"  "o"  "s"  "k"  "e"
[16] "x"  "c"  "h"  "i"  "g"  "j"  "r"  "t"  "y"  "l"  "w"  "v"
```

11.3 Task: Reorder rows in `dd` in the order of `ds`'s columns

```
colnames(ds)
```

```
 [1] "ID" "b"  "z"  "a"  "p"  "f"  "u"  "m"  "q"  "n"  "d"  "o"  "s"  "k"  "e"
[16] "x"  "c"  "h"  "i"  "g"  "j"  "r"  "t"  "y"  "l"  "w"  "v"
```

```
dd$VARNAME
```

```
 [1] "c"  "m"  "f"  "e"  "a"  "d"  "v"  "h"  "k"  "t"  "p"  "j"  "l"  "x"  "w"
[16] "y"  "b"  "o"  "s"  "r"  "i"  "z"  "u"  "n"  "g"  "q"  "ID"
```

This assumes that every row of `dd` is in `colnames(ds)` and every `colnames(ds)` value is represented in `dd`. Perhaps that should be checked first.

11.4 Assumption Check Question

How would you check that every variable listed in the data dictionary `dd` is named in `colnames(ds)` and every `colnames(ds)` value is represented in the data dictionary `dd`?

Expand to see solution

```
table(dd$VARNAME %in% colnames(ds))
```

```
TRUE
  27
```

```
table(colnames(ds) %in% dd$VARNAME)
```

```
TRUE
  27
```

Note that we should also check to see if the `VARNAME`'s are unique and the `colnames` of `ds` are unique.

```
sum(duplicated(dd$VARNAME))
```

```
[1] 0
```

```r
sum(duplicated(colnames(ds)))
```

```
[1] 0
```

11.5 Task: Reorder rows in `dd` to match the order of the columns in `ds`

Task: Reorder rows in the data dictionary `dd` to match the order of the columns in the data `ds`

- What are various ways you could rearrange the rows of a data frame?

Expand to see solution

```r
# Assign VARNAME to be the rownames of dd
rownames(dd) <- dd$VARNAME
# Rearrange by row names:
dd2 <- dd[colnames(ds), ]
# Check if this worked:
all.equal(dd2$VARNAME, colnames(ds))
```

```
[1] TRUE
```

We can use `match` also:

```r
# match returns a vector of the positions of (first) matches of its first
# argument in its second.
dd3 <- dd[match(colnames(ds), dd$VARNAME), ]
# Check if this worked:
all.equal(dd3$VARNAME, colnames(ds))
```

```
[1] TRUE
```

11.6 Question: use `arrange`?

Question: Is there a way to do this using `arrange`?

Expand to see the first attempt

This does not work, because `tidyverse` wants to work on columns of data within `dd`:

```
dd4 <- dd %>%
    arrange(colnames(ds))
# Check if this worked:
all.equal(dd4$VARNAME, colnames(ds))
```

```
[1] "27 string mismatches"
```

11.7 Question: use `arrange`?

Question: Is there a way to do this using `arrange`?

`arrange()` orders the rows of a data frame by the values of selected columns.

Expand to see solution

```
dd4 <- dd %>%
    mutate(neworder = match(.$VARNAME, colnames(ds))) %>%
    arrange(neworder) %>%
    select(-neworder)
```

```
mutate: new variable 'neworder' (integer) with 27 unique values and 0% NA

select: dropped one variable (neworder)
```

```
all.equal(dd4$VARNAME, colnames(ds))
```

```
[1] TRUE
```

11.8 Question: use `slice`

Question: Is there a way to do this using the `slice` command?

`slice()` lets you index rows by their (integer) locations.

Expand to see solution

```
dd6 <- dd %>%
    slice(match(colnames(ds), .$VARNAME))
```

```
slice: no rows removed
```

```
all.equal(dd6$VARNAME, colnames(ds))
```

```
[1] TRUE
```

11.9 Question: use `select`?

Question: Is there a way to do this by transposing and then using `select`?

Expand to see solution

```
# Transpose so rows become columns, and then we can use 'select' to rearrange
# those columns, and then transpose back, and rename columns as needed.
dd5 <- dd %>%
    t() %>%
    as_tibble(.name_repair = "unique") %>%
    select(colnames(ds)) %>%
    t() %>%
    as.data.frame() %>%
    rename(VARNAME = "V1", TYPE = "V2")
```

```
select: columns reordered (ID, b, z, a, p, …)
```

```
rename: renamed 2 variables (VARNAME, TYPE)
```

```
all.equal(dd5$VARNAME, colnames(ds))
```

```
[1] TRUE
```

11.10 Question: use row names

Question: What about using row names?

"While a tibble can have row names (e.g., when converting from a regular data frame), they are removed when subsetting with the [operator. A warning will be raised when attempting to assign non-NULL row names to a tibble. Generally, it is best to avoid row names, because they are basically a character column with different semantics than every other column."

From: https://tibble.tidyverse.org/reference/rownames.html

12 R Exploratory Data Analysis Exercise

12.1 Load Libraries

```
library(tidyverse)
library(tidylog)
library(DataExplorer)
library(GGally)
```

12.2 Explore Project 1 data

Let's explore the Project 1 data set:

```
load("data/project1.RData", verbose = TRUE)
```

```
Loading objects:
  ds
  dd
```

- ds = data set
- dd = data dictionary

12.3 Dimensions

- What are the dimensions of our data?

12.4 Dimensions

Task: Examine the dimensions of our data and data dictionary.

12.4.1 Data `ds`

```
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"
```

12.4.2 Data dictionay `dd`

```
dim(dd)
```

```
[1] 27  5
```

```
names(dd)
```

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

12.5 Arrangement

- How are the data arranged?
  - Is it in tidy format?
  - Is it one row per sample or per subject?
  - Were subjects sampled more than once?

12.5.1 Samples or subjects

Is it one row per sample or per subject?

Question: How would you figure out the answer to this question?

Expand to see solution

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

```
[1] 72
```

```
length(unique(ds$sample_id))
```

```
[1] 119
```

```
length(unique(ds$subject_id))
```

```
[1] 54
```

12.5.2 Unique values

Question: How can we figure out the number of unique values in each column of our `ds` data frame?

Expand to see solution

```
sapply(ds, function(x) {
    length(unique(x))
}) %>%
    kable()
```

	x
sample_id	119
Sample_trimester	4
Gestationalage_sample	189
subject_id	54
strata	21

race	3
maternal_age_delivery	54
case_control_status	2
prepregnancy_weight	51
height	42
prepregnancy_BMI	54
gravidity	5
parity	4
gestationalage_delivery	54
average_SBP_lt20weeks	19
average_DBP_lt20weeks	16
average_SBP_labor	23
average_DBP_labor	27
smoke_lifetime	2
baby_birthweight	30
baby_sex	2
baby_birthweight_centile	52
baby_SGA	1
placental_pathology	2

12.5.3 Subject-level data set

Task: Construct a subject-level data set

How would you construct a subject-level data set?

Expand to see solution

```
ds.subj <- ds %>%
    select(-sample_id, -Sample_trimester, -Gestationalage_sample) %>%
    distinct()
```

```
select: dropped 3 variables (sample_id, Sample_trimester, Gestationalage_sample)

distinct: removed 136 rows (71%), 55 rows remaining
```

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

```
[1] 1
```

```r
  ds.subj %>%
      group_by(subject_id) %>%
      filter(n() > 1)
```

```
group_by: one grouping variable (subject_id)

filter (grouped): removed 53 rows (96%), 2 rows remaining

# A tibble: 2 x 21
# Groups:   subject_id [1]
  subject_id strata race  matern~1 case_~2 prepr~3 height prepr~4 gravi~5 parity
  <chr>       <dbl> <chr>    <dbl>   <dbl>   <dbl>  <dbl>   <dbl>   <dbl>  <dbl>
1 SUBJ20         35 W         29.4       1    244.   63.1    44.0       1      0
2 SUBJ20         35 White     29.4       1    244.   63.1    44.0       1      0
# ... with 11 more variables: gestationalage_delivery <dbl>,
#   average_SBP_lt20weeks <dbl>, average_DBP_lt20weeks <dbl>,
#   average_SBP_labor <dbl>, average_DBP_labor <dbl>, smoke_lifetime <chr>,
#   baby_birthweight <dbl>, baby_sex <chr>, baby_birthweight_centile <dbl>,
#   baby_SGA <chr>, placental_pathology <chr>, and abbreviated variable names
#   1: maternal_age_delivery, 2: case_control_status, 3: prepregnancy_weight,
#   4: prepregnancy_BMI, 5: gravidity
```

```r
  ds.subj <- ds.subj %>%
      filter(race != "White")
```

```
filter: removed one row (2%), 54 rows remaining
```

```r
  sum(duplicated(ds.subj$subject_id))
```

```
[1] 0
```

12.6 Coding

- How are the data coded?
  - Are they coded correctly?
  - Which are categorical and which are continuous?
  - Are they coded consistently with the data dictionary?
  - Is there a data dictionary?

- Do we need to skip rows when reading the data in?

12.6.1 Recode for understandability

Let's recode `case_control_status` from `0` and `1` into a new `PE_status` variable coded as `control` and `case`.

```
dd %>%
    filter(R21.Variable.Name == "case_control_status") %>%
    pull(Variable.Coding)
```

```
filter: removed 26 rows (96%), one row remaining
```

```
[1] "0: normotensive control; 1: preeclampsia case"
```

Task: recode `case_control_status` from `0` and `1` into a new `PE_status` variable coded as `control` and `case`.

Expand to see solution

```
ds.subj$PE_status <- factor(ds.subj$case_control_status)
levels(ds.subj$PE_status)
```

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

```
levels(ds.subj$PE_status) <- c("control", "case")
xtabs(~case_control_status + PE_status, data = ds.subj)
```

```
                  PE_status
case_control_status control case
                  0      26    0
                  1       0   28
```

12.7 Missing data

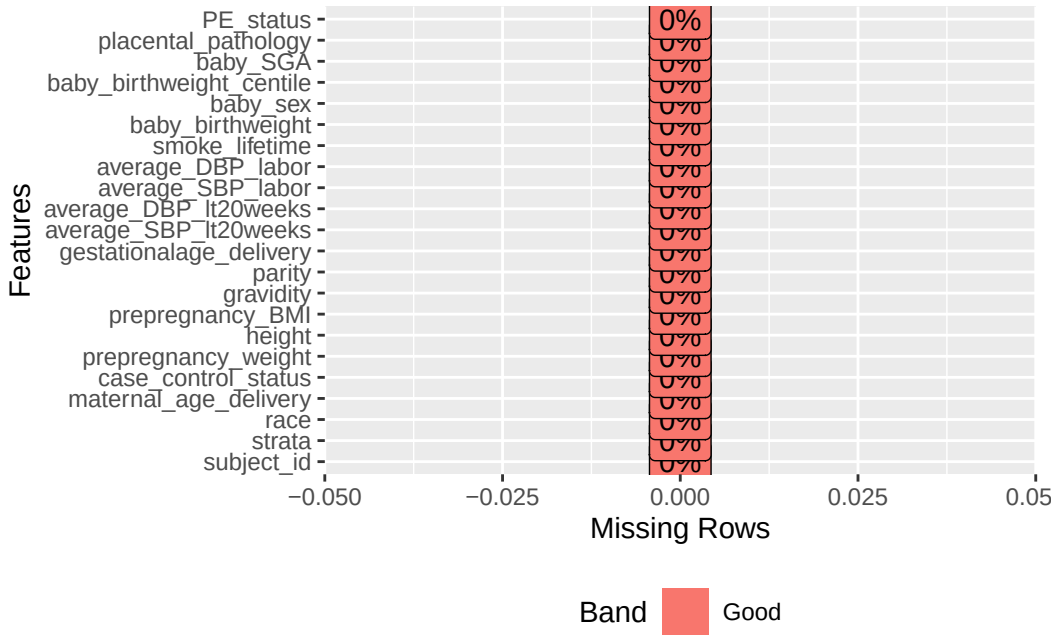
- What is the pattern of missing data?
  - How are missing data coded?

- Is there a single missing data code?

Here we use some plotting commands from the `DataExplorer` R package.

https://boxuancui.github.io/DataExplorer/index.html

```
plot_missing(ds.subj)
```

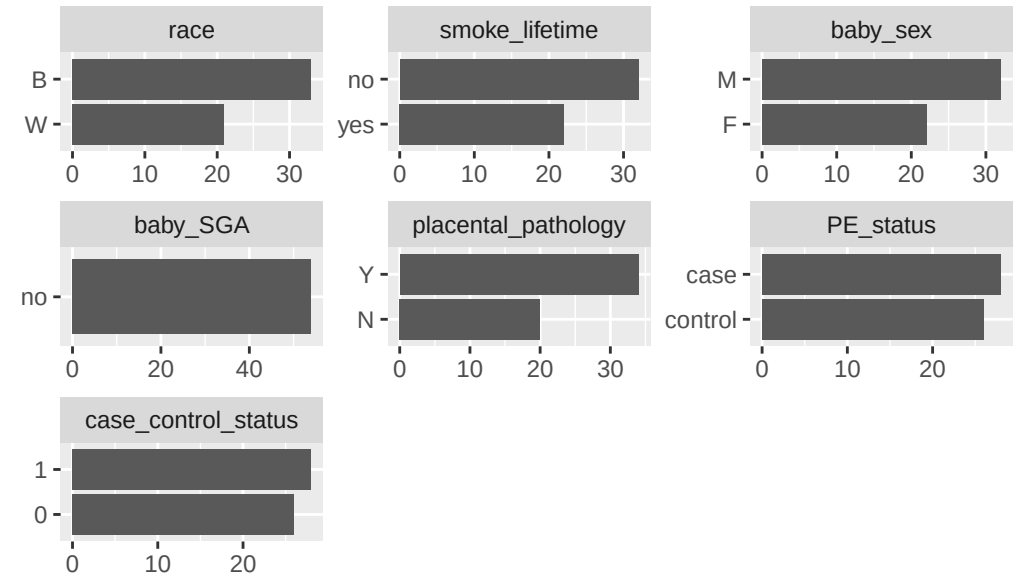


12.8 Distribution

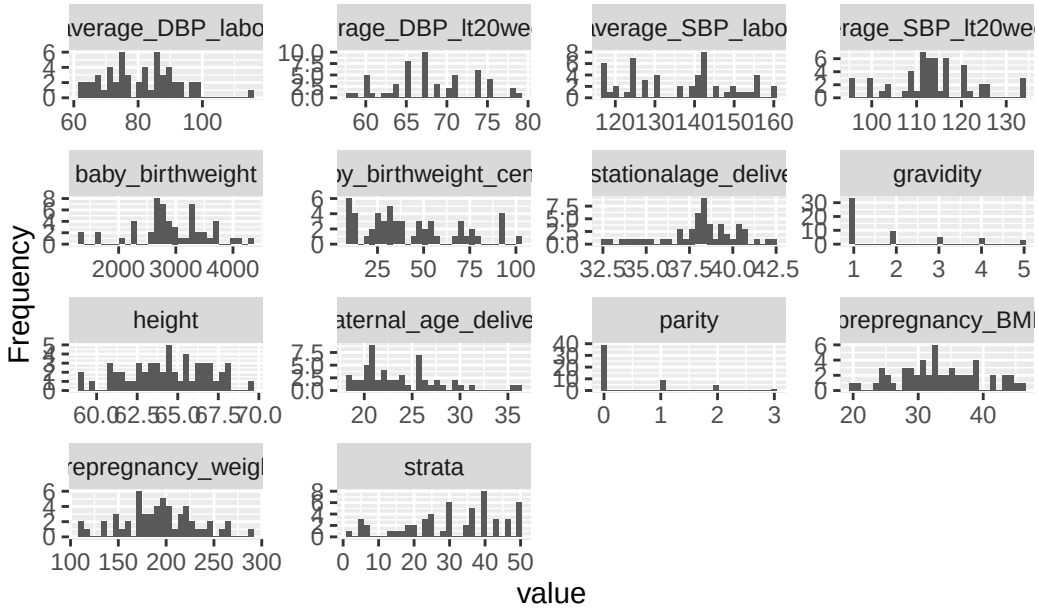
- What is the distribution of each of our phenotypes?
  - Are data skewed?
  - What is the range of values?
  - Is the range of values realistic?

```
plot_bar(ds.subj)
```

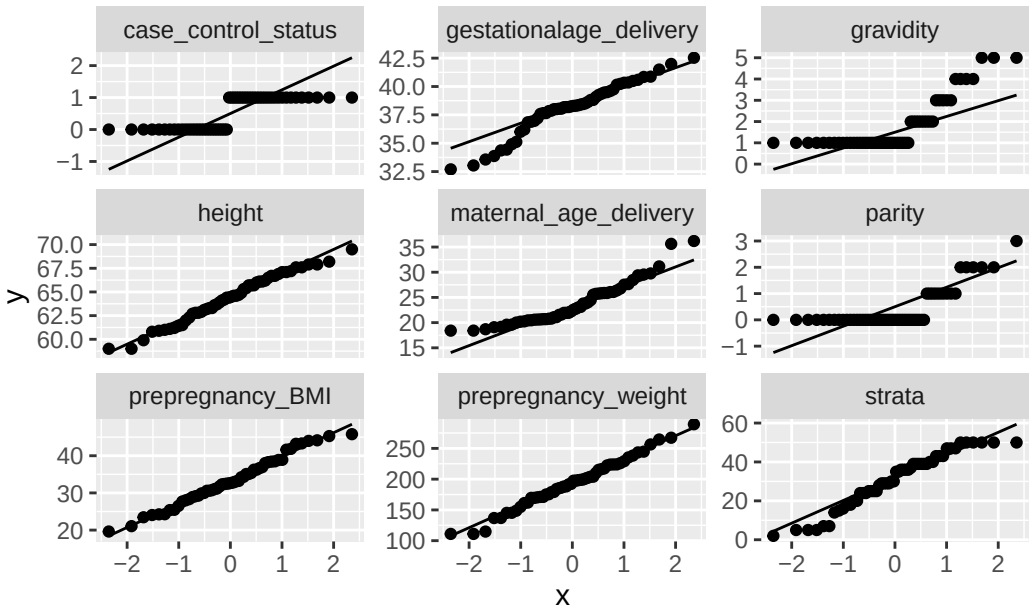
```
1 columns ignored with more than 50 categories.
subject_id: 54 categories
```



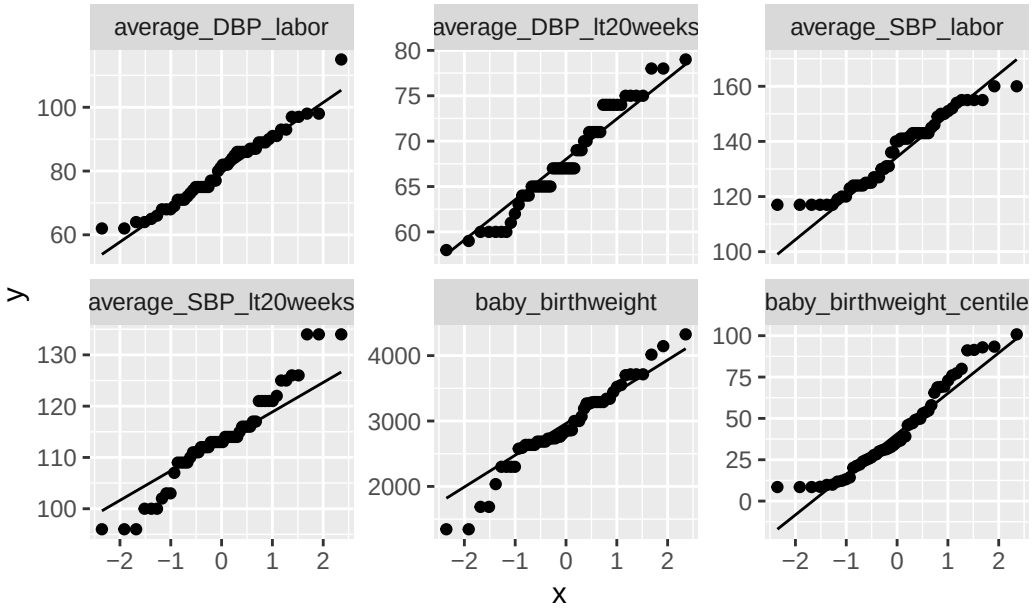

```
plot_histogram(ds.subj)
```




```
plot_qq(ds.subj)
```



Page 1



Page 2

12.9 Variation

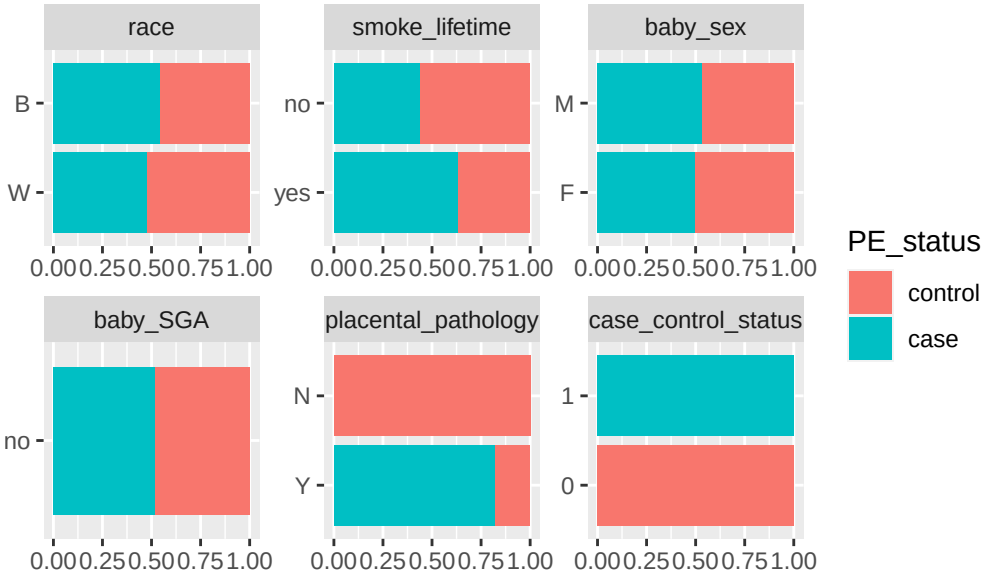
- How do our data vary and co-vary?

- Do multiple measures agree with each other?
- Are there sex-specific or age-specific differences?

12.9.1 Bar plots

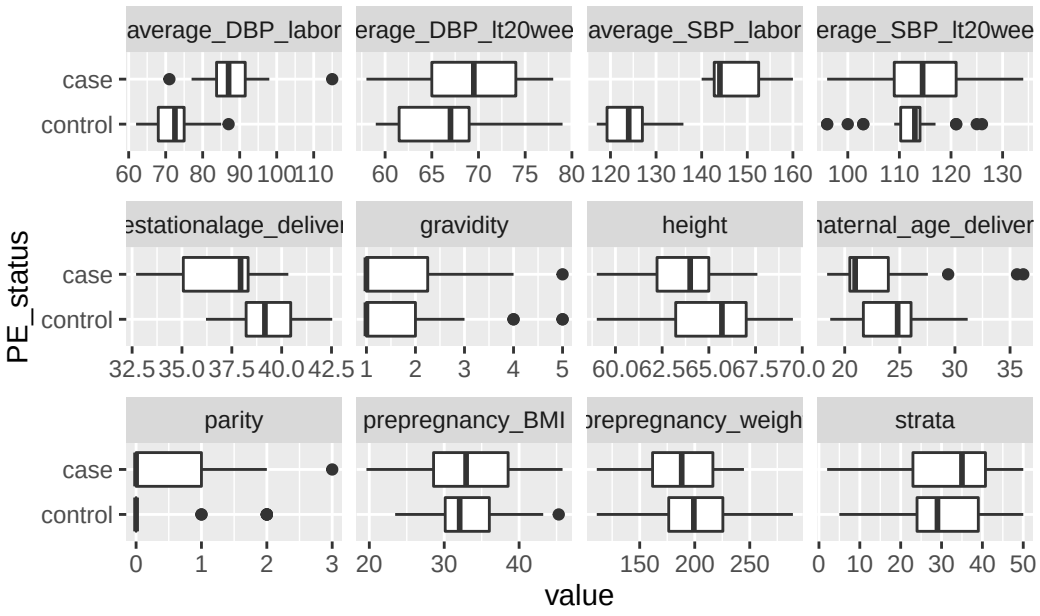
```
plot_bar(ds.subj, by = "PE_status")
```

```
1 columns ignored with more than 50 categories.
subject_id: 54 categories
```

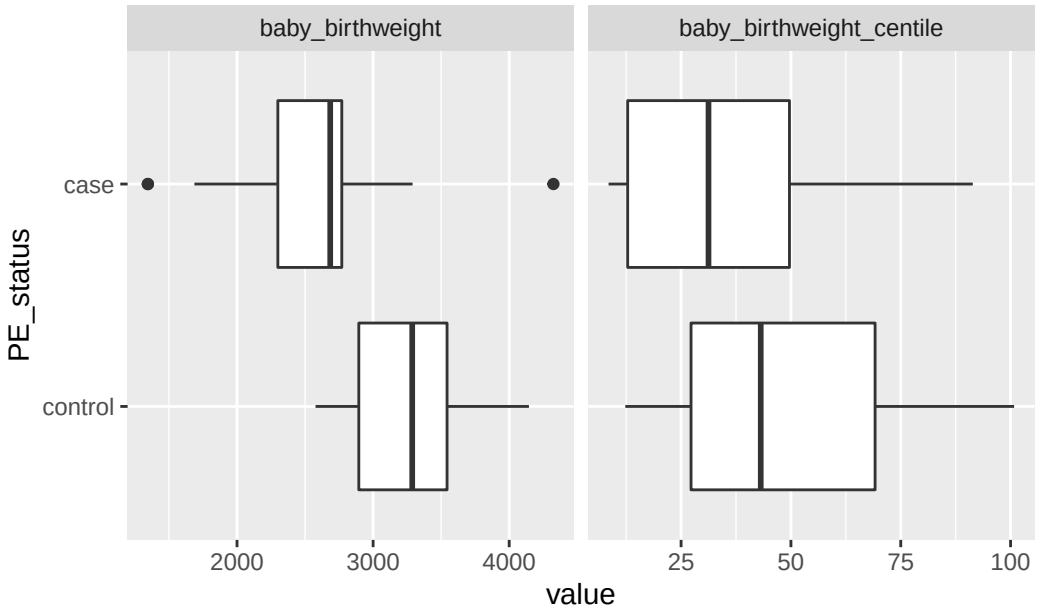


12.9.2 Box plots

```
plot_boxplot(ds.subj, by = "PE_status")
```


average_DBP_labor
erage_DBP_lt20wee
average_SBP_labor
erage_SBP_lt20wee
estationalage_deliver
gravidity
height
aternal_age_deliver
parity
prepregnancy_BMI
prepregnancy_weigh
strata
case
control
PE_status
value

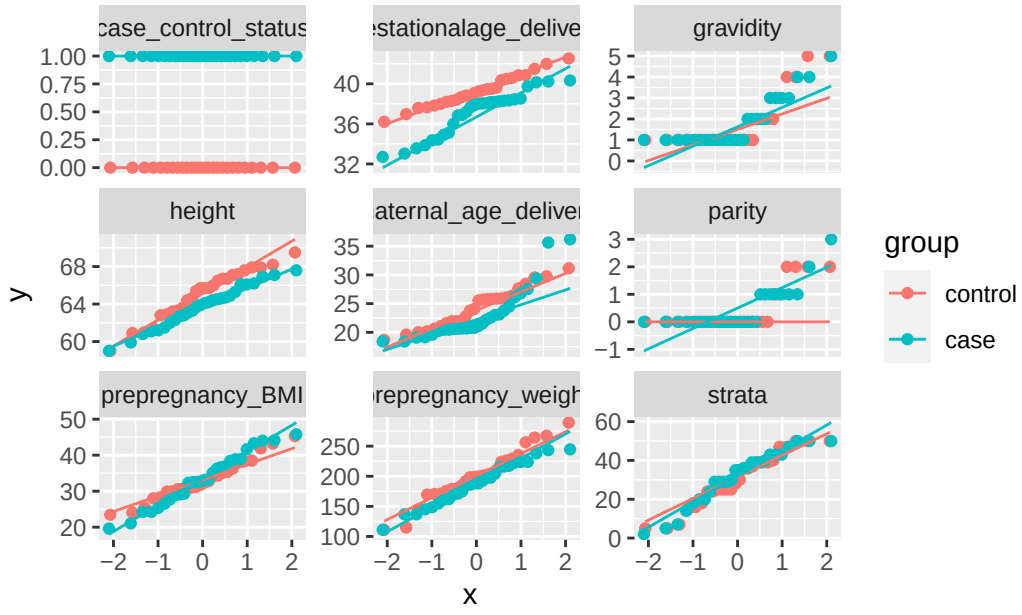
Page 1


baby_birthweight
baby_birthweight_centile
case
control
PE_status
value

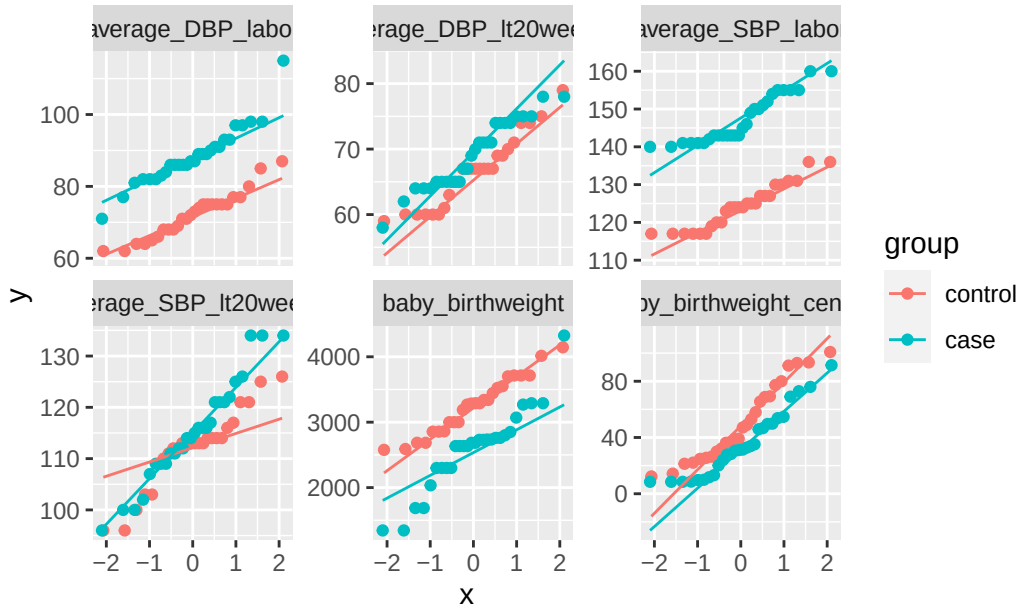
Page 2

12.9.3 QQ plots

```
plot_qq(ds.subj, by = "PE_status")
```



Page 1



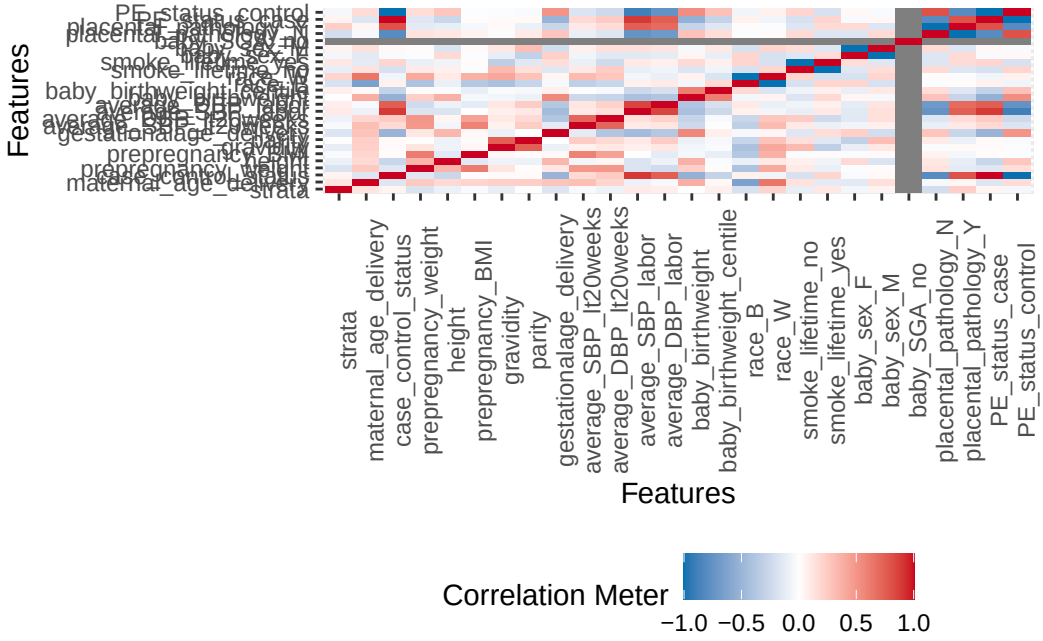
Page 2

12.9.4 Correlation

```
plot_correlation(ds.subj)
```

```
1 features with more than 20 categories ignored!
subject_id: 54 categories
```

```
Warning in cor(x = structure(list(strata = c(29, 39, 29, 36, 40, 39, 40, : the
standard deviation is zero
```

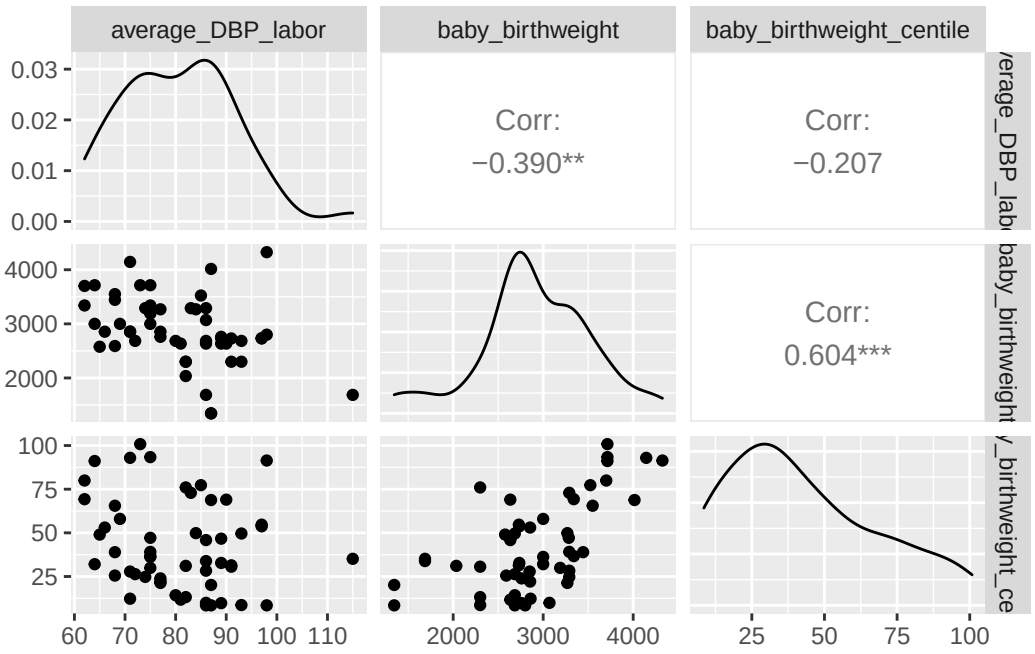


12.9.5 `ggpairs` from the `GGally` R package.

Use `ggpairs` from the `GGally` R package.

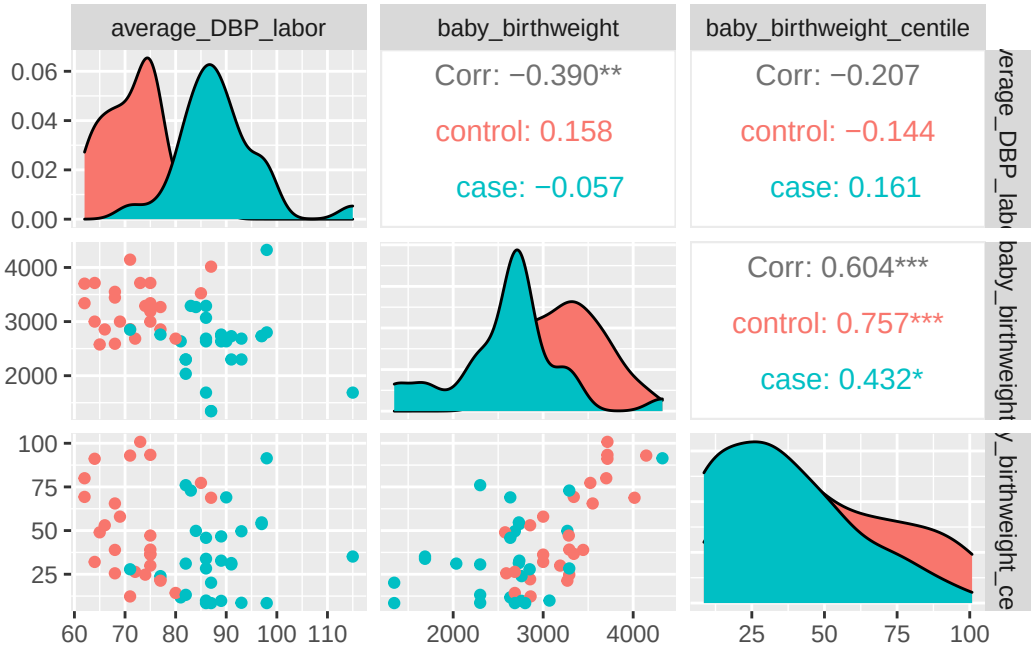
```
# Pull out numeric columns
ds1 <- ds.subj[, sapply(ds.subj, is.numeric)]
```

```
ggpairs(ds1[, c(13:15)])
```

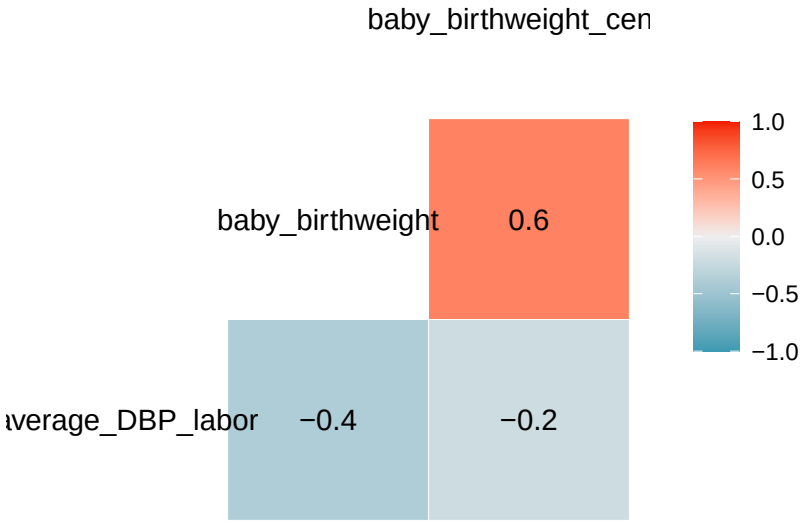


ggpairs - color by ggplot2 aes

```
ggpairs(ds.subj, columns = c(15, 17, 19), ggplot2::aes(color = PE_status))
```



```
ggcorr(ds1[, c(13:15)], label = TRUE)
```



12.10 DataExplorer

We can quickly create a report using the `create_report` function from the `DataExplorer` R package

```
create_report(ds.subj)
```

13 Basic Shell Commands

13.1 Licence and Acknowledgment

This chapter is a derivative of the Basic Shell Commands cheat sheet from the

`DEPRECATED-boot-camps/shell/shell_cheatsheet.md`

section of the Software Carpentry's GitHub repository

https://github.com/swcarpentry/DEPRECATED-boot-camps.

It is used under the Creative Commons - Attribution license

Attribution 3.0 Unported (CC BY 3.0)

https://creativecommons.org/licenses/by/3.0/

Minor section numbering and formatting changes were made here.

This chapter is licensed under the CC BY 3.0 license by Daniel E. Weeks.

13.2 Shell Basics:

Command	Definition
`.`	a single period refers to the current directory
`..`	a double period refers to the directory immediately above the current directory
`~`	refers to your home directory. *Note:* this command does NOT work on Windows machines (Mac and Linux are okay)
`cd ./dirname`	changes the current directory to the directory `dirname`
`ls -F`	tells you what files and directories are in the current directory
`pwd`	tells you what directory you are in (`pwd` stands for *p*rint *w*orking *d*irectory)

Command	Definition
`history`	lists previous commands you have entered. `history | less` lets you page through the list.
`man` *cmd*	displays the *man*ual page for a command.

13.3 Creating Things:

13.3.1 How to create new files and directories..

Command	Definition
`mkdir ./dirname`	makes a new directory called dirname below the current directory. *Note:* Windows users will need to use `\` instead of `/` for the path separator
`nano filename`	if `filename` does not exist, `nano` creates it and opens the `nano` text editor. If the file exists, `nano` opens it. *Note: (i)* You can use a different text editor if you like. In gnome Linux, `gedit` works really well too. *(ii)* `nano` (or `gedit`) create text files. It doesn't matter what the file extension is (or if there is one)

13.3.2 How to delete files and directories...

13.3.2.1 *Remember that deleting is forever. There is NO going back*

Command	Definition
`rm ./filename`	deletes a file called `filename` from the current directory
`rmdir ./dirname`	deletes the directory `dirname` from the current directory. *Note:* `dirname` must be empty for `rmdir` to run.

13.3.3 How to copy and rename files and directories...

Command	Definition
`mv tmp/filename .`	moves the file `filename` from the directory `tmp` to the current directory. *Note: (i)* the original `filename` in `tmp` is deleted. *(ii)* `mv` can also be used to rename files (e.g., `mv filename newname`
`cp tmp/filename .`	copies the file `filename` from the directory `tmp` to the current directory. *Note: (i)* the original file is still there

13.4 Pipes and Filters

13.4.1 How to use wildcards to match filenames...

Wildcards are a shell feature that makes the command line much more powerful than any GUI file managers. Wildcards are particularly useful when you are looking for directories, files, or file content that can vary along a given dimension. These wildcards can be used with any command that accepts file names or text strings as arguments.

13.4.1.1 Table of commonly used wildcards

Wildcard	Matches
`*`	zero or more characters
`?`	exactly one character
`[abcde]`	exactly one of the characters listed
`[a-e]`	exactly one character in the given range
`[!abcde]`	any character not listed
`[!a-e]`	any character that is not in the given range
`{software,carpentry}`	exactly one entire word from the options given

See the cheatsheet on regular expressions on the second page of this PDF cheatsheet for more "wildcard" shortcuts.

13.4.2 How to redirect to a file and get input from a file ...

Redirection operators can be used to redirect the output from a program from the display screen to a file where it is saved (or many other places too, like your printer or to another program where it can be used as input).

Command	Description
`>`	write `stdout` to a new file; overwrites any file with that name (e.g., `ls *.md > mardkownfiles.txt`)
`>>`	append `stdout` to a previously existing file; if the file does not exist, it is created (e.g., `ls *.md >> markdownfiles.txt`)
`<`	assigns the information in a file to a variable, loop, etc (e.g., `n < markdownfiles.md`)

13.4.2.1 How to use the output of one command as the input to another with a pipe...

A special kind of redirection is called a pipe and is denoted by `|`.

Command	Description
|	Output from one command line program can be used as input to another one (e.g. ls *.md | head gives you the first 5 `*.md` files in your directory)

13.4.2.1.1 Example:

```
ls *.md | head | sed -i `s/markdown/software/g`
```

changes all the instances of the word `markdown` to `software` in the first 5 `*.md` files in your current directory.

13.5 How to repeat operations using a loop...

Loops assign a value in a list or counter to a variable that takes on a different value each time through the loop. There are 2 primary kinds of loops: `for` loops and `while` loops.

13.5.1 For loop

For loops loop through variables in a list

```
for varname in list
do
    command1 $varname
    command2 $varname
done
```

where,

- `for`, `in`, `do`, and `done` are keywords
- `list` contains a list of values separated by spaces. e.g. `list` can be replaced by `1 2 3 4 5 6` or by `Bob Mary Sue Greg`. `list` can also be a variable:
- `varname` is assigned a value without using a `$` and the value is retrieved using `$varname`

```
list[0]=Sam
list[1]=Lynne
list[2]=Dhavide
list[3]=Trevor
.
.
.
list[n]=Mark
```

which is referenced in the loop by:

```
for varname in ${list[@]}
do
    command1 $varname
    command2 $varname
done
```

Note: Bash is zero indexed, so counting always starts at `0`, not `1`.

13.5.2 While Loop

While loops loop through the commands until a condition is met. For example

```
COUNTER=0
while [ ${COUNTER} -lt 10 ]; do
    command 1
    command 2
    COUNTER=`expr ${COUNTER} + 1`
done
```

continues the loop as long as the value in the variable COUNTER is less than 10 (incremented by 1 on each iteration of the loop).

- `while`, `do`, and `done` are keywords

13.5.2.1 Commonly used conditional operators

Operator	Definition
`-eq`	is equal to
`-ne`	is not equal to
`-gt`	greater than
`-ge`	greater than or equal to
`-lt`	less than
`-le`	less than or equal to

Use `man bash` or `man test` to learn about other operators you can use.

13.6 Finding Things

13.6.1 How to select lines matching patterns in text files...

To find information within files, you use a command called `grep`.

Example command	Description
`grep [options] day haiku.txt`	finds every instance of the string `day` in the file haiku.txt and pipes it to standard output

13.6.1.1 Commonly used `grep` options

	`grep` options
`-E`	tells grep you will be using a regular expression. Enclose the regular expression in quotes. *Note:* the power of `grep` comes from using regular expressions. Please see the regular expressions sheet for examples
`-i`	makes matching case-insensitive
`-n`	limits the number of lines that match to the first n matches
`-v`	shows lines that do not match the pattern (inverts the match)
`-w`	outputs instances where the pattern is a whole word

13.6.2 How to find files with certain properties...

To find file and directory names, you use a command called `find`

Example command	Description
`find . -type d`	`find` recursively descends the directory tree for each path listed to match the expression given in the command line with file or directory names in the search path

13.6.2.1 Commonly used `find` options

	`find` options
`-type [df]`	`d` lists directories; `f` lists files
`-maxdepth n`	`find` automatically searches subdirectories. If you don't want that, specify the number of levels below the working directory you would like to search
`-mindepth n`	starts `find`'s search `n` levels below the working directory

14 Summary

In summary, this book is a work in progress.

References