# HuGen2071 book

Daniel E. Weeks

September 12, 2023

# 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.sph.pitt.edu/directory/daniel-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 much 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://docs.github.com/en/get-started/using-git/about-git

https://docs.github.com/en/get-started/quickstart/hello-world

## 1.5 R Markdown

To introduce yourself or refresh yourself on R Markdown:

https://rmarkdown.rstudio.com/

Scroll down and click on "Get Started", which will take you to Lesson 1:

https://rmarkdown.rstudio.com/lesson-1.html

## 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.sph.pitt.edu/directory/daniel-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 Active Learning and Readings

## 4.1 Introduction and Overview

### 4.1.1 Learning Objectives

- Review the syllabus
- Describe bioinformatics and genetic/genomic data
- Describe dbGaP, an important genomic data repository

### 4.1.2 Required Reading

Mailman MD, Feolo M, Jin Y, Kimura M, Tryka K, Bagoutdinov R, Hao L, Kiang A, Paschall J, Phan L, Popova N, Pretel S, Ziyabari L, Lee M, Shao Y, Wang ZY, Sirotkin K, Ward M, Kholodov M, Zbicz K, Beck J, Kimelman M, Shevelev S, Preuss D, Yaschenko E, Graeff A, Ostell J, Sherry ST. The NCBI dbGaP database of genotypes and phenotypes. Nat Genet. 2007 Oct;39(10):1181-6. doi: 10.1038/ng1007-1181. PMID: 17898773; PMCID: PMC2031016. https://pubmed.ncbi.nlm.nih.gov/17898773/

### 4.1.3 Suggested Readings

Barnes (2007) Chapter 1 Carey MA, Papin JA. Ten simple rules for biologists learning to program. PLoS Comput Biol. 2018;14(1):e1005871. https://doi.org/10.1371/journal.pcbi.1005871

Dudley JT, Butte AJ. A quick guide for developing effective bioinformatics programming skills. PLoS Comput Biol. 2009;5(12):e1000589. https://doi.org/10.1371/journal.pcbi.1000589

## 4.2 GitHub

### 4.2.1 Learning Objectives

- To learn how to use GitHub
- To learn how to use GitHub Classroom

- To learn how to use GitHub within RStudio

### 4.2.2 Online Lecture

GitHub Introduction: https://danieleweeks.github.io/HuGen2071/gitIntro.html

### 4.2.3 Active Learning

Version Control with git and GitHub (Sections 4.1 - 4.4): https://learning.nceas.ucsb.edu/2020-11-RRCourse/session-4-version-control-with-git-and-github.html

### 4.2.4 Required Readings

#### GitHub Classroom Guide for Students

To set up GitHub Classroom, please follow the steps to set up RStudio, R, and git in this detailed guide: https://github.com/jfiksel/github-classroom-for-students

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

Be sure to generate an SSH key so you don't need to enter your password every time you interact with GitHub.

> ⚠ Warning
>
> Do not clone your repository onto a OneDrive or other cloud folder, as git does not work properly on cloud drives. Cloud drive systems typically maintain their own backup copies and this confuses git.

### 4.2.5 Suggested Readings

Happy Git and GitHub for the useR. https://happygitwithr.com/

Perez-Riverol Y, Gatto L, Wang R, et al. Ten Simple Rules for Taking Advantage of Git and GitHub. PLoS Comput Biol. 2016;12(7):e1004947. https://doi.org/10.1371/journal.pcbi.1004947

Version Control with Git: https://swcarpentry.github.io/git-novice/

Using Git from RStudio: https://ucsbcarpentry.github.io/2020-08-10-Summer-GitBash/24-supplemental-rstudio/index.html

## 4.3 R: Basics

### 4.3.1 Learning Objectives

- To become familiar with the R language and concepts
- To learn how to read and write data with R
- To learn control flow: choices and loops

### 4.3.2 Online Lectures

R Basics: https://danieleweeks.github.io/HuGen2071/RBasicsLecture.html

### 4.3.3 Active Learning:

https://datacarpentry.org/genomics-r-intro/01-r-basics.html

### 4.3.4 Suggested Readings

Buffalo (2015) Chapter 8 'R Language Basics' (Available online through PittCat+)

Read the first four sections, up to the end of 'Vectors, Vectorization, and Indexing'

https://pitt.primo.exlibrisgroup.com/permalink/01PITT_INST/i25aoe/cdi_askewsholts_vlebooks_9781449367510

https://datacarpentry.org/R-genomics/01-intro-to-R.html

Supplementary Reading: Spector (2008) Chapters 1 & 2 (Available online through PittCat+; link in syllabus)

## 4.4 R: Factors, Dates, Subscripting

### 4.4.1 Learning Objectives

- To learn how to subset data with R
- To learn how to handle factors and dates with R
- To learn how to manipulate characters with R

### 4.4.2 Online Lecture

R: factors, subscripting: https://danieleweeks.github.io/HuGen2071/RFactors.html

### 4.4.3 Active Learning:

Subsetting: https://swcarpentry.github.io/r-novice-gapminder/06-data-subsetting.html. This uses the gapminder data from here.

Factors: https://swcarpentry.github.io/r-novice-inflammation/12-supp-factors.html. This uses data from this Zip file.

### 4.4.4 Suggested Readings

Buffalo (2015) Chapter 8 'R Language Basics' (Available online through PittCat+)

Read the 'Factors and classes in R' subsection at the end of the 'Vectors, Vectorization, and Indexing' section.

Read the 'Exploring Data Through Slicing and Dicing: Subsetting Dataframes' section.

Read the 'Working with Strings' section.

https://pitt.primo.exlibrisgroup.com/permalink/01PITT_INST/i25aoe/cdi_askewsholts_vlebooks_9781449367510

https://datacarpentry.org/R-ecology-lesson/02-starting-with-data.html

Supplementary Readings: Spector (2008) Chapters 4, 5, 6

## 4.5 R: Character Manipulation

### 4.5.1 Learning Objectives

- To learn how to handle character data in R
- To learn how to use regular expressions in R

### 4.5.2 Active Learning

Regular expressions: https://csiro-data-school.github.io/regex/08-r-regexs/index.html

### 4.5.3 Required Readings

Read the chapter on "Strings" in "R for Data Science": https://r4ds.hadley.nz/strings

### 4.5.4 Suggested Readings

See the "String manipulation with stringr cheatsheet" at https://rstudio.github.io/cheatsheets/html/strings.html

Buffalo (2015) Chapter 8 'R Language Basics' (Available online through PittCat+)

Read the 'Working with Strings' section at the end of the "Working with and Visualizing Data in R" section.

https://pitt.primo.exlibrisgroup.com/permalink/01PITT_INST/i25aoe/cdi_askewsholts_vlebooks_9781449367510

Read the chapter on "Strings" in "R for Data Science": https://r4ds.hadley.nz/strings

Read the chapter on "Regular expressions" in "R for Data Science": https://r4ds.hadley.nz/regexps

Supplementary Reading: Spector (2008) Chapter 7

## 4.6 R: Loops and Flow Control

### 4.6.1 Learning Objectives

- To learn how to implement loops in R
- To learn how to control flow in R
- To learn how to vectorize operations

### 4.6.2 Online Lectures

Loops in R: https://danieleweeks.github.io/HuGen2071/RLoops.html

### 4.6.3 Active Learning:

Flow control and loops: https://swcarpentry.github.io/r-novice-gapminder/07-control-flow.html

Loops in R, Part I: https://danieleweeks.github.io/HuGen2071/loops.html

Vectorization: https://swcarpentry.github.io/r-novice-gapminder/09-vectorization.html

## 4.7 R: Functions and Packages, Debugging R

### 4.7.1 Learning Objectives

- To learn how to write R functions and packages
- To learn how to debug R code

### 4.7.2 Active Learning:

https://swcarpentry.github.io/r-novice-gapminder/10-functions.html

### 4.7.3 Suggested Readings

Functions Explained: https://swcarpentry.github.io/r-novice-gapminder/10-functions.html

Buffalo (2015) Chapter 8: Read the section 'Digression: Debugging R Code'

## 4.8 R: Tidyverse

### 4.8.1 Learning Objectives

- To learn how to use the pipe operator
- To learn how to use Tidyverse functions

### 4.8.2 Active Learning:

https://uomresearchit.github.io/r-day-workshop/04-dplyr/

### 4.8.3 Suggested Readings

Introduction to the Tidyverse: Manipulating tibbles with dplyr https://uomresearchit.github.io/r-day-workshop/04-dplyr/

Supplementary Reading: Buffalo (2015) Chapter 8: section 'Exploring Dataframes with dplyr'

## 4.9 R: Recoding and Reshaping Data

### 4.9.1 Learning Objectives

- To learn how to reformat and reshape data in R

### 4.9.2 Active Learning:

Recoding data: Pay particular attention to the `Recoding values` and `Creating new variables` sections

https://librarycarpentry.org/lc-r/03-data-cleaning-and-transformation.html

### 4.9.3 Suggested Readings

Reshaping data https://sscc.wisc.edu/sscc/pubs/dwr/reshape-tidy.html

Supplementary Reading: Spector (2008) Chapters 8 & 9

## 4.10 R: Merging Data

### 4.10.1 Learning Objectives

- To learn how to use the R 'merge' command
- To learn how to use the R Tidyverse join commands

### 4.10.2 Active Learning:

https://mikoontz.github.io/data-carpentry-week/lesson_joins.html

### 4.10.3 Required Reading

Tidy Animated Verbs https://www.garrickadenbuie.com/project/tidyexplain/

### 4.10.4 Suggested Readings

https://mikoontz.github.io/data-carpentry-week/lesson_joins.html#practice_with_joins_using_gapminder

Supplementary Reading: Buffalo (2015) Chapter 8 'Merging and Combining Data'. Spector (2008) Chapter 9.

## 4.11 R: Traditional Graphics & Advanced Graphics

### 4.11.1 Learning Objectives

- To learn the basic graphics commands of R
- To learn the R graphing package ggplot2

### 4.11.2 Active Learning:

https://datacarpentry.org/R-ecology-lesson/04-visualization-ggplot2.html

### 4.11.3 Suggested Readings

Plotting with ggplot2 https://datacarpentry.org/R-ecology-lesson/04-visualization-ggplot2.html

Supplementary Reading: Wickham (2009) Chapters 2 & 3

## 4.12 R: Exploratory Data Analysis

### 4.12.1 Learning Objectives

- To learn how to summarize data frames
- To learn how to visualize missing data patterns
- To learn how to visualize covariation

### 4.12.2 Readings

Missing value visualization with tidyverse in R https://towardsdatascience.com/missing-value-visualization-with-tidyverse-in-r-a9b0fefd2246

Suggested Reading: Buffalo (2015) Chapter 8 Sections: Exploring Data Visually with ggplot2 I: Scatterplots and Densities Exploring Data Visually with ggplot2 II: Smoothing Binning Data with cut() and Bar Plots with ggplot2 Using ggplot2 Facets.

## 4.13 R: Interactive and Dynamic Graphics

### 4.13.1 Learning Objectives

- To learn how to use interactive and dynamic graphics to explore your data more thoroughly
- To learn to use iPlots and Ggobi
- To learn to use plotly

### 4.13.2 Readings

Create interactive ggplot2 graphs with plotly https://www.littlemissdata.com/blog/interactiveplots

Suggested Reading: Wickham (2009) Chapters 2 & 3

## 4.14 Data Quality Checking and Filters

### 4.14.1 Learning Objectives

- To learn how to check genotype data for quality

### 4.14.2 Readings

Anderson CA, Pettersson FH, Clarke GM, Cardon LR, Morris AP, Zondervan KT. Data quality control in genetic case-control association studies. Nat Protoc. 2010 Sep;5(9):1564–1573. DOI: https://doi.org/10.1038/nprot.2010.116

Suggested Reading: Laurie CC, Doheny KF, Mirel DB, Pugh EW, Bierut LJ, Bhangale T, Boehm F, Caporaso NE, Cornelis MC, Edenberg HJ, Gabriel SB, Harris EL, Hu FB, Jacobs KB, Kraft P, Landi MT, Lumley T, Manolio TA, McHugh C, Painter I, Paschall J, Rice

JP, Rice KM, Zheng X, Weir BS, GENEVA Investigators. Quality control and quality assurance in genotypic data for genome-wide association studies. Genetic epidemiology. 2010 Sep;34(6):591–602. PMID: 20718045 DOI: https://doi.org/10.1002/gepi.20516

## 4.15 Unix: Basics, Streams, Redirection, & Pipe

### 4.15.1 Learning Objectives

- To learn basic Unix commands
- To learn how streams operate in Unix
- To learn out to pass streamed data from program to program in Unix

### 4.15.2 Readings

Buffalo (2015) Chapter 3

"Chapter 43: Redirecting Input and Output" in Unix Power Tools, 3rd Edition by Jerry Peek, Shelley Powers, Tim O'Reilly, Mike Loukides. Published by O'Reilly Media, Inc. https://pitt.primo.exlibrisgroup.com/permalink/01PITT_INST/e8h8hp/alma9998520758606236

Terminus, a web-based game for learning and practicing basic UNIX commands https://web.mit.edu/mprat/Public/web/Terminus/Web/main.html

## 4.16 Unix: Interacting with Processes, Cluster Jobs, Shell Scripting

### 4.16.1 Learning Objectives

- To learn how to interact with running processes
- To learn about the cluster and how to submit jobs there
- To learn how to write a script that can run in Unix

### 4.16.2 Active Learning:

Software Carpentry Unix Shell intro parts 1-3 https://swcarpentry.github.io/shell-novice/

### 4.16.3 Readings

Buffalo (2015) Chapter 7 up to the start of "Sorting Plain-Text Data with Sort" section.

Suggested Reading: Software Carpentry Unix Shell intro parts 1-3 (https://swcarpentry.github.io/shell-novice/)

## 4.17 Genetic Data Structures

### 4.17.1 Learning Objectives

- To learn about what genetic data is stored and principles for storing it

### 4.17.2 Readings

Introduction to PLINK (22n14-rlm-Introduction_to_PLINK.pdf, included in this lecture's folder)

Bennett RL, Steinhaus KA, Uhrich SB, O'Sullivan CK, Resta RG, Lochner-Doyle D, Markel DS, Vincent V, Hamanishi J. Recommendations for standardized human pedigree nomenclature. J Genet Couns. 1995 Dec;4(4):267-79. https://doi.org/10.1007/BF01408073. PMID: 24234481.

Bennett RL, French KS, Resta RG, Doyle DL. Standardized human pedigree nomenclature: update and assessment of the recommendations of the National Society of Genetic Counselors. J Genet Couns. 2008 Oct;17(5):424-33. https://doi.org/10.1007/s10897-008-9169-9. Epub 2008 Sep 16. PMID: 18792771.

Bennett RL, French KS, Resta RG, Austin J. Practice resource-focused revision: Standardized pedigree nomenclature update centered on sex and gender inclusivity: A practice resource of the National Society of Genetic Counselors. J Genet Couns. 2022 Sep 15. https://doi.org/10.1002/jgc4.1621. Epub ahead of print. PMID: 36106433.

## 4.18 PLINK I

### 4.18.1 Learning Objectives

- Describe PLINK formats
- Create PLINK datafiles
- Use PLINK to perform genetic association testing

### 4.18.2 Readings

Marees AT, de Kluiver H, Stringer S, Vorspan F, Curis E, Marie-Claire C, Derks EM. A tutorial on conducting genome-wide association studies: Quality control and statistical analysis. Int J Methods Psychiatr Res. 2018 Jun;27(2):e1608. PMID: 29484742 PMCID: PMC6001694 DOI: https://doi.org/10.1002/mpr.1608

https://github.com/MareesAT/GWA_tutorial/

## 4.19 PLINK II

### 4.19.1 Learning Objectives

- To learn how to use PLINK to manipulate data files

## 4.20 PLINK Computer Lab

### 4.20.1 Learning Objectives

- To practice using PLINK to manipulate data files

## 4.21 Unix: Data Manipulation

### 4.21.1 Learning Objectives

- To learn Unix tools like sed and awk that can be used to manipulate data

### 4.21.2 Readings

Buffalo (2015) Chapter 7 from the "Sorting Plain-Text Data with Sort" section on.

## 4.22 Unix: Pipes & Parallelization

### 4.22.1 Learning Objectives

- To learn to string programs together to process data
- To learn how to parallelize functions in Unix

### 4.22.2 Active Learning:

Software Carpentry Unix Shell intro part 4 https://swcarpentry.github.io/shell-novice/04-pipefilter.html

### 4.22.3 Readings

Buffalo (2015) Chapter 12: "Bioinformatics Shell Scripting, Writing Pipelines, and Parallelizing Tasks"

## 4.23 Unix: Scripting, Control Structures and Variables

### 4.23.1 Learning Objectives

- To learn how to use control structures in Unix scripting
- To learning how to use variables in Unix

### 4.23.2 Active Learning:

Software Carpentry Unix Shell intro parts 5-7 https://swcarpentry.github.io/shell-novice/

### 4.23.3 Readings

Software Carpentry Unix Shell intro parts 5-7 (https://swcarpentry.github.io/shell-novice/)

## 4.24 VCF, bcftools, vcftools

### 4.24.1 Learning Objectives

- To learn about VCF data format
- To learn about bcftools and vcftools for manipulating VCF files

## 4.25 SAM & samtools

### 4.25.1 Learning Objectives

- To learn about SAM data format for sequence data
- To learn about samtools to manipulate SAM data files

### 4.25.2 Readings

Buffalo Chapter 11 "Working with Alignment Data"

Data Wrangling and Processing for Genomics https://data-lessons.github.io/wrangling-genomics/

Relevant links: The Sequence Alignment/Map Format Specification http://samtools.github.io/hts-specs/

## 4.26 Genetic Data in R, GDS

### 4.26.1 Learning Objectives

- To learn about data structures in R for storing genetic data
- To learn about the GDS format

### 4.26.2 Active Learning:

https://uw-gac.github.io/topmed_workshop_2017/gds-format.html (Only 2.1 - Exploring a GDS file)

### 4.26.3 Readings

Zheng X, Gogarten SM, Lawrence M, Stilp A, Conomos MP, Weir BS, Laurie C, Levine D. SeqArray-a storage-efficient high-performance data format for WGS variant calls. Bioinformatics. 2017 Aug 1;33(15):2251-2257. doi: 10.1093/bioinformatics/btx145. PMID: 28334390; PMCID: PMC5860110. https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5860110/

# 5 GitHub

## 5.1 GitHub Introduction lecture

Here's a recording of this lecture (32 minutes 8 seconds):

Recording

## 5.2 GitHub Introduction slides

PDF slide set

# 6 Git Commands

Here's an outline of essential Git commands, initially created by ChatGPT:

## 6.1 Initialization and Configuration

- `git init`: Initializes a new Git repository in the current directory.
- `git config`: Configure Git settings.

## 6.2 Basic Workflow

- `git add`: Stage changes.
- `git commit -m "message"`: Commits staged changes with a descriptive message.

## 6.3 Remote Repositories

- `git clone`: Clones a remote repository to your local machine.
- `git push`: Send local changes to remote repository.
- `git pull`: Retrieve changes from remote.
- `git remote`: Manage remote repositories.

## 6.4 Status and Changes

- `git status`: Shows the current state of your working directory.
- `git diff`: Displays changes between working directory and the last commit.

## 6.5 History and Logs

- `git log`: View commit history.
- `git log --oneline`: Compact commit history.

## 6.6 Ignoring Files

- Create `.gitignore` file.

## 6.7 Branching

- `git branch`: List/create branches.
- `git checkout`: Switch branches.
- `git merge`: Merge branches.

## 6.8 Undoing Changes

- `git reset`: Unstage or reset changes.
- `git revert`: Create undoing commits.

## 6.9 Tagging

- `git tag`: Create and manage tags.

## 6.10 Stashing

- `git stash`: Temporarily store changes.

# 7 Lecture: R Basics

## 7.1 R Basics lecture

Here's a recording of this lecture (48 minutes 14 seconds):

Recording

## 7.2 R Basics slides

PDF slide set

# 8 R Basics Group Exercise

## 8.1 Question: Recycling in a dataframe

Suppose you have a dataframe `df` with three columns, `A`, `B`, and `C`, as follows:

```
df <- data.frame(
A = c(1, 2, 3, 4),
B = c(5, 6, 7, 8),
C = c(9, 10, 11, 12)
)
df
```

```
  A B  C
1 1 5  9
2 2 6 10
3 3 7 11
4 4 8 12
```

Now, you want to insert a shorter vector `D` into the `df` dataframe:

```
df$D <- c(13,14)
```

What will be the `D` column of `df` after the operation?

- (A) c(13, 14, NA, NA)
- (B) c(13, 14, 11, 12)
- (C) c(13, 14, 13, 14)
- (D) c(13, 14)

Please select the correct option.

## 8.2 Exercise 1: recycling

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

First, let's 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
```

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.

> **Warning**
>
> Use a Chrome or Firefox browser - WebR does not work with the Safari browser yet. If the following WebR chunk is working properly, you should see an editor window below the `Run code` tab displaying this line of R code: `(a <- data.frame(n = 1:4))`.

```
(a <- data.frame(n = 1:4))
# Edit/add R code here
```

> **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
```

## 8.3 Question: Vector addition and recycling

Suppose you have two vectors in R:

Vector A: c(1, 2, 3)
Vector B: c(4, 5)

If you perform the operation A + B, what will be the result of vector recycling?

- (A) c(5, 7, 3)
- (B) c(5, 7, 8)
- (C) c(5, 7, 7)
- (D) c(5, 5, 3)

Please select the correct option.

## 8.4 Exercise 2: vector addition

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

```
# Edit/add code here
```

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
```

## 8.5 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:

```r
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:

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

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

```r
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.

```r
# Edit/add code here
```

> **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

```
# 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
```

## 8.6 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.

```
# Edit/add code here
```

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
```

## 8.7 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.

```
# Edit/add code here
```

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
```

## 8.8 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.

```
# Edit/add code here
```

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
```

## 8.9 Exercise 7

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

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

```
# Edit/add code here
```

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
```

## 8.10 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.

```
dir.create("data")
infile <- "data/testdata.txt"
srcfile <- "https://raw.githubusercontent.com/DanielEWeeks/HuGen2071/main/data/testdata.tx
download.file(srcfile, infile)
# Adjust the read.table options to read the file correctly as desired.
b <- read.table(infile)
b
```

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
```

# 9 Lecture: R: factors, subscripting

## 9.1 R: factors, subscripting lecture

Here's a recording of this lecture (43 minutes 25 seconds):

Recording

## 9.2 R: factors, subscripting slides

PDF slide set

# 10 R Character Exercise

## 10.1 Load Libraries

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

## 10.2 Useful RStudio cheatsheet

See the "String manipulation with stringr cheatsheet" at

https://rstudio.github.io/cheatsheets/html/strings.html

## 10.3 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 |

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

| ID | SBP |
|---|---|
| SG0001 | 120 |
| SG0104 | 111 |
| SG2112 | 125 |
| SG2543 | 119 |

```
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 |

## 10.4 Discussion Questions

### 10.4.1 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'.

Table 10.4: The `clin` data frame

| ID | height |
|---|---|
| 1 | 152 |
| 104 | 172 |
| 2112 | 180 |
| 2543 | 163 |

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.

Hint: Use the `formatC` function.

#### 10.4.1.1 Interactive WebR chunk

You can interactively run R within this WebR chunk by clicking the `Run code` tab. Note that this is a limited version of R which runs within your web browser.

> **Warning**
>
> Use a Chrome or Firefox browser - WebR does not work with the Safari browser yet.

> **Note**
>
> This `Run code` WebR chunk needs to be run first, before the later ones, as it downloads and reads in the required data files. The WebR chunks should be run in order, as you encounter them, from beginning to end.

```
# Download files within the WebR environment
dir.create("data")
infiles <- c("data/clinical_data.txt","data/lab_data.txt","data/genotype_data.txt")
root_srcfile <- "https://raw.githubusercontent.com/DanielEWeeks/HuGen2071/main/"
for (i in 1:length(infiles)) {
   download.file(paste0(root_srcfile,infiles[i]), infiles[i])
}
# kable is not available in WebR
kable <- head
# Read the three files in:
clin <- read.table(file = "data/clinical_data.txt", header=TRUE)
kable(clin)
lab <- read.table(file = "data/lab_data.txt", header = TRUE)
kable(lab)
geno <- read.table(file = "data/genotype_data.txt", header = TRUE)
kable(geno)
# Edit/add R code here
```

### 10.4.2 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 |

### 10.4.3 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 10.6: 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 from the `stringr` package.

Warning

To read in and load the data within the WebR environment, be sure to run all of the WebR chunks in order. For example, to usefully run R code in this WebR chunk here, you first need to run the WebR chunk above in Question 1.

```
# str_replace_all is in the stringr R package
library(stringr)
# Edit/add code here
```

### 10.4.4 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 |

### 10.4.5 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 10.9: 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 from the `stringr` package or the `separate` function from the `tidyr` package.

```
# separate is in the tidyr R package
library(tidyr)
# Edit/add code here
```

### 10.4.6 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
```

## 10.5 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 |

### 10.5.1 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 10.12: The `joint` data frame

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

```
# Download files within the WebR environment
dir.create("data")
infiles <- c("data/joint_data.txt")
root_srcfile <- "https://raw.githubusercontent.com/DanielEWeeks/HuGen2071/main/"
for (i in 1:length(infiles)) {
   download.file(paste0(root_srcfile,infiles[i]), infiles[i])
}
joint <- read.table(file = "data/joint_data.txt", header = TRUE)
kable(joint)
# Edit/add code here
```

### 10.5.2 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 |

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

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

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

| ID | SBP |
|---|---|
| RP5002 | 121 |
| RP5012 | 118 |
| RP5113 | 112 |
| RP5213 | 142 |

# 11 Lecture: Loops in R

## 11.1 Loops in R lecture

Here's a recording of this lecture (8 minutes 11 seconds):

Recording

## 11.2 Loops in R slides

PDF slide set

# 12 Loops in R, Part I

## 12.1 Acknowledgment/License

The original source for this chapter was from the web site

https://datacarpentry.org/semester-biology/

which was built using this underlying code

https://github.com/datacarpentry/semester-biology

and is used under the

Attribution 4.0 International (CC BY 4.0)

license https://creativecommons.org/licenses/by/4.0/.

The material presented here has been modified from the original source.

Accordingly this chapter is made available under the same license terms.

## 12.2 Source code

If you'd like to work within R Studio using the source code of this chapter, you can obtain it from here.

## 12.3 Basic `for` loop

- Loops are the fundamental structure for repetition in programming
- `for` loops perform the same action for each item in a list of things

```
for (item in list_of_items) {
  do_something(item)
}
```

- To see an example of this let's calculate masses from volumes using a loop

- Need `print()` to display values inside a loop or function

```
volumes = c(1.6, 3, 8)
for (volume in volumes){
  mass <- 2.65 * volume ^ 0.9
  print(mass)
}
```

- Code in the loop will run once for each value in volumes
- Everything between the curly brackets is executed each time through the loop
- Code takes the first value from `volumes` and assigns it to `volume` and does the calculation and prints it
- Then it takes the second value from `volumes` and assigns it to `volume` and does the calculation and prints it
- And so on
- So, this loop does the same exact thing as

```
volume <- volumes[1]
mass <- 2.65 * volume ^ 0.9
print(mass)
volume <- volumes[2]
mass <- 2.65 * volume ^ 0.9
print(mass)
volume <- volumes[3]
mass <- 2.65 * volume ^ 0.9
print(mass)
```

**!** Do Tasks 1 & 2 in Basic For Loops

1. The code below prints the numbers 1 through 5 one line at a time. Modify it to print each of these numbers multiplied by 3.

```
numbers <- c(1, 2, 3, 4, 5)
for (number in numbers){
  print(number)
}
```

2. Write a for loop that loops over the following vector and prints out the mass in kilograms (`mass_kg = 2.2 * mass_lb`)

```
(mass_lbs <- c(2.2, 3.5, 9.6, 1.2))
# Edit/add/try out R code here
```

## 12.4 Looping with an index & storing results

- R loops iterate over a series of values in a vector or other list like object
- When we use that value directly this is called looping by value
- But there is another way to loop, which is called looping by index
- Looping by index loops over a list of integer index values, typically starting at 1
- These integers are then used to access values in one or more vectors at the position inicated by the index
- If we modified our previous loop to use an index it would look like this
- We often use `i` to stand for "index" as the variable we update with each step through the loop

```
volumes = c(1.6, 3, 8)
for (i ...)
```

- We then create a vector of position values starting at 1 (for the first value) and ending with the length of the object we are looping over

```
volumes = c(1.6, 3, 8)
for (i in 1:3)
```

- We don't want to have to know the length of the vector and it might change in the future, so we'll look it up using the `length()` function

```
volumes = c(1.6, 3, 8)
for (i in 1:length(volumes)){

}
```

- Then inside the loop instead of doing the calculation on the index (which is just a number between 1 and 3 in our case)
- We use square brackets and the index to get the appropriate value out of our vector

```
volumes = c(1.6, 3, 8)
for (i in 1:length(volumes)){
```

```
   mass <- 2.65 * volumes[i] ^ 0.9
   print(mass)
}
```

- This gives us the same result, but it's more complicated to understand
- So why would we loop by index?
- The advantage to looping by index is that it lets us do more complicated things
- One of the most common things we use this for are storing the results we calculated in the loop
- To do this we start by creating an empty object the same length as the results will be before the loop starts
- To store results in a vector we use the function `vector` to create an empty vector of the right length
- `mode` is the type of data we are going to store
- `length` is the length of the vector

```
masses <- vector(mode = "numeric", length = length(volumes))
masses
```

- Then add each result in the right position in this vector
- For each trip through the loop put the output into the empty vector at the `i`th position

```
for (i in 1:length(volumes)){
   mass <- 2.65 * volumes[i] ^ 0.9
   masses[i] <- mass
}
masses
```

❗ Do Tasks 3-4 in Basic For Loops.

3. Complete the code below so that it prints out the name of each bird one line at a time.

```
birds = c('robin', 'woodpecker', 'blue jay', 'sparrow')
for (i in 1:length(_________)){
  print(birds[__])
}
```

4. Complete the code below so that it stores one area for each radius.

```
radius <- c(1.3, 2.1, 3.5)
areas <- vector(_____ = "numeric", length = ______)
for (__ in 1:length(________)){
  areas[__] <- pi * radius[i] ^ 2
}
areas
```

## 12.5 Looping over multiple values

- Looping with an index also allows us to access values from multiple vectors

```
as <- c(2.65, 1.28, 3.29)
bs <- c(0.9, 1.1, 1.2)
volumes = c(1.6, 3, 8)
masses <- vector(mode="numeric", length=length(volumes))
for (i in 1:length(volumes)){
   mass <- as[i] * volumes[i] ^ bs[i]
   masses[i] <- mass
}
masses
```

❗ Do Task 5 in Basic For Loops.

5. Complete the code below to calculate an area for each pair of `lengths` and `widths`, store the areas in a vector, and after they are all calculated print them out:

```
lengths = c(1.1, 2.2, 1.6)
widths = c(3.5, 2.4, 2.8)
areas <- vector(length = __________)
for (i in _____) {
  areas[__] <- lengths[__] * widths[__]
}
areas
```

# 13 Loops in R, Part II

## 13.1 Acknowledgment/License

The original source for this chapter was from the web site

https://datacarpentry.org/semester-biology/

which was built using this underlying code

https://github.com/datacarpentry/semester-biology

and is used under the

Attribution 4.0 International (CC BY 4.0)

license https://creativecommons.org/licenses/by/4.0/.

The material presented here has been modified from the original source.

Accordingly this chapter is made available under the same license terms.

## 13.2 Source code

If you'd like to work within R Studio using the source code of this chapter, you can obtain it from here.

## 13.3 Looping with functions

- It is common to combine loops with with functions by calling one or more functions as a step in our loop
- For example, let's take the non-vectorized version of our `est_mass` function that returns an estimated mass if the `volume > 5` and `NA` if it's not.

```
est_mass <- function(volume, a, b){
  if (volume > 5) {
    mass <- a * volume ^ b
```

```
  } else {
    mass <- NA
  }
  return(mass)
}
class(est_mass)
```

- We can't pass the vector to the function and get back a vector of results because of the `if` statements
- So let's loop over the values
- First we'll create an empty vector to store the results
- And them loop by index, callling the function for each value of `volumes`

```
as <- c(2.65, 1.28, 3.29)
bs <- c(0.9, 1.1, 1.2)
volumes = c(1.6, 3, 8)
masses <- vector(mode="numeric", length=length(volumes))
for (i in 1:length(volumes)){
   mass <- est_mass(volumes[i], as[i], bs[i])
   masses[i] <- mass
}
masses
```

- This is the for loop equivalent of an `mapply` statement

```
(masses_apply <- mapply(est_mass, volumes, as, bs))
```

**!** Do Size Estimates By Name Loop.

If `dinosaur_lengths.csv` is not already in your working directory download a copy of the data on dinosaur lengths with species names. Load it into R.
Write a function `mass_from_length()` that uses the equation `mass <- a * length^b` to estimate the size of a dinosaur from its length. This function should take two arguments, `length` and `species`. For each of the following inputs for `species`, use the given values of `a` and `b` for the calculation:

- For `Stegosauria`: `a = 10.95` and `b = 2.64` (Seebacher 2001).
- For `Theropoda`: `a = 0.73` and `b = 3.63` (Seebacher 2001).
- For `Sauropoda`: `a` = `214.44` and `b = 1.46` (Seebacher 2001).
- For any other value of `species`: `a = 25.37` and `b = 2.49`.

1. Use this function and a for loop to calculate the estimated mass for each dinosaur,

store the masses in a vector, and after all of the calculations are complete show the first few items in the vector using `head()`.

```
root_srcfile <- "https://raw.githubusercontent.com/DanielEWeeks/HuGen2071/main/"
download.file(paste0(root_srcfile,"data/dinosaur_lengths.csv"),
              "dinosaur_lengths.csv")
list.files(pattern = "dinosaur")
# Edit/add/try out R code here
```

2. Add the results in the vector back to the original data frame. Show the first few rows of the data frame using `head()`.

```
# Edit/add/try out R code here
```

3. Calculate the mean mass for each `species` using `dplyr`.

```
# Edit/add/try out R code here
```

## 13.4 Looping over files

- Repeat same actions on many similar files
- Let's download some simulated satellite collar data

```
# Download files within the WebR environment
root_srcfile <- "https://raw.githubusercontent.com/DanielEWeeks/HuGen2071/main/"
download.file(paste0(root_srcfile,"data/locations.zip"),
              "locations.zip")
unzip("locations.zip")
```

- Now we need to get the names of each of the files we want to loop over
- We do this using `list.files()`
- If we run it without arguments it will give us the names of all files in the directory

```
list.files()
```

- But we just want the data files so we'll add the optional `pattern` argument to only get the files that start with `"locations-"`

```
(data_files = list.files(pattern = "locations-"))
```

- Once we have this list we can loop over it count the number of observations in each file
- First create an empty vector to store those counts

```
(n_files = length(data_files))
(results <- integer(n_files))
```

- Then write our loop

```
for (i in 1:n_files){
  filename <- data_files[i]
  data <- read.csv(filename)
  count <- nrow(data)
  results[i] <- count
}
results
```

❗ Do Task 1 of Multiple-file Analysis.

**Exercise uses different collar data**
You have a satellite collars on a number of different individuals and want to be able to quickly look at all of their recent movements at once. The data is posted daily to a zip file that contains one csv file for each individual: data/individual_collar_data.zip
Start your solution by:

- If `individual_collar_data.zip` is not already in your working directory download the zip file using `download.file()`
- Unzip it using `unzip()`
- Obtain a list of all of the files with file names matching the pattern `"collar-data-.*.txt"` (using `list.files()`)

1. Use a loop to load each of these files into R and make a line plot (using `geom_path()`) for each file with `long` on the `x` axis and `lat` on the `y` axis. Graphs, like other types of output, won't display inside a loop unless you explicitly display them, so you need put your `ggplot()` command inside a `print()` statement.

Include the name of the file in the graph as the graph title using `labs()`.

```
root_srcfile <- "https://raw.githubusercontent.com/DanielEWeeks/HuGen2071/main/"
download.file(paste0(root_srcfile,"data/individual_collar_data.zip"),
              "individual_collar_data.zip")
unzip("individual_collar_data.zip")
list.files()
# Edit/add/try out R code here
```

## 13.5 Storing loop results in a data frame

- We often want to calculate multiple pieces of information in a loop making it useful to store results in things other than vectors
- We can store them in a data frame instead by creating an empty data frame and storing the results in the `i`th row of the appropriate column
- Associate the file name with the count
- Also store the minimum latitude
- Start by creating an empty data frame
- Use the `data.frame` function
- Provide one argument for each column
- "Column Name" = "an empty vector of the correct type"

```
(results <- data.frame(file_name = character(n_files),
                       count = integer(n_files),
                       min_lat = numeric(n_files)))
```

- Now let's modify our loop from last time
- Instead of storing `count` in `results[i]` we need to first specify the `count` column using the `$`: `results$count[i]`
- We also want to store the filename, which is `data_files[i]`

```
for (i in 1:n_files){
  filename <- data_files[i]
  data <- read.csv(filename)
  count <- nrow(data)
  min_lat = min(data$lat)
  results$file_name[i] <- filename
  results$count[i] <- count
  results$min_lat[i] <- min_lat
}
```

```
results
```

❗ Do Task 2 of Multiple-file Analysis.

**Exercise uses different collar data**

2. Add code to the loop to calculate the minimum and maximum latitude in the file, and store these values, along with the name of the file, in a data frame. Show the data frame as output.

```
# Edit/add/try out R code here
```

If you're interested in seeing another application of for loops, check out the code below used to simulate the data for this exercise using for loops.

```
individuals = paste(c('A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J'), c(1:10), sep =
for (individual in individuals) {
    lat = vector("numeric", 24)
    long = vector("numeric", 24)
    lat[1] = rnorm(1, mean = 26, sd = 2)
    long[1] = rnorm(1, mean = -35, sd = 3)
    for (i in 2:24) {
        lat[i] = lat[i - 1] + rnorm(1, mean = 0, sd = 1)
        long[i] = long[i - 1] + rnorm(1, mean = 0, sd = 1)
    }
    times = seq(from=as.POSIXct("2016-02-26 00:00", tz="UTC"),
                to=as.POSIXct("2016-02-26 23:00", tz="UTC"),
                by="hour")
    df = data.frame(date = "2016-02-26",
                    collar = individual,
                    time = times,
                    lat = lat,
                    long = long)
    write.csv(df, paste("collar-data-", individual, "-2016-02-26.txt", sep = ""))
}
zip("data/individual_collar_data.zip", list.files(pattern = "collar-data-[A-Z][0-9]+-.*"
```

## 13.6 Subsetting Data

- Loops can subset in ways that are difficult with things like `group_by`

- Look at some data on trees from the National Ecological Observatory Network

```
library(ggplot2)
library(dplyr)

root_srcfile <- "https://raw.githubusercontent.com/DanielEWeeks/HuGen2071/main/"
download.file(paste0(root_srcfile,"data/harv_034subplt.csv"),
              "harv_034subplt.csv")
neon_trees <- read.csv('harv_034subplt.csv')
head(neon_trees)
ggplot(neon_trees, aes(x = easting, y = northing)) +
  geom_point()
```

- Look at a north-south gradient in number of trees
- Need to know number of trees in each band of y values
- Start by defining the size of the window we want to use
  - Use the grid lines which are 2.5 m

```
(window_size <- 2.5)
```

- Then figure out the edges for each window

```
(south_edges <- seq(4713095, 4713117.5, by = window_size))
(north_edges <- south_edges + window_size)
```

- But we don't want to go all the way to the far edge

```
(south_edges <- seq(4713095, 4713117.5 - window_size, by = window_size))
(north_edges <- south_edges + window_size)
```

- Set up an empty data frame to store the output

```
(counts <- vector(mode = "numeric", length = length(south_edges)))
```

- Look over the left edges and subset the data occuring within each window

```
for (i in 1:length(south_edges)) {
  data_in_window <- filter(neon_trees, northing >= south_edges[i], northing < north_edges[
  counts[i] <- nrow(data_in_window)
}
counts
```

```
yedges <- unique(c(south_edges, north_edges))
ggplot(neon_trees, aes(x = easting, y = northing)) +
  geom_point() +
  geom_hline(yintercept = yedges) +
  scale_y_reverse()
```

## 13.7 Nested Loops

- Sometimes need to loop over multiple things in a coordinate fashion
- Pass a window over some spatial data
- Look at full spatial pattern not just east-west gradient
- Basic nested loops work by putting one loop inside another one

```
for (i in 1:3) {
  for (j in 1:2) {
    print(paste("i = " , i, "; j = ", j))
  }
}
```

- Loop over x and y coordinates to create boxes
- Need top and bottom edges

```
(east_edges <- seq(731752.5, 731772.5 - window_size, by = window_size))
(west_edges <- east_edges + window_size)
```

- Redefine out storage

```
(output <- matrix(nrow = length(south_edges), ncol = length(east_edges)))
```

```
for (i in 1:length(south_edges)) {
  for (j in 1:length(east_edges)) {
    data_in_window <- filter(neon_trees,
                             northing >= south_edges[i], northing < north_edges[i],
                             easting >= east_edges[j], easting < west_edges[j],)
    output[i, j] <- nrow(data_in_window)
  }
}
```

```
output

xedges <- unique(c(east_edges, west_edges))
yedges <- unique(c(south_edges, north_edges))
ggplot(neon_trees, aes(x = easting, y = northing)) +
  geom_point() +
  geom_vline(xintercept=xedges) +
  geom_hline(yintercept = yedges) +
  scale_y_reverse()
```

## 13.8 Sequence along

- `seq_along()` generates a vector of numbers from 1 to `length(volumes)`

```
1:length(east_edges)
seq_along(east_edges)
```

# 14 Conditionals in R

## 14.1 Acknowledgment/License

The original source for this chapter was from the web site

https://datacarpentry.org/semester-biology/

which was built using this underlying code

https://github.com/datacarpentry/semester-biology

and is used under the

Attribution 4.0 International (CC BY 4.0)

license https://creativecommons.org/licenses/by/4.0/.

The material presented here has been modified from the original source.

Accordingly this chapter is made available under the same license terms.

## 14.2 Source code

If you'd like to work within R Studio using the source code of this chapter, you can obtain it from here.

## 14.3 Conditionals

- Conditional statements are when we check to see if some condition is true or not
- We used these for filtering data in `dplyr`

```
weight <- 65
species <- "DM"
weight > 50
species == "DM"
```

- These statements generate a value is of type `"logical"`
- The value is `TRUE` if the condition is satisfied
- The value is `FALSE` if the condition is not satisfied
- These aren't the strings "TRUE" and "FALSE"
- They are a special type of value
- Conditional statements are made with a range of operators
- We've seen
  - `==` for equals
  - `!=` for not equals
  - `<`, `>` for less than and greater than
  - `<=`, `>=` for less than or equal to and greater than or equal to
  - `is.na()` for is this value null
- There are others, including `%in%`, which checks to see if a value is present in a vector of possible values

```
10 >= 5
is.na(5)
"DM" %in% c("DM", "DO", "DS")
"PP" %in% c("DM", "DO", "DS")
```

- We can combine conditions using "and" and "or"
- We use the `&` for "and"
- Which means if both conditions are `TRUE` return `TRUE`
- If one of the contions is `FALSE` then return `FALSE`

```
5 > 2 & 6 >=10
```

- We use the `|` for "or"
- Which means if either or both of the conditions are `TRUE` return `TRUE`

```
5 > 2 | 6 >=10
```

- Vectors of values compared to a single value return one logical per value

```
c(1, 1, 2, 3, 1) == 1
```

- Checks each value to see if equal to 1
- This is what subsetting approaches use to subset

- They keep the values where the value in this condition vector is equal to `TRUE`
- Let's look at an example where we have a vector of sites and a vector the the states they occur in

```
(site = c('a', 'b', 'c', 'd'))
(state = c('FL', 'FL', 'GA', 'AL'))
```

- A conditional statement checking if the state is `'FL'` returns a vector of `TRUE`'s and `FALSE`s

```
state == 'FL'
```

- So when we filter the `site` vector to only return values where the `state` is equal to `'FL'`

```
site[state == 'FL']
```

- It is the same as pass a vector of `TRUE` and `FALSE` values inside the square brackets

```
site[c(TRUE, TRUE, FALSE, FALSE)]
```

- This keeps the first and second values in `site` because the values in the vector are `TRUE`
- This is how `dplyr::filter()` and other methods for subsetting data work

### 14.3.1 Tasks: Choice Operators

> **!** Important
>
> **Do Tasks 1-4 in Choice Operators**
> Create the following variables.
>
> ```
> (w <- 10.2)
> (x <- 1.3)
> (y <- 2.8)
> (z <- 17.5)
> (colors <- c("red", "blue", "green"))
> (masses <- c(45.2, 36.1, 27.8, 81.6, 42.4))
> ```
>
> Use them to print whether or not the following statements are `TRUE` or `FALSE`.
>
> 1. `w` is greater than 10
> 2. `"green"` is in `colors`

3. `x` is greater than `y`
4. Each value in `masses` is greater than 40.

```
# Edit/add/try out R code here
```

## 14.4 `if` statements

- Conditional statements generate logical values to filter inputs.
- `if` statements use conditional statements to control flow of the program.

```
if (the conditional statement is TRUE ) {
  do something
}
```

- Example

```
x = 6
if (x > 5){
  x = x^2
}
x
```

- `x > 5` is `TRUE`, so the code in the `if` runs
- `x` is now 6^2 or 36
- Change `x` to 4

```
x = 4
if (x > 5){
  x = x^2
}
x
```

- `x > 5` is `FALSE`, so the code in the `if` doesn't run
- `x` is still 4
- This is *not* a function, so everything that happens in the if statement influences the global environment
- Different mass calculations for different vegetation types

```
veg_type <- "shrub"
volume <- 16.08
if (veg_type == "shrub") {
  mass <- 2.65 * volume^0.9
  }
mass
```

### 14.4.1 Task 1: Basic If Statements

> **!** Important
>
> **Do Task 1 in Basic If Statements**
> 1. Complete (i.e., copy into your code and them modify) the following `if` statement so that if `age_class` is equal to "sapling" it sets `y <- 10`.
>
> ```
> age_class = "sapling"
> if (){
>
> }
> y
> ```

- Often want to chose one of several options
- Can add more conditions and associated actions with `else if`

```
veg_type <- "grass"
volume <- 16.08
if (veg_type == "shrub") {
  mass <- 2.65 * volume^0.9
} else if (veg_type == "grass") {
  mass <- 0.65 * volume^1.2
}
mass
```

- Checks the first condition
- If `TRUE` runs that condition's code and skips the rest
- If not it checks the next one until it runs out of conditions
- Can specify what to do if none of the conditions is `TRUE` using `else` on its own

```
veg_type <- "tree"
volume <- 16.08
if (veg_type == "shrub") {
  mass <- 2.65 * volume^0.9
} else if (veg_type == "grass") {
  mass <- 0.65 * volume^1.2
} else {
  mass <- NA
}
mass
```

### 14.4.2 Tasks 2-3: Basic If Statements

> **!** Important
>
> **Do Tasks 2-3 in Basic If Statements**
> 2. Complete the following `if` statement so that if `age_class` is equal to "sapling" it sets `y <- 10` and if `age_class` is equal to "seedling" it sets `y <- 5`.
>
> ```
> age_class = "seedling"
> if (){
>
> }
> y
> ```
>
> 3. Complete the following `if` statement so that if `age_class` is equal to "sapling" it sets `y <- 10` and if `age_class` is equal to "seedling" it sets `y <- 5` and if `age_class` is something else then it sets the value of `y <- 0`.
>
> ```
> age_class = "adult"
> if (){
>
> }
> y
> ```

## 14.5 Multiple ifs vs else if

- Multiple ifs check each conditional separately
- Executes code of all conditions that are `TRUE`

```
x <- 5
if (x > 2){
  x * 2
}
if (x > 4){
  x * 4
}
x
```

- `else if` checks each condition sequentially
- Executes code for the first condition that is `TRUE`

```
x <- 5
if (x > 2){
  x * 2
} else if (x > 4){
  x * 4
}
x
```

## 14.6 Using Conditionals Inside Functions

- We've used a conditional to estimate mass differently for different types of vegetation
- This is the kind of code we are going to want to reuse, so let's move it into a function
- We do this by placing the same code inside of a function
- And making sure that the function takes all required variables as input

```
est_mass <- function(volume, veg_type){
  if (veg_type == "shrub") {
    mass <- 2.65 * volume^0.9
  } else if (veg_type == "grass") {
    mass <- 0.65 * volume^1.2
  } else {
    mass <- NA
  }
  return(mass)
}
```

- We can then run this function with different vegetation types and get different estimates for mass

```
est_mass(1.6, "shrub")
est_mass(1.6, "grass")
est_mass(1.6, "tree")
```

- Let's walk through how this code executes using the debugger
- When we call the function the first thing that happens is that 1.6 gets assigned to `volume` and `"tree"` gets assigned to `veg_type`
- The code then checks to see if `veg_type` is equal to `"shrub"`
- It isn't so the code then checks to see if `veg_type` is equal to `"grass"`
- It isn't so the code then hits the `else` statement and executes the code in the `else` block
- It assigns `NA` to mass
- It then finishes the if/else if/else statement and returns the value for `mass`, which is `NA` to the global environment

### 14.6.1 Task: Size Estimates by Name

> **!** Important
>
> **Do Size Estimates by Name**
>
> #### 14.6.1.1 Part I
>
> The length of an organism is typically strongly correlated with its body mass. This is useful because it allows us to estimate the mass of an organism even if we only know its length. This relationship generally takes the form:
>
> mass = a * length^b
>
> Where the parameters `a` and `b` vary among groups. This allometric approach is regularly used to estimate the mass of dinosaurs since we cannot weigh something that is only preserved as bones.
> The following function estimates the mass of an organism in kg based on its length in meters for a particular set of parameter values, those for *Theropoda* (where `a` has been estimated as `0.73` and `b` has been estimated as `3.63`; Seebacher 2001).
>
> ```
> get_mass_from_length_theropoda <- function(length){
>   mass <- 0.73 * length ^ 3.63
>   return(mass)
> }
> ```
>
> 1. Use this function to print out the mass of a Spinosaurus that is 16 m long based on its reassembled skeleton.

```
# Edit/add/try out R code here
```

2. Create a new version of this function called `get_mass_from_length()` that takes `length`, `a` and `b` as arguments and uses the following code to estimate the mass `mass <- a * length ^ b`. Use this function to estimate the mass of a Sauropoda (`a = 214.44`, `b = 1.46`) that is 26 m long.

```
# Edit/add/try out R code here
```

#### 14.6.1.2 Part II

To make it even easier to work with your dinosaur size estimation functions you decide to create a function that lets you specify which dinosaur group you need to estimate the size of by name and then have the function automatically choose the right parameters. Create a new function `get_mass_from_length_by_name()` that takes two arguments, the `length` and the name of the dinosaur group. Inside this function use `if`/`else if`/`else` statements to check to see if the name is one of the following values and if so use the associated `a` and `b` values to estimate the species mass.

- *Stegosauria*: `a` = `10.95` and `b` = `2.64` (Seebacher 2001).
- *Theropoda*: `a` = `0.73` and `b` = `3.63` (Seebacher 2001).
- *Sauropoda*: `a` = `214.44` and `b` = `1.46` (Seebacher 2001).

If the name is not any of these values the function should return `NA`.
Run the function for: 1. A *Stegosauria* that is 10 meters long. 2. A *Theropoda* that is 8 meters long. 3. A *Sauropoda* that is 12 meters long. 4. A *Ankylosauria* that is 13 meters long.

```
# Edit/add/try out R code here
```

*Challenge (**optional**)*: If the name is not one of values that have `a` and `b` values print out a message that it doesn't know how to convert that group that includes that groups name in a message like "No known estimation for Ankylosauria". (the function `paste()` will be helpful here). Doing this successfully will modify your answer to (4), which is fine.

```
# Edit/add/try out R code here
```

*Challenge (**optional**)*: Change your function so that it uses two different values of `a` and `b` for *Stegosauria*. When *Stegosauria* is greater than 8 meters long use the equation above.

When it is less than 8 meters long use `a = 8.5` and `b = 2.8`. Run the function for a *Stegosauria* that is 6 meters long.

```
# Edit/add/try out R code here
```

*Challenge (**optional**)*: Rewrite your function so that instead of calculating mass directly it sets the values of `a` and `b` to the values for the species (or to `NA` if the species doesn't have an equation) and then calls another function to do the basic `mass = a * length ^ b` calculation.

```
# Edit/add/try out R code here
```

## 14.7 Automatically extracting functions

- Can pull code out into functions
- Highlight the code
- Code -> Extract Function
- Provide a name for the function

## 14.8 Nested conditionals

- Sometimes decisions are more complicated
- For example we might have different equations for some vegetation types based on the age of the plant
- Can "nest" conditionals inside of one another

```
est_mass <- function(volume, veg_type, age){
  if (veg_type == "shrub") {
    if (age < 5) {
      mass <- 1.6 * volume^0.8
    } else {
      mass <- 2.65 * volume^0.9
  }
  } else if (veg_type == "grass" | veg_type == "sedge") {
    mass <- 0.65 * volume^1.2
  } else {
    mass <- NA
  }
```

```
  return(mass)
}

est_mass(1.6, "shrub", age = 2)
est_mass(1.6, "shrub", age = 6)
```

- First checks if the vegetation type is "shrub"
- If it is checks to see if it is $< 5$ years old
- If so does one calculation, if not does another
- But nesting can be difficult to follow so try to minimize it

### 14.8.1 Task 4: Basic If Statements

> **!** Important
>
> **Do Task 4 in Basic If Statements**
> 4. Convert your conditional statement from Task 3 in Section 14.4.2 into a function that takes `age_class` as an argument and returns `y`. Call this function 5 times, once with each of the following values for `age_class`: "sapling", "seedling", "adult", "mature", "established".
>
> ```
> # Edit/add/try out R code here
> ```

# 15 Functions

## 15.1 Acknowledgment/License

The original source for this chapter was from the web site

https://datacarpentry.org/semester-biology/

which was built using this underlying code

https://github.com/datacarpentry/semester-biology

and is used under the

Attribution 4.0 International (CC BY 4.0)

license https://creativecommons.org/licenses/by/4.0/.

The material presented here has been modified from the original source.

Accordingly this chapter is made available under the same license terms.

## 15.2 Source code

If you'd like to work within R Studio using the source code of this chapter, you can obtain it from here.

## 15.3 Understandable and reusable code

- Write code in understandable chunks.
- Write reusable code.

## 15.4 Understandable chunks

- Human brain can only hold limited number of things in memory
- Write programs that don't require remembering all of the details at once
- Treat functions as a single conceptual chunk.

## 15.5 Reuse

- Want to do the same thing repeatedly?
  - Inefficient & error prone to copy code
  - If it occurs in more than one place, it will eventually be wrong somewhere.
- Functions are written to be reusable.

## 15.6 Function basics

```
function_name <- function(inputs) {
  output_value <- do_something(inputs)
  return(output_value)
}
```

- The braces indicate that the lines of code are a group that gets run together

```
{a = 2
b = 3
a + b}
```

- Pressing run anywhere in this group runs all the lines in that group
- A function runs all of the lines of code in the braces
- Using the arguments provided
- And then returns the output

```
calc_shrub_vol <- function(length, width, height) {
  area <- length * width
  volume <- area * height
  return(volume)
}
class(calc_shrub_vol)
```

- Creating a function doesn't run it.
- Call the function with some arguments.

```
calc_shrub_vol(0.8, 1.6, 2.0)
```

- Store the output to use it later in the program

```
(shrub_vol <- calc_shrub_vol(0.8, 1.6, 2.0))
```

**!** Do Writing Functions

Edit the following function to replace the `________` with variables names for the input and output.

```
convert_pounds_to_grams <- function(________) {
    grams = 453.6 * pounds
    return(________)
}
```

Use the function to calculate how many grams there are in 3.75 pounds.

```
# Edit/add/try out R code here
```

- Treat functions like a black box
  - *Draw a box on board showing inputs->function->outputs*
  - The only things the function knows about are the inputs we pass it
  - The only thing the program knows about the function is the output it produces
- Walk through function execution (using debugger)
  - Call function
  - Assign 0.8 to length, 1.6 to width, and 2.0 to height inside function
  - Calculate the area and assign it to `area`
  - Calculate volume and assign it to `volume`
  - Send `volume` back as output
  - Store it in `shrub_vol`
- Treat functions like a black box.
  - Can't access a variable that was created in a function
    - `> volume`
    - `Error: object 'width' not found`

- Or an argument by name
  - `> width`
  - `Error: object 'width' not found`
- 'Global' variables can influence function, but should not.
  - Very confusing and error prone to use a variable that isn't passed in as an argument

**!** Do Use and Modify.

The length of an organism is typically strongly correlated with its body mass. This is useful because it allows us to estimate the mass of an organism even if we only know its length. This relationship generally takes the form:

mass = a * length^b

Where the parameters `a` and `b` vary among groups. This allometric approach is regularly used to estimate the mass of dinosaurs since we cannot weigh something that is only preserved as bones.
The following function estimates the mass of an organism in kg based on its length in meters for a particular set of parameter values, those for *Theropoda* (where `a` has been estimated as `0.73` and `b` has been estimated as `3.63`; Seebacher 2001).

```
get_mass_from_length_theropoda <- function(length){
  mass <- 0.73 * length ^ 3.63
  return(mass)
}
class(get_mass_from_length_theropoda)
```

1. Use this function to print out the mass of a Spinosaurus that is 16 m long based on its reassembled skeleton.

```
# Edit/add/try out R code here
```

2. Create a new version of this function called `get_mass_from_length()` that takes `length`, `a` and `b` as arguments and uses the following code to estimate the mass `mass <- a * length ^ b`.

```
# Edit/add/try out R code here
```

Use this function to estimate the mass of a Sauropoda (`a = 214.44`, `b = 1.46`) that is 26 m long.

```
# Edit/add/try out R code here
```

## 15.7 Default arguments

- Defaults can be set for common inputs.
- For example, many of our shrubs are the same height so for those shrubs we only measure the `length` and `width`.
- So we want a default value for the `height` for cases where we don't measure it

```
calc_shrub_vol <- function(length, width, height = 1) {
  area <- length * width
  volume <- area * height
  return(volume)
}

calc_shrub_vol(0.8, 1.6)
calc_shrub_vol(0.8, 1.6, 2.0)
calc_shrub_vol(length = 0.8, width = 1.6, height = 2.0)
```

**!** Do Default Arguments.

This is a follow up to the Use and Modify exercise above.
Allowing `a` and `b` to be passed as arguments to `get_mass_from_length()` made the function more flexible, but for some types of dinosaurs we don't have specific values of `a` and `b` and so we have to use general values that can be applied to a number of different species.
Rewrite your `get_mass_from length()` function from Use and Modify so that its arguments have default values of `a = 39.9` and `b = 2.6` (the average values from Seebacher 2001).

```
# Edit/add/try out R code here
```

1. Use this function to estimate the mass of a Sauropoda (`a = 214.44`, `b = 1.46`) that is 22 m long (by setting `a` and `b` when calling the function).

```
# Edit/add/try out R code here
```

2. Use this function to estimate the mass of a dinosaur from an unknown taxonomic group that is 16m long. Only pass the function `length`, not `a` and `b`, so that the default values are used.

```
# Edit/add/try out R code here
```

*Discuss why passing `a` and `b` in is more useful than having them fixed*

## 15.8 Named vs unnamed arguments

- When to use or not use argument names

```
calc_shrub_vol(length = 0.8, width = 1.6, height = 2.0)
```

Or

```
calc_shrub_vol(0.8, 1.6, 2.0)
```

- You can always use names
  - Value gets assigned to variable of that name
- If not using names then order determines naming
  - First value is `length`, second value is `width`, third value is `height`
  - If order is hard to remember use names
- In many cases there are *a lot* of optional arguments
  - Convention to always name optional argument
- So, in our case, the most common approach would be

```
calc_shrub_vol(0.8, 1.6, height = 2.0)
```

## 15.9 Combining Functions

- Each function should be single conceptual chunk of code
- Functions can be combined to do larger tasks in two ways
- Calling multiple functions in a row

```
est_shrub_mass <- function(volume){
  mass <- 2.65 * volume^0.9
}

(shrub_volume <- calc_shrub_vol(0.8, 1.6, 2.0))
(shrub_mass <- est_shrub_mass(shrub_volume))
```

- We can also use pipes with our own functions
- The output from the first function becomes the first argument for the second function

```
library(dplyr)
(shrub_mass <- calc_shrub_vol(0.8, 1.6, 2.0) %>%
  est_shrub_mass())
```

**!** Do Combining Functions.

This is a follow up to the Default Argument exercise above.
Measuring things using the metric system is the standard approach for scientists, but when communicating your results more broadly it may be useful to use different units (at least in some countries). Write a function called `convert_kg_to_pounds` that converts kilograms into pounds (`pounds = 2.205 * kg`).

```
  # Edit/add/try out R code here
```

Use that function and your `get_mass_from_length()` function from Default Arguments to estimate the weight, in pounds, of a 12 m long Stegosaurus with `a = 10.95` and `b = 2.64` (The estimated `a` and `b` values for *Stegosauria* from Seebacher 2001).

```
  # Edit/add/try out R code here
```

- We can nest functions

```
(shrub_mass <- est_shrub_mass(calc_shrub_vol(0.8, 1.6, 2.0)))
```

- But we careful with this because it can make code difficult to read
- Don't nest more than two functions
- Can also call functions from inside other functions
- Allows organizing function calls into logical groups

```
est_shrub_mass_dim <- function(length, width, height){
  volume = calc_shrub_vol(length, width, height)
  mass <- est_shrub_mass(volume)
  return(mass)
}

est_shrub_mass_dim(0.8, 1.6, 2.0)
```

- We ***don't*** need to pass the function name into the function
- That's the one violation of the black box rule

## 15.10 Using dplyr & ggplot in functions

- There is an extra step we need to take when working with functions from dplyr and ggplot that work with "data variables", i.e., names of columns that are not in quotes
- These functions use tidy evaluation, a special type of non-standard evaluation
- This basically means they do fancy things under the surface to make them easier to work with
- But it means they don't work if we just pass things to functions in the most natural way

```
library(ggplot2)

make_plot <- function(df, column, label) {
  ggplot(data = df, mapping = aes(x = column)) +
    geom_histogram() +
    xlab(label)
}

root_srcfile <- "https://raw.githubusercontent.com/DanielEWeeks/HuGen2071/main/"
download.file(paste0(root_srcfile,"data/surveys.csv"),
              "surveys.csv")
list.files(pattern = "surveys")

surveys <- read.csv("surveys.csv")
make_plot(surveys, hindfoot_length, "Hindfoot Length [mm]")
```

- To fix this we have to tell our code which inputs/arguments are this special type of data variable
- We do this by "embracing" them in double braces

```
library(ggplot2)

make_plot <- function(df, column, label) {
  ggplot(data = df, mapping = aes(x = {{ column }})) +
    geom_histogram() +
    xlab(label)
}
```

```
surveys <- read.csv("surveys.csv")
make_plot(surveys, hindfoot_length, "Hindfoot Length [mm]")
make_plot(surveys, weight, "Weight [g]")
```

## 15.11 Code design with functions

- Functions let us break code up into logical chunks that can be understood in isolation
- Write functions at the top of your code then call them at the bottom
- The functions hold the details
- The function calls show you the outline of the code execution

```
clean_data <- function(data){
  do_stuff(data)
}

process_data <- function(cleaned_data){
  do_dplyr_stuff(cleaned_data)
}

make_graph <- function(processed_data){
  do_ggplot_stuff(processed_data)
}

raw_data <- read.csv('mydata.csv')
cleaned_data <- clean_data(raw_data)
processed_data <- process_data(cleaned_data)
make_graph(processed_data)
```

## 15.12 Documentation & Comments

- Documentation
  - How to use code
  - Use Roxygen comments for functions
- Comments
  - Why & how code works
  - Only if it code is confusing to read

## 15.13 Working with functions in RStudio

- It is possible to find and jump between functions
- Click on list of functions at bottom of editor and select
- Can be helpful to clearly see what is a function
- Can have RStudio highlight them
- Global Options -> Code -> Display -> Highlight R function calls

# 16 R Functions Excercise

## 16.1 Load Libraries

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

## 16.2 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"
```

## 16.3 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"))
}
```

## 16.4 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)
f11 <- read.table("data/dataset1.txt",sep="\t",header=TRUE)
results$dataset[1] <- "dataset1"
results$N <- nrow(f11)
results$mean[1] <- mean(f11$trait)
results$median[1] <- median(f11$trait)
results$var[1] <- var(f11$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
```

```
f12 <- read.table("data/dataset2.txt",sep="\t",header=TRUE)
results$dataset[2] <- "dataset2"
results$N <- nrow(f12)
results$mean[2] <- mean(f12$trait)
results$median[2] <- median(f12$trait)
results$var[2] <- var(f12$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
```

```
f13 <- 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.

### 16.4.1 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"
```

### 16.4.2 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

### 16.4.3 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

### 16.4.4 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)
}
```

### 16.4.5 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.

### 16.4.6 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
```

### 16.4.7 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)
  }
```

# 17 R Tidyverse Exercise

## 17.1 Load Libraries

Load the `tidyverse` packages

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

## 17.2 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> <dbl>       <dbl>        <dbl>
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.

## 17.3 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> <dbl>       <dbl>
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.

## 17.4 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 -1.21  -2.86  -6.17
2 2009-01-02 -1.43  -1.10   0.696
3 2009-01-03 -1.34   0.191  3.05
4 2009-01-04 -0.587 -1.27   0.531
5 2009-01-05 -0.102 -2.25  -2.83
6 2009-01-06 -2.46   3.04   5.60
```

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

```
# A tibble: 6 x 3
  time       stock  price
  <date>     <chr>  <dbl>
1 2009-01-01 X     -1.21
2 2009-01-02 X     -1.43
3 2009-01-03 X     -1.34
4 2009-01-04 X     -0.587
```

```
5 2009-01-05 X     -0.102
6 2009-01-06 X     -2.46
```

## 17.5 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     -1.21
2 2009-01-01 Y     -2.86
3 2009-01-01 Z     -6.17
4 2009-01-02 X     -1.43
5 2009-01-02 Y     -1.10
6 2009-01-02 Z      0.696
```

## 17.6 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> <dbl>       <dbl>        <dbl>
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> <dbl> <chr>        <dbl>
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> <dbl> <chr>        <dbl>
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> <dbl> <chr> <chr> <chr>     <dbl>
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
```

## 17.7 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> <dbl> <chr> <chr> <chr>     <dbl>
```

```
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
```

## 17.8 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

# 18 R Recoding Reshaping Exercise

## 18.1 Load Libraries

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

## 18.2 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"
```

## 18.3 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
```

## 18.4 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
```

## 18.5 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
```

## 18.6 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
```

## 18.7 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
```

### 18.7.1 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

### 18.7.2 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
```

### 18.7.3 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
```

## 18.8 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
contain list-cols.
* 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>
```

### 18.8.1 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.

## 18.9 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
```

### 18.9.1 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

### 18.9.2 `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
```

## 18.10 Exercise 4

**Skill**: Summarizing within groups

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

Expand to see solution

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

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

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

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

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

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

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

## 18.11 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
```

## 18.12 Recoding data

### Approach 2

- Implement our dictionaries using left joins

### 18.12.1 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
```

## 18.13 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
```

## 18.14 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
```

# 19 R Merging Exercise

## 19.1 Load Libraries

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

## 19.2 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
```

## 19.3 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
```

## 19.4 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
```

### 19.4.1 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
```

## 19.5 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]
# i 4 variables: subject_id <chr>, maternal_age_delivery <dbl>,
#   case_control_status <dbl>, prepregnancy_BMI <dbl>
```

## 19.6 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
```

## 19.7 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.

## 19.8 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`.

## 19.9 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.
```

# 20 R Graphics Exercise

## 20.1 Load Libraries

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

## 20.2 Exercise 1

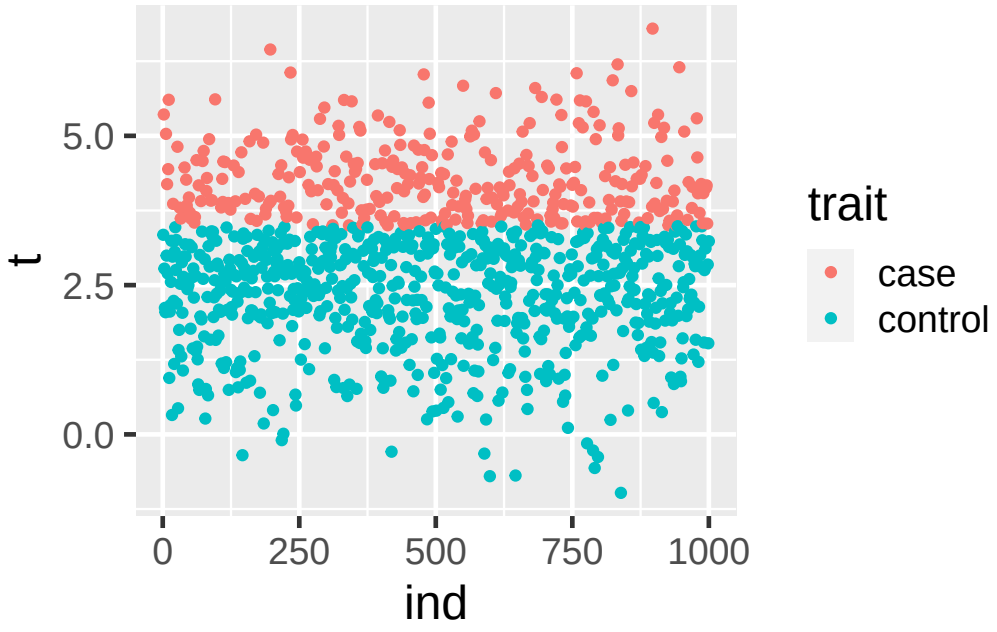
Read in and set up the data set `b`, a cleaned version of our simulated data set:

```
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()
```

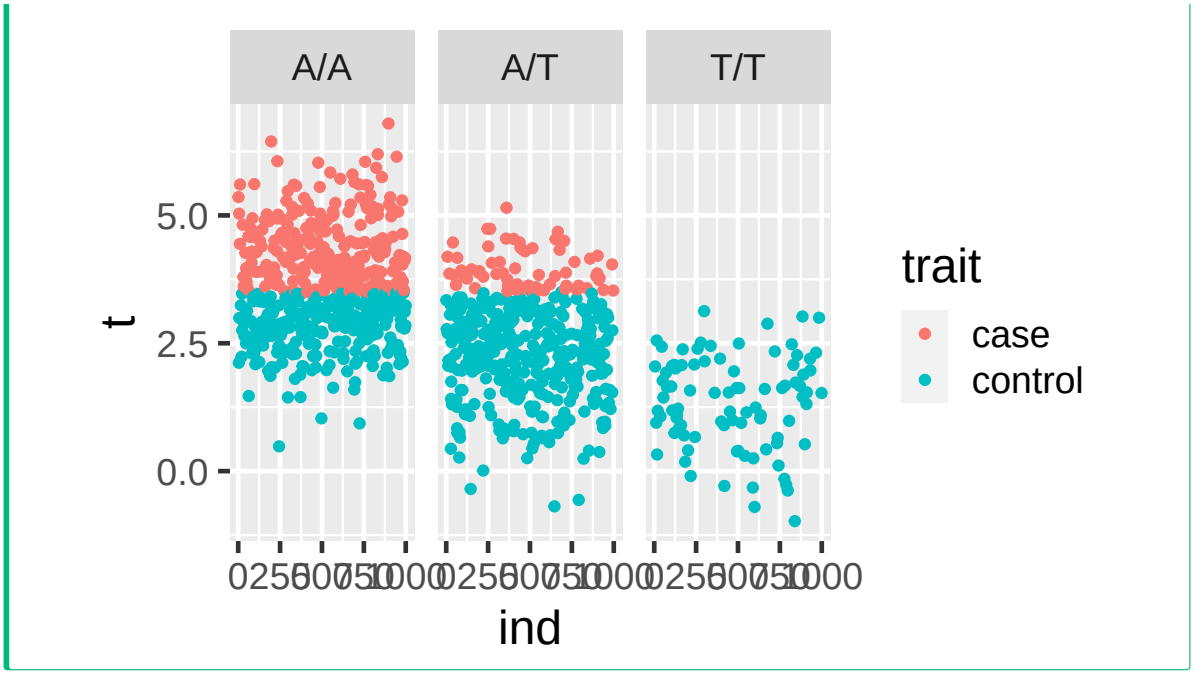



## 20.3 Exercise 2

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)
```




## 20.4 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`
```

```r
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"
```

## 20.5 Similar regression lines

These three data sets have very similar regression lines:

```r
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
```

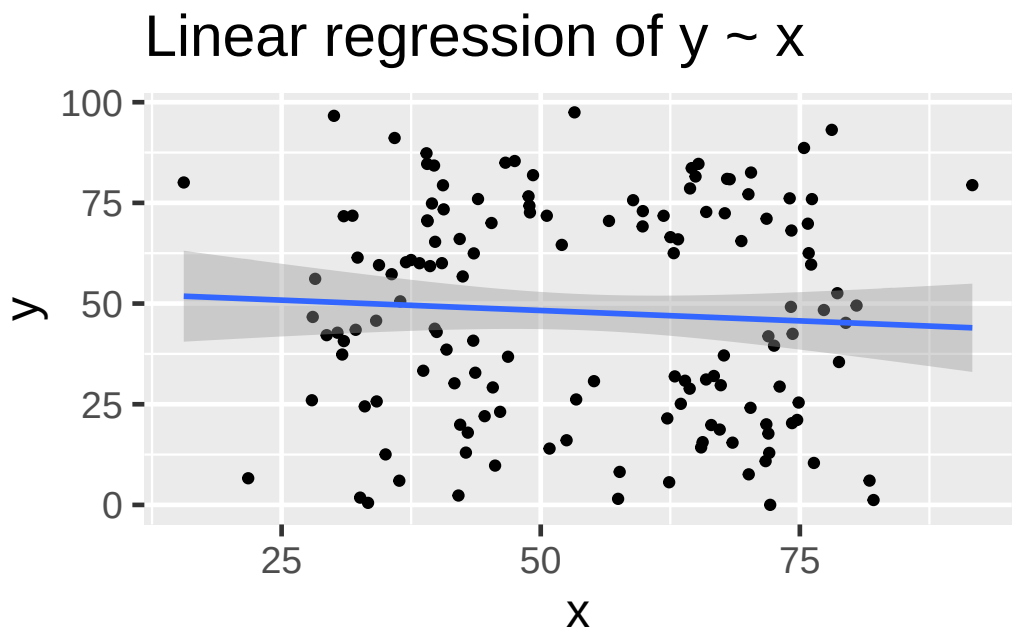
```
  summary(lm(x_1 ~ y_1, data=d)) %>%  coef()
```

```
               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")
```



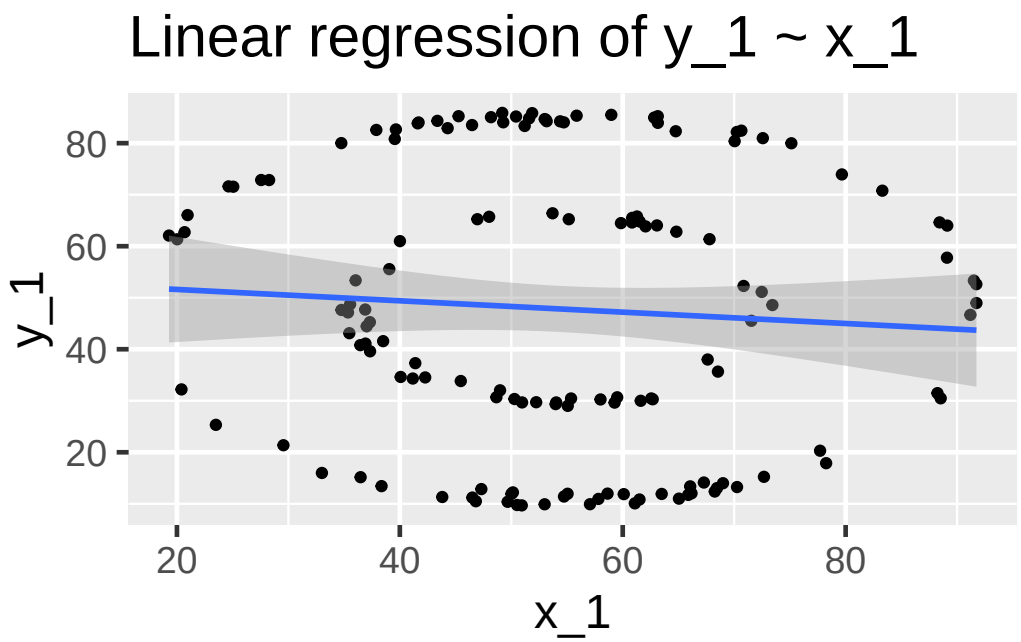

Now try this:

```
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:

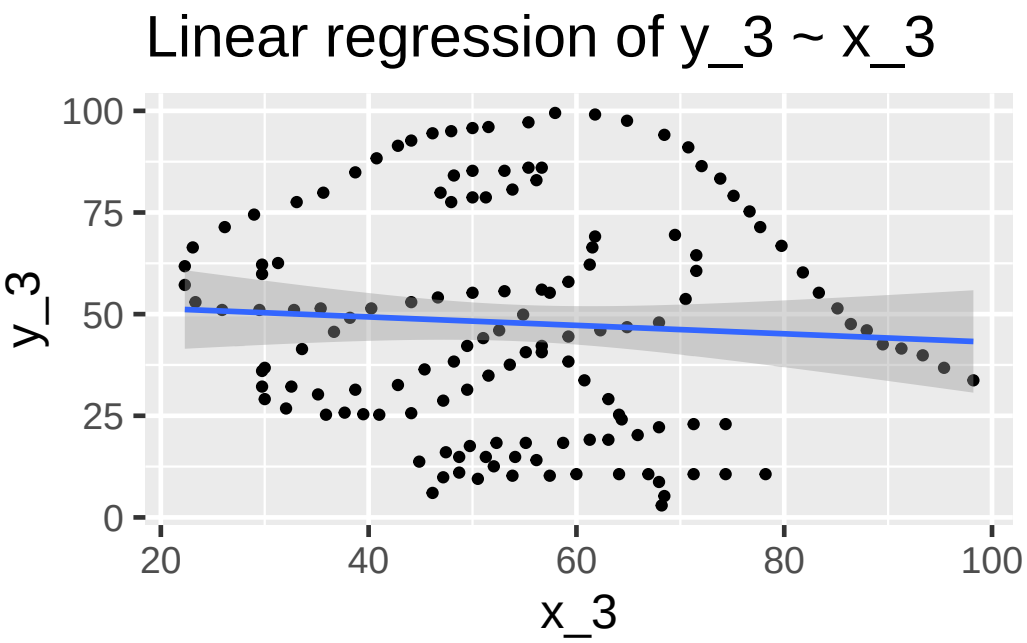
```
ggplot(d,aes(x=x_3,y=y_3)) + geom_point() +
  geom_smooth(method="lm")
```

Expand to see solution

### 20.5.1 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'
```




## 20.6 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
```

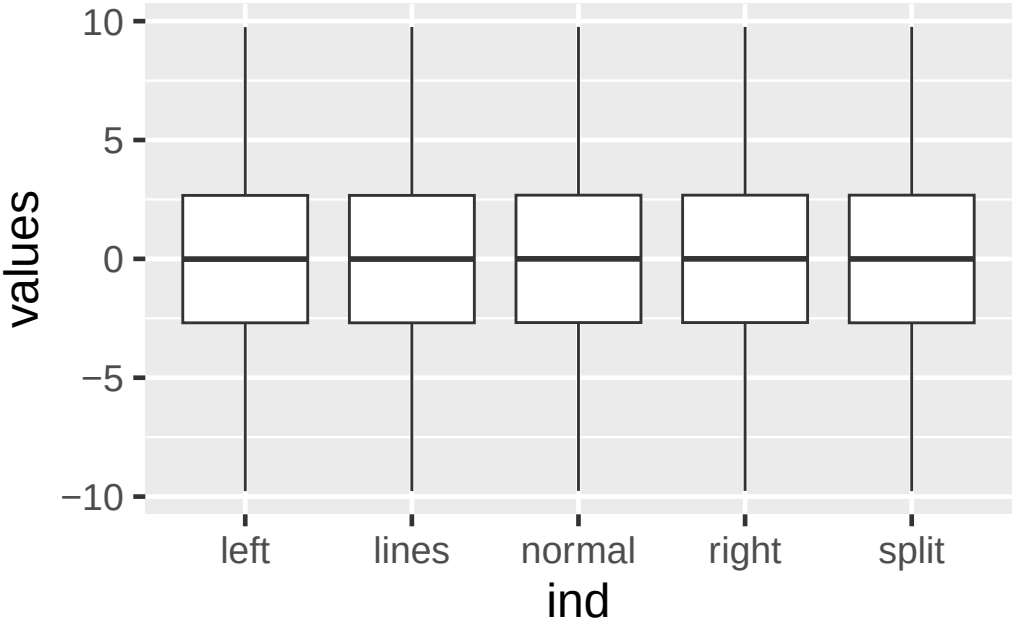
Now try this:

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

Expand to see solution

## 20.7 Identical box plots

These data have essentially identical box plots.




## 20.8 Boxplots

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

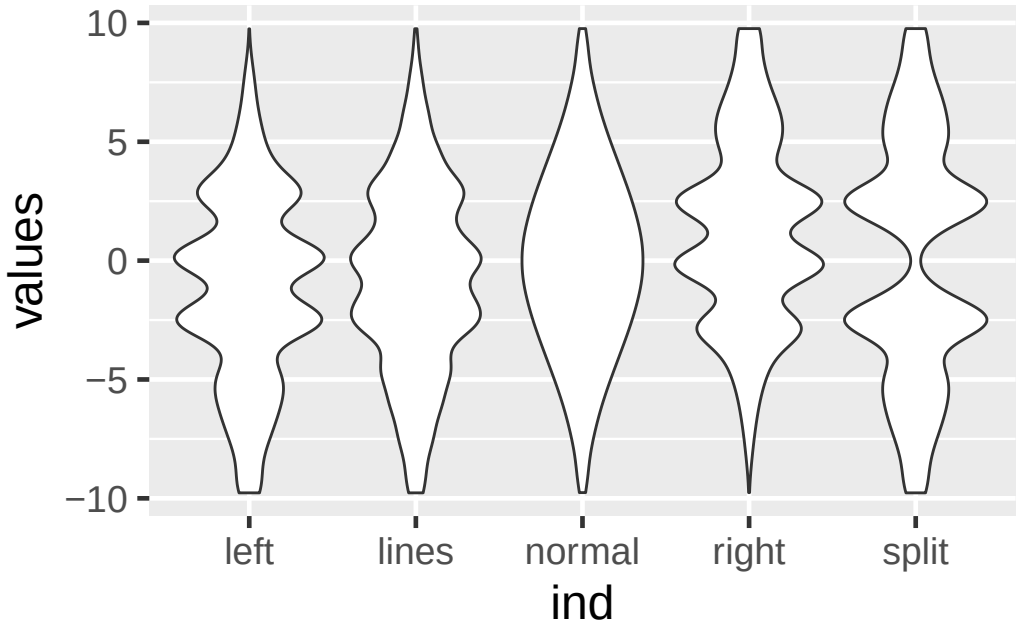
Let's try violin plots instead:

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

A violin plot is a mirrored density plot.

Expand to see solution

## 20.9 Non-identical violin plots



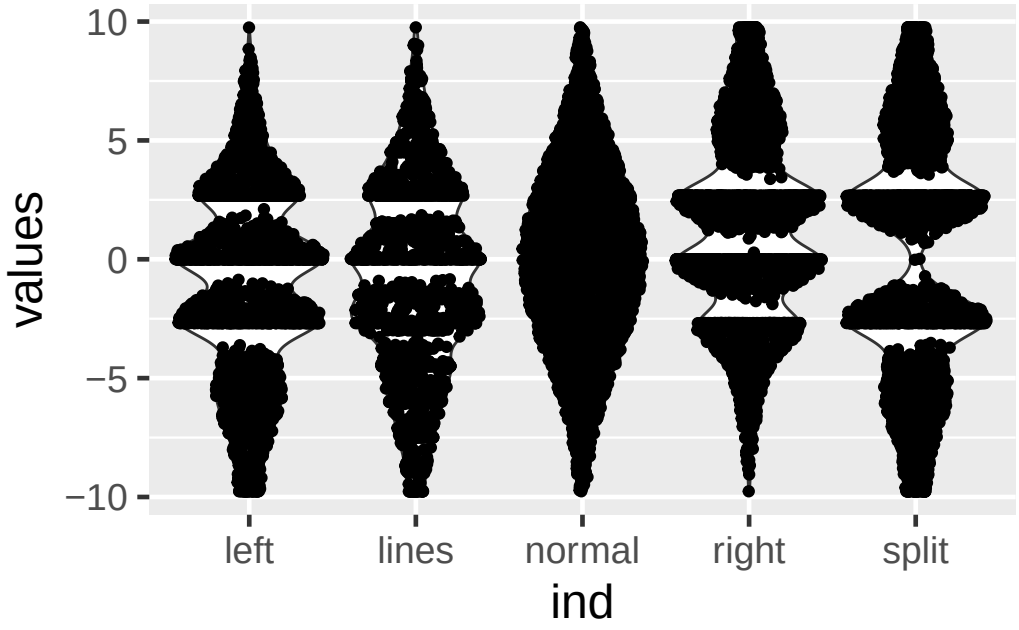

## 20.10 Sina plots

Sidiropoulos N, Sohi SH, Pedersen TL, Porse BT, Winther O, Rapin N, Bagger FO. SinaPlot: An Enhanced Chart for Simple and Truthful Representation of Single Observations Over Multiple Classes. Journal of Computational and Graphical Statistics. Taylor & Francis; 2018 Jul 3;27(3):673–676. DOI: https://doi.org/10.1080/10618600.2017.1366914

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

Expand to see solution

## 20.11 Sina plots



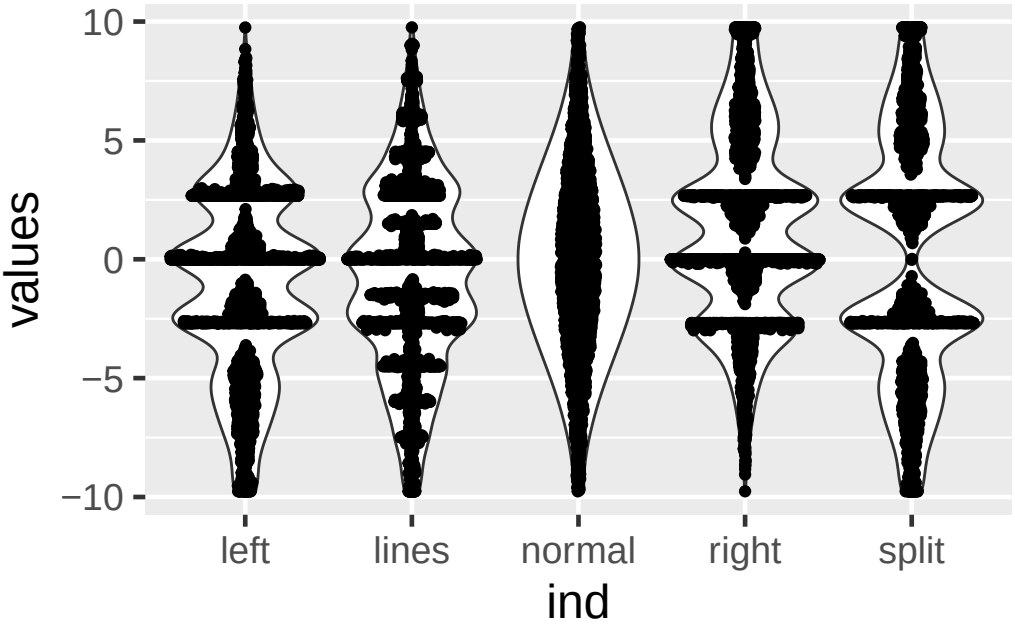

## 20.12 Sina plots

`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

### 20.13 Sina plots

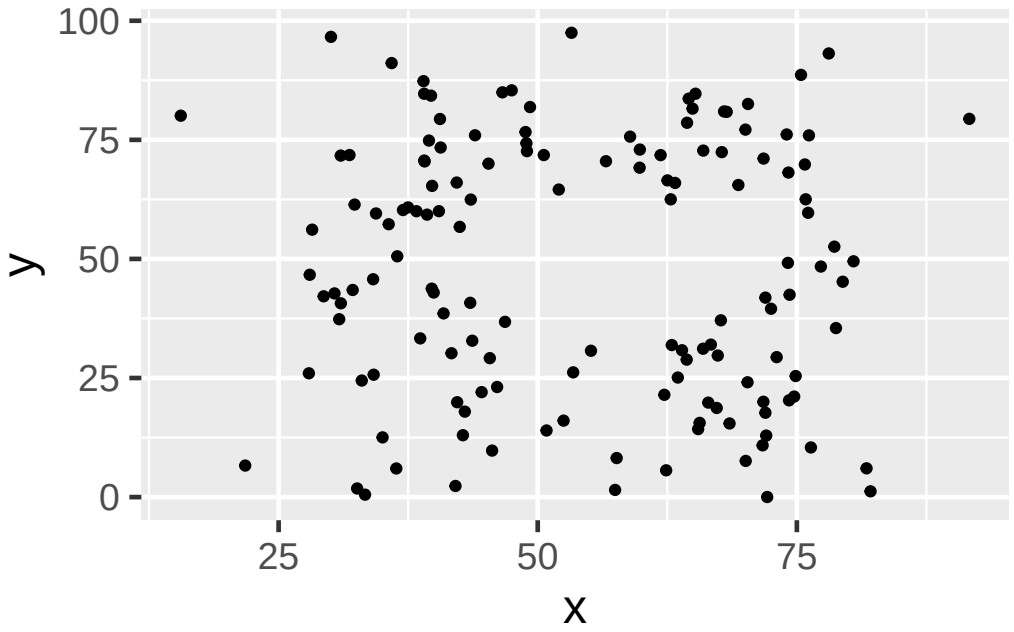


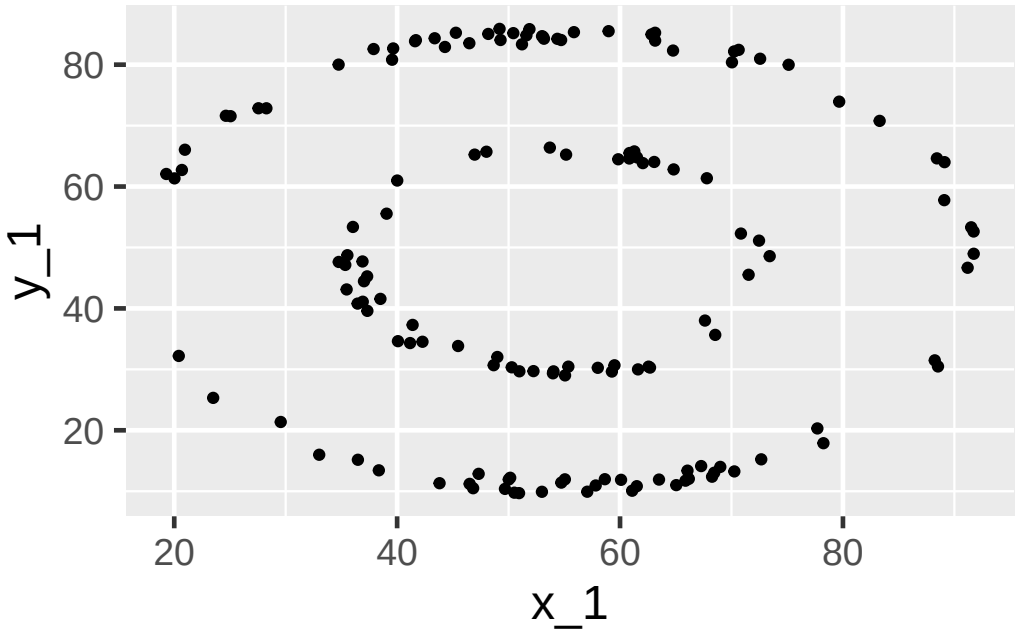

## 20.14 Drawing multiple graphs

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())
}
```



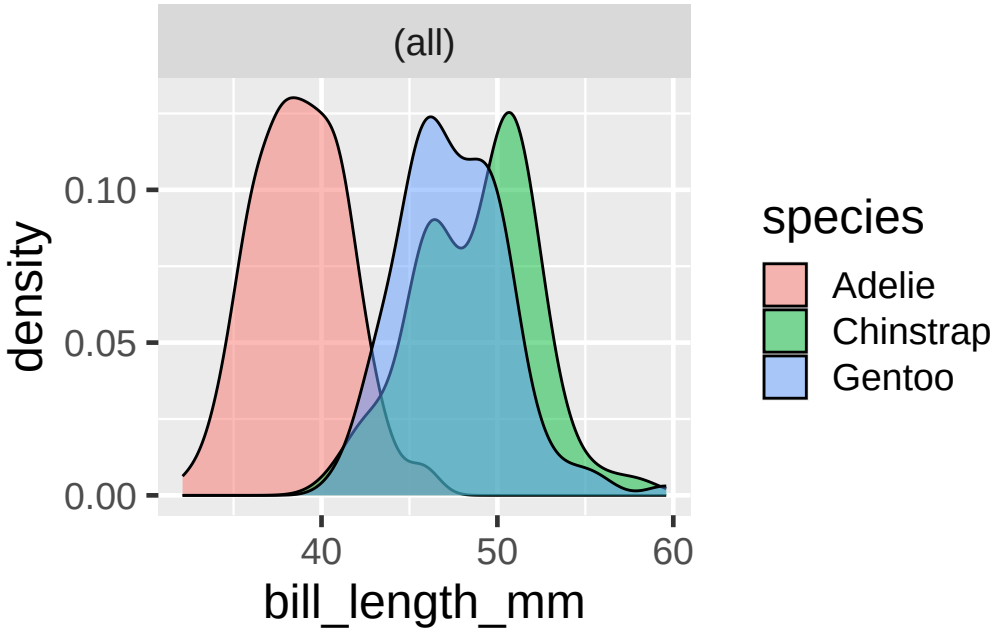

## 20.15 Writing ggplot functions

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

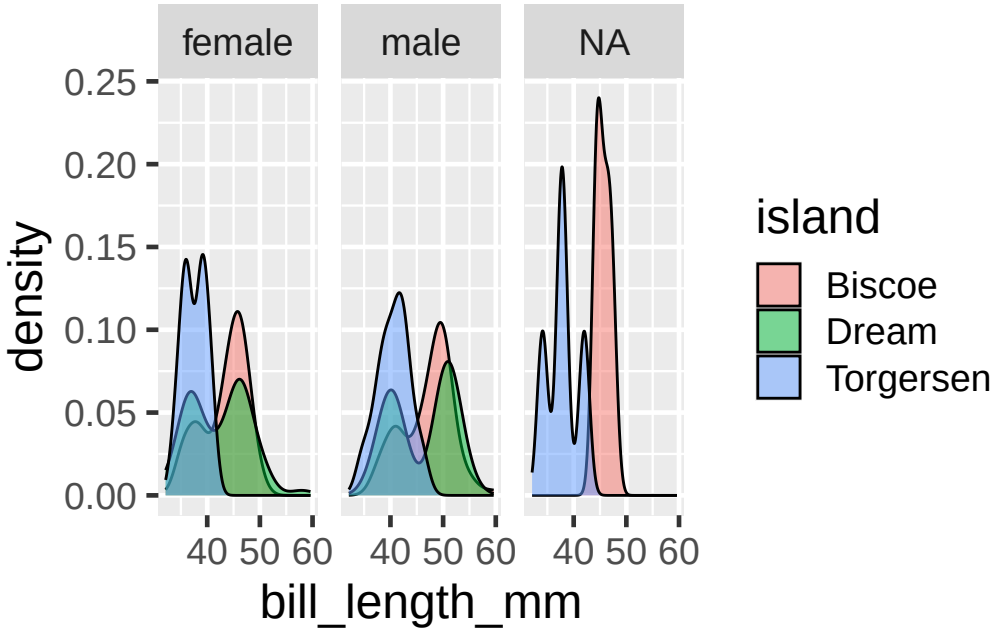
```r
library(palmerpenguins)

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

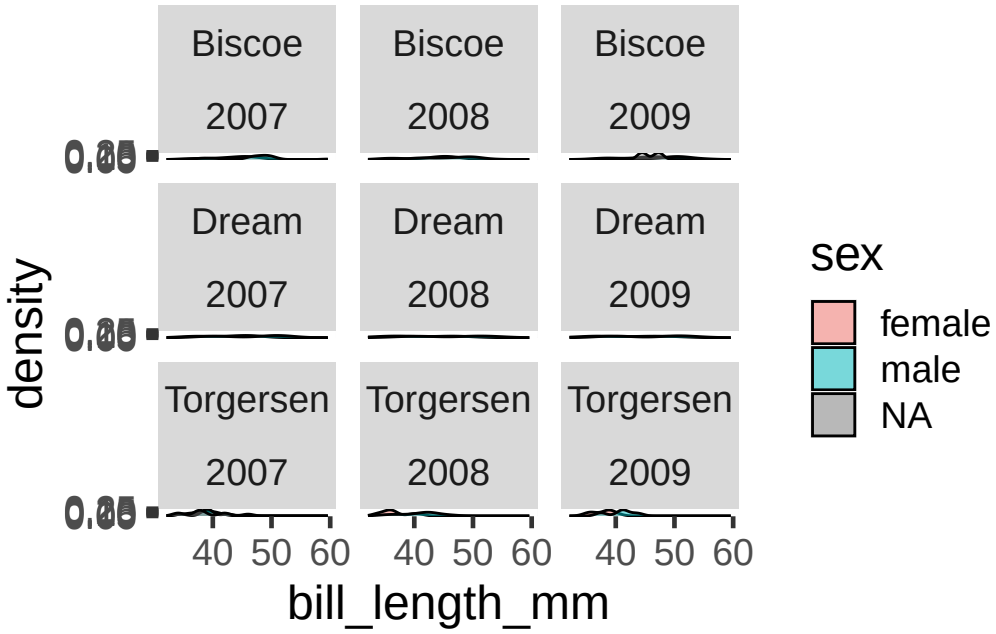
```r
PlDensity(species)
```




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




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



## 20.16 Exercise 3

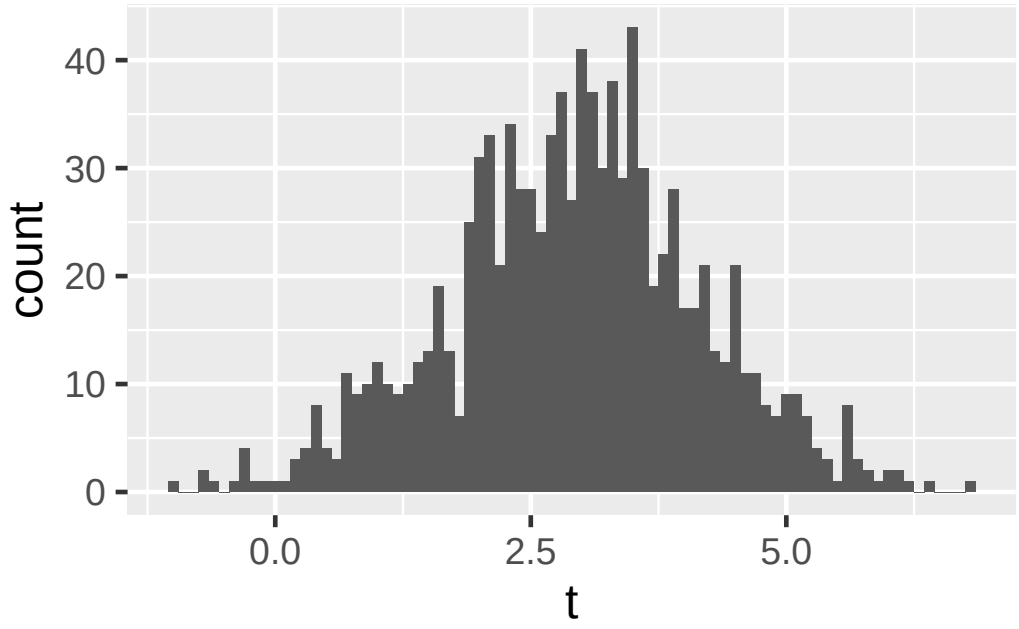
Consider this example code:

```
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)
```




### 20.16.1 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#plot-functions
- Use the `{{ var }}` approach

Expand to see solution

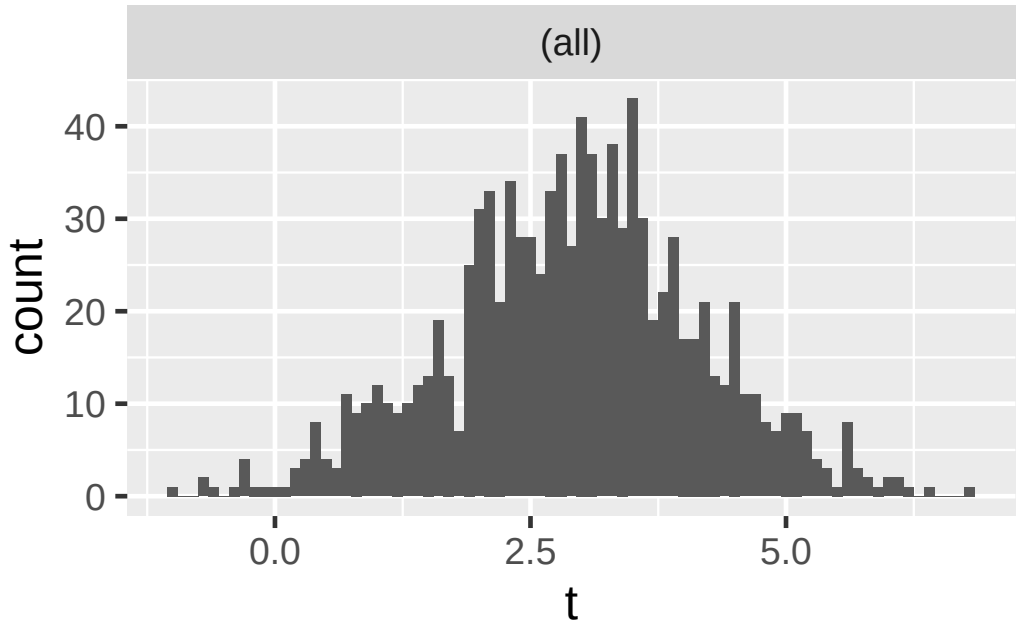
Hadley Wickham states:
You have to use the vars() syntax

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

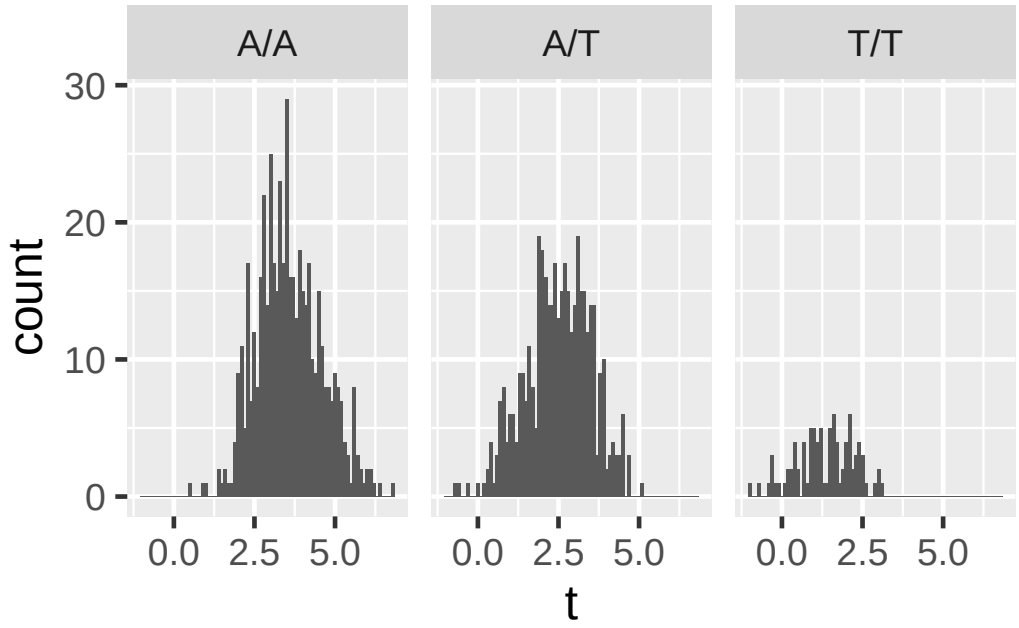
Tweet: https://twitter.com/hadleywickham/status/1574380137524887554?s=20&t=FLbwErwEKQKWtKIGufDLIQ

```
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)
```



## 20.17 Source of data

Illustrative data sets from https://www.research.autodesk.com/publications/same-stats-different-graphs/

# 21 R Reordering Exercise

## 21.1 Load Libraries

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

## 21.2 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"
```

## 21.3 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.

## 21.4 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
```

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

```
[1] 0
```

## 21.5 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

```
# 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:

```
# 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
```

## 21.6 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] "26 string mismatches"
```

## 21.7 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
```

## 21.8 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
```

## 21.9 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
```

## 21.10 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

# 22 R Exploratory Data Analysis Exercise

## 22.1 Load Libraries

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

## 22.2 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

## 22.3 Dimensions

- What are the dimensions of our data?

## 22.4 Dimensions

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

### 22.4.1 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"
```

### 22.4.2 Data dictionay `dd`

```
dim(dd)
```

```
[1] 27  5
```

```
names(dd)
```

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

## 22.5 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?

### 22.5.1 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
```

### 22.5.2 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 |

### 22.5.3 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  maternal_age_delivery case_control_status
  <chr>       <dbl> <chr>                 <dbl>                <dbl>
1 SUBJ20         35 W                      29.4                    1
2 SUBJ20         35 White                  29.4                    1
# i 16 more variables: prepregnancy_weight <dbl>, height <dbl>,
#   prepregnancy_BMI <dbl>, gravidity <dbl>, parity <dbl>,
#   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>
```

```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
```

## 22.6 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?

### 22.6.1 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
```

## 22.7 Missing data

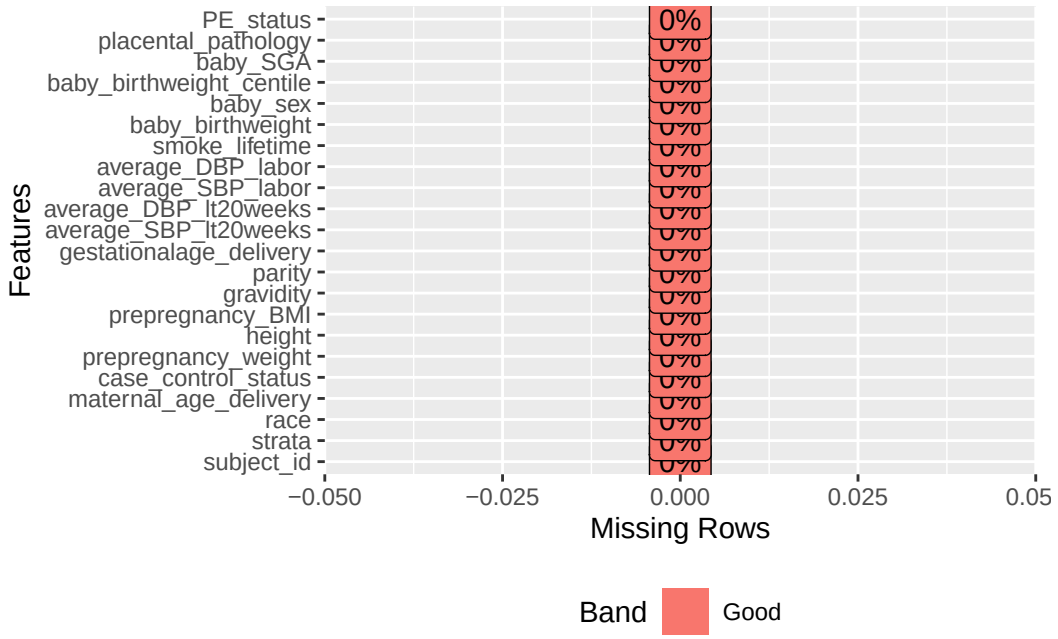
- What is the pattern of missing data?
  - How are missing data coded?

  - 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)
```

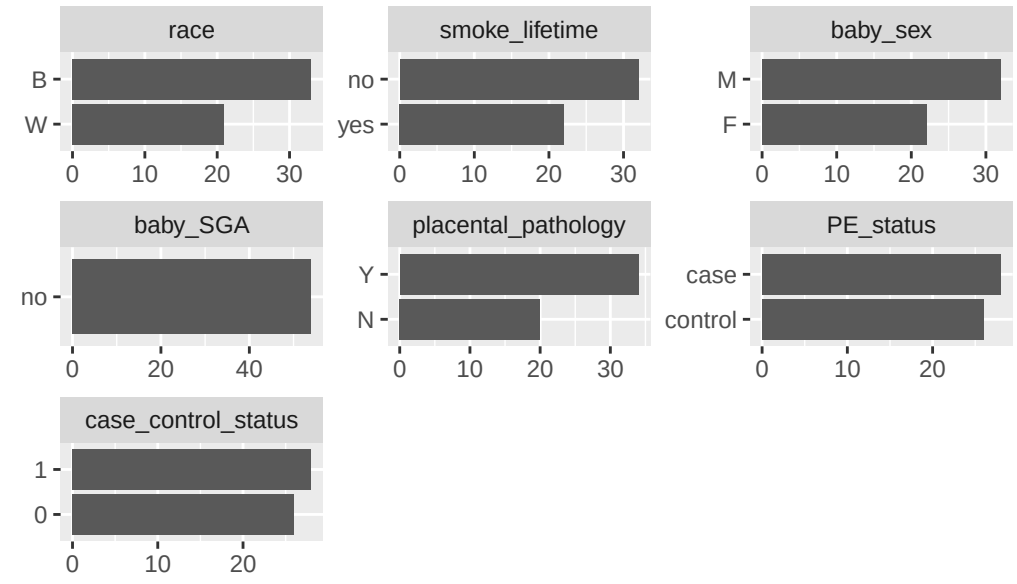



## 22.8 Distribution

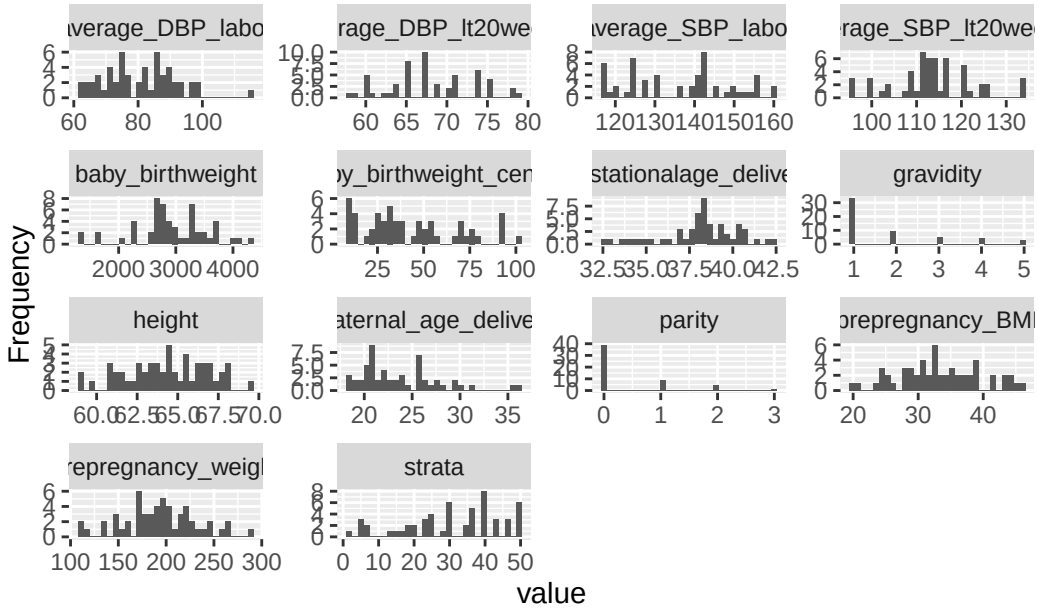
- What is the distribution of each of our phenotypes?
  - Are data skewed?
  - What is the range of values?
  - Is the range of values realistic?

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

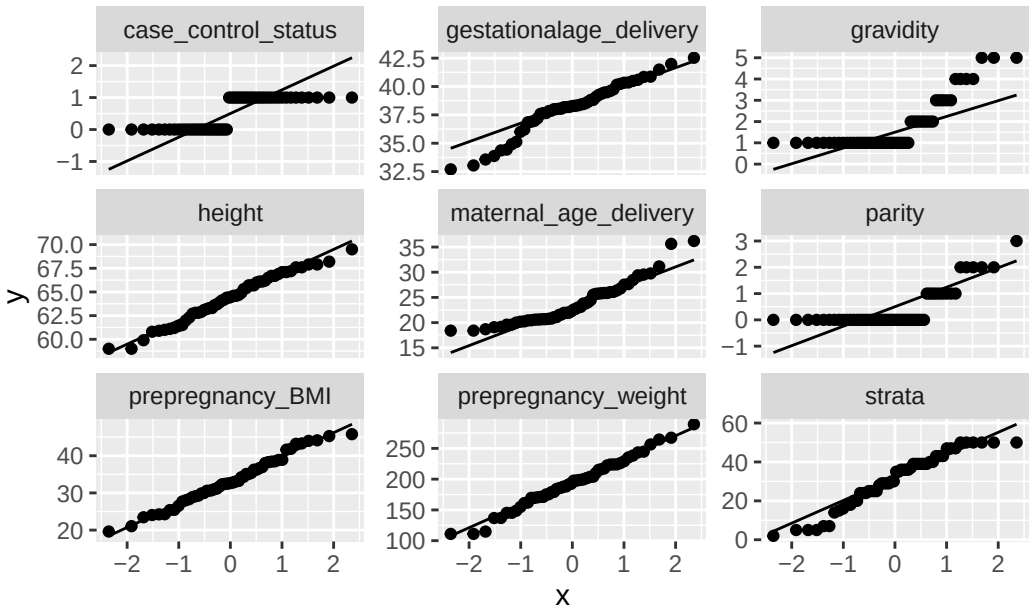
```
1 columns ignored with more than 50 categories.
subject_id: 54 categories
```



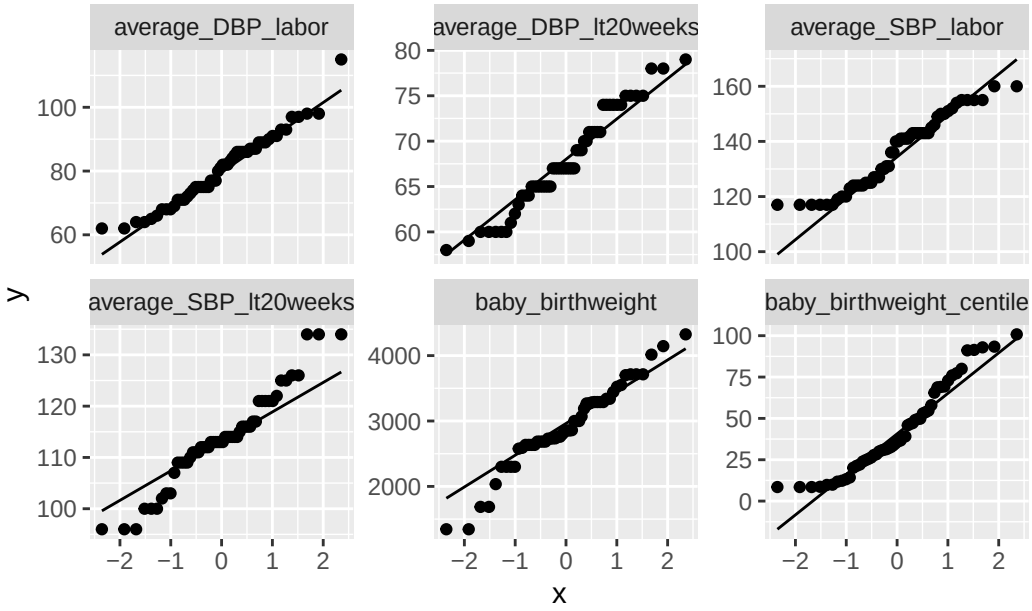

```
plot_histogram(ds.subj)
```




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




Page 1




Page 2

## 22.9 Variation

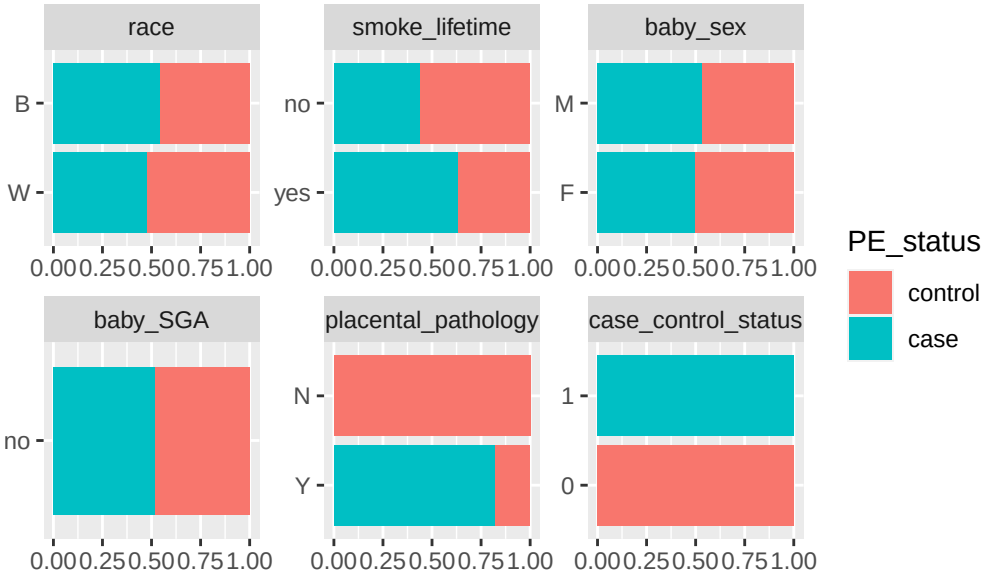
- How do our data vary and co-vary?

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

### 22.9.1 Bar plots

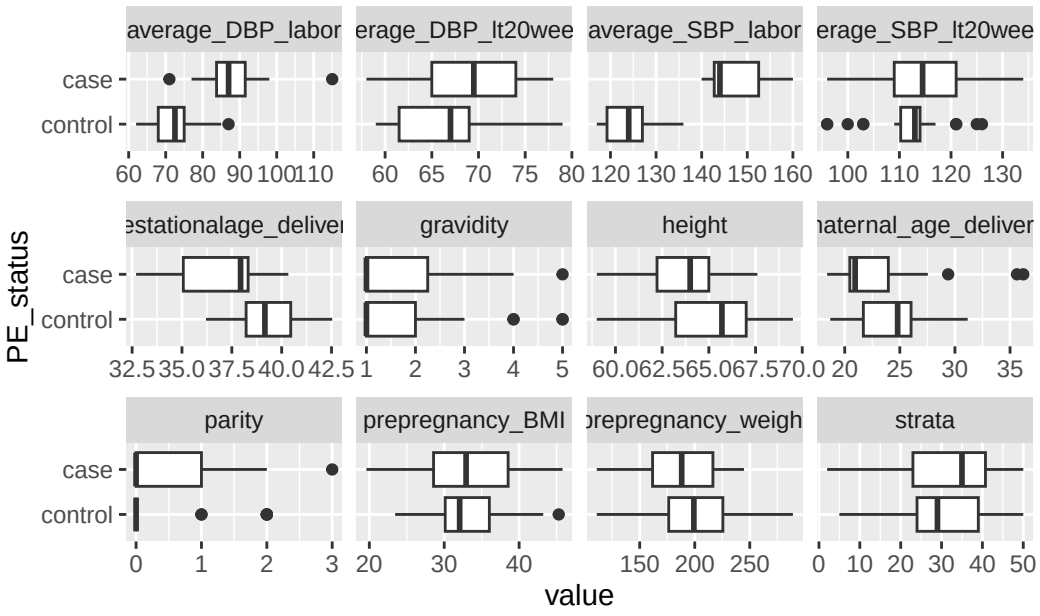
```
plot_bar(ds.subj, by = "PE_status")
```

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

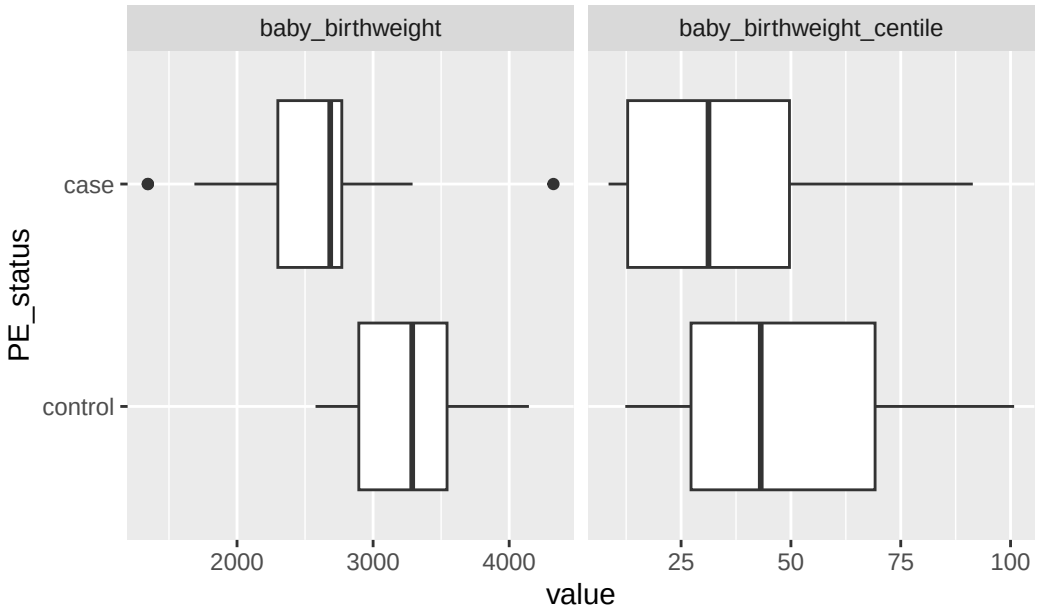



### 22.9.2 Box plots

```
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

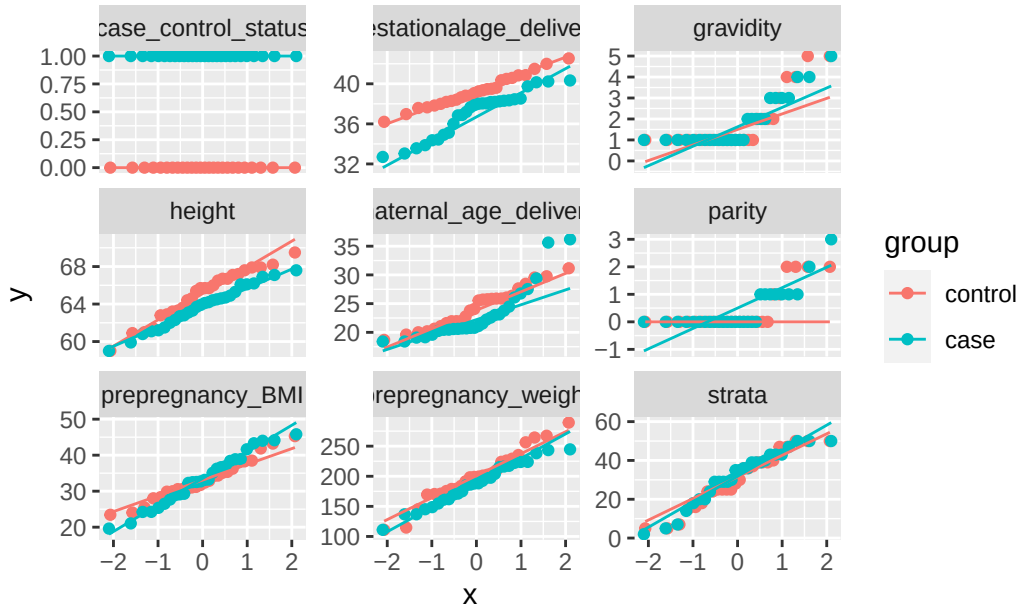
Page 1


baby_birthweight
baby_birthweight_centile
case
control
PE_status
value

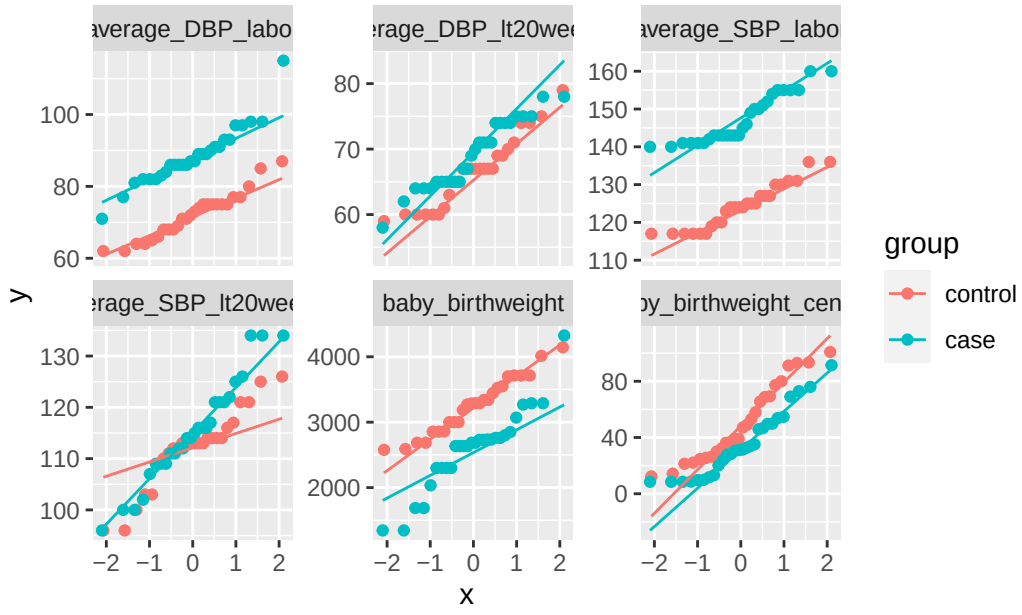
Page 2

### 22.9.3 QQ plots

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




Page 1



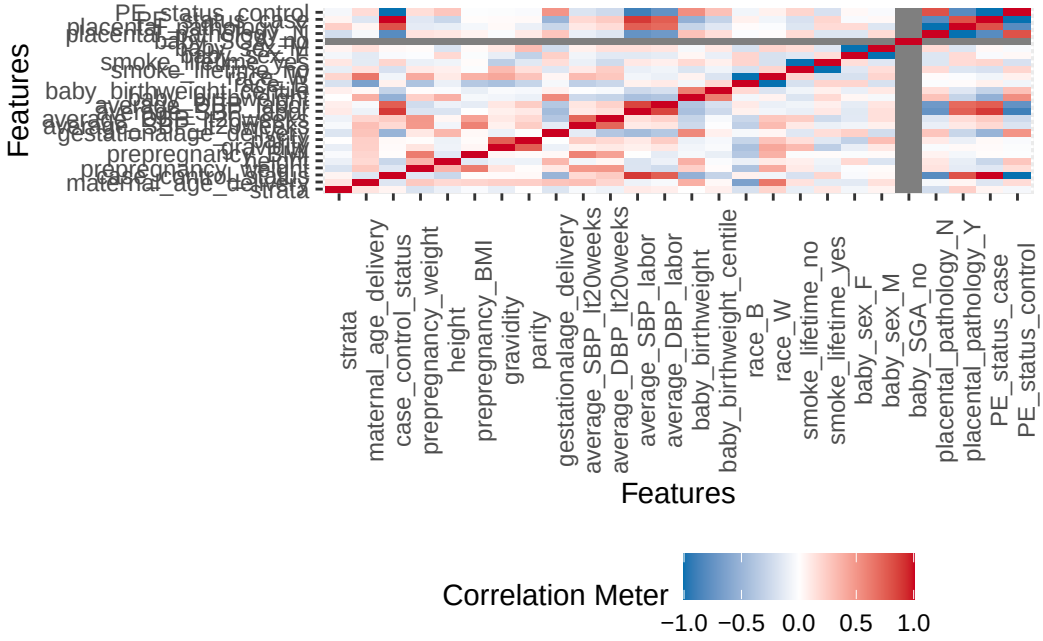

Page 2

### 22.9.4 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
```

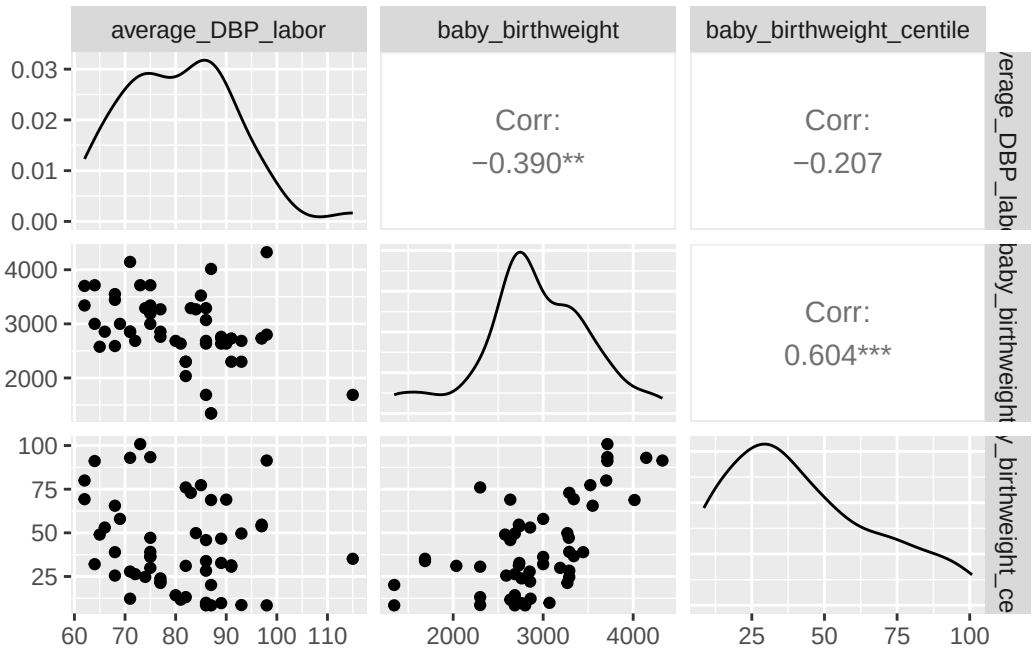



### 22.9.5 `ggpairs` from the `GGally` R package.

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

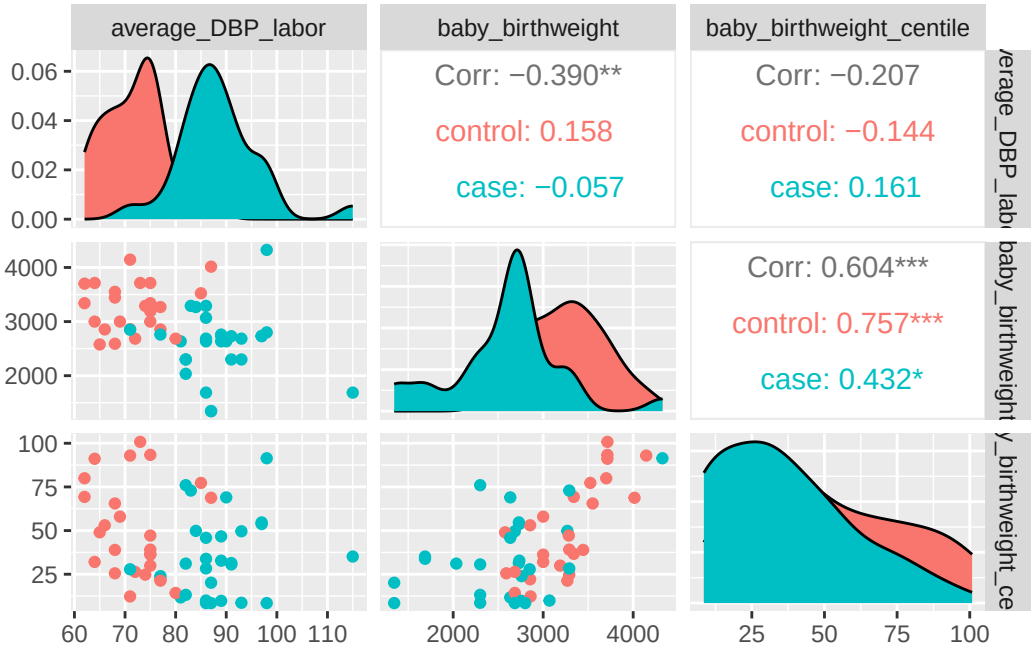
```
# Pull out numeric columns
ds1 <- ds.subj[, sapply(ds.subj, is.numeric)]
```

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

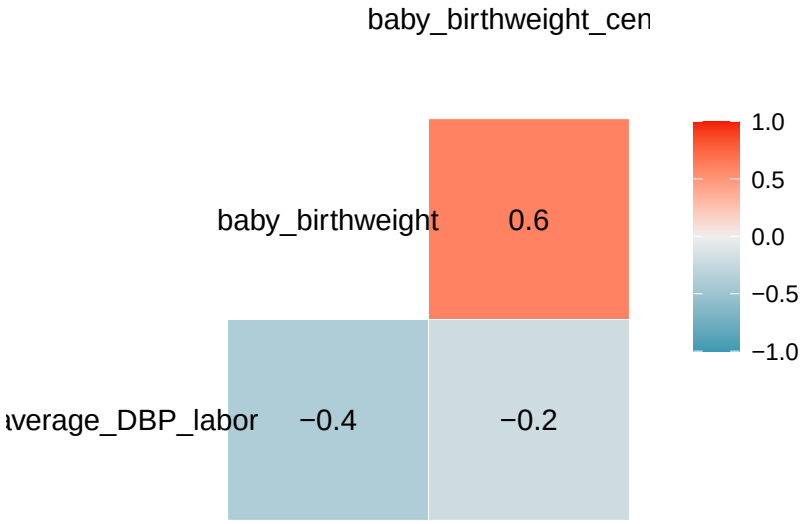



ggpairs - color by ggplot2 aes

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



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




## 22.10 DataExplorer

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

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

# 23 Basic Shell Commands

## 23.1 Acknowledgment and License

This chapter is a derivative of the Basic Shell Commands cheat sheet from the DEPRECATED-boot-camps/shell/shell_cheatsheet.md file created by Software Carpentry and is used under the Creative Commons - Attribution license CC 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.

---

## 23.2 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` | changes the current directory to the directory `dirname` |
| `./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) |
| `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. |

## 23.3 Creating Things:

### 23.3.1 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) |

### 23.3.2 How to delete files and directories...

#### 23.3.2.1 *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. |

### 23.3.3 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 |

## 23.4 Pipes and Filters

### 23.4.1 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.

#### 23.4.1.1 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.

### 23.4.2 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`) |

#### 23.4.2.1 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) |

##### 23.4.2.1.1 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.

## 23.5 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.

### 23.5.1 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`.

### 23.5.2 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

#### 23.5.2.1 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.

## 23.6 Finding Things

### 23.6.1 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 |

#### 23.6.1.1 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 |

### 23.6.2 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 |

#### 23.6.2.1 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 |

# 24 Summary

In summary, this book is a work in progress.

# 25 WebR - R in the web browser

This is a WebR-enabled code cell in a Quarto HTML document. As the WebR documentation states: "WebR makes it possible to run R code in the browser without the need for an R server to execute the code: the R interpreter runs directly on the user's machine."

> **Warning**
>
> Use a Chrome or Firefox browser - WebR does not work with the Safari browser yet. If the following WebR chunk is working properly, you should see an editor window below the `Run code` tab displaying two lines of R code.

```
# Edit/add code here
fit = lm(mpg ~ am, data = mtcars)
summary(fit)
library(ggplot2)
ggplot(mtcars, aes(x=am,y=mpg)) +
   geom_point() +
   geom_smooth(method="lm")
```

Link: WebR.

---

# 26 Technical Details

## 26.1 Quarto

This book was build using Quarto.

### 26.1.1 Callout blocks

To hide a solution that then can be clicked to view, we use a `.callout-tip collapse="true"` callout block.

Here are some examples from the Quarto documentation:

> **Note**
>
> Note that there are five types of callouts, including: `note`, `tip`, `warning`, `caution`, and `important`.

> **Warning**
>
> Callouts provide a simple way to attract attention, for example, to this warning.

> **This is Important**
>
> Danger, callouts will really improve your writing.

> **Tip With Title**
>
> This is an example of a callout with a title.

> **Expand To Learn About Collapse**
>
> This is an example of a 'collapsed' caution callout that can be expanded by the user. You can use `collapse="true"` to collapse it by default or `collapse="false"` to make a collapsible callout that is expanded by default.

### 26.1.2 Adding a chapter

To add a new chapter to the book, make a Quarto file containing the chapter text and code. It should have only one top-level header at the beginning which will be the title of the chapter.

Then add it to the list of chapters in the `_quarto.yml` file.

## 26.2 Previewing the book

Type `quarto preview` in the Terminal window.

## 26.3 Deploying the book to GitHub Pages

Type `quarto publish` in the Terminal window.

## 26.4 Deploying the book to Netlify

Type `quarto publish netlify` in the Terminal window.

## 26.5 WebR: R in the browser

This Quarto book uses this WebR Quarto extension

https://github.com/coatless/quarto-webr

WebR makes installs a version of R that runs within the browser, and the Quarto extension makes it interactively available in `webr-r` chunks.

```
# Edit/add/try out R code here
```

To get this to work, the `_quarto.yml` had to be modified.

We added a 'resources' directive to copy over the java script files, which places them next to the 'index.html' file during deployment of the book:

```
project:
  type: book
  resources:
    - "webr-serviceworker.js"
    - "webr-worker.js"
```

We also enabled the `webr` filter:

```
filters:
    - webr
```

## 26.6 embedpdf Quarto extension

This book uses the `embedpdf` Quarto extension from https://github.com/jmgirard/embedpdf, which was installed via this command:

```
quarto add jmgirard/embedpdf
```

To embed a PDF, use code like this:

```
{{< pdf dummy.pdf width=100% height=800 >}}
```

However, the PDF embedding done this way did not work in Chrome.

Example:

So instead we used an iframe, which works on Chrome, Firefox, and Safari:

```
 <iframe width="100%" height="800" src="pdfs/GitHubIntro.pdf">
```

Note that for iframe embedding of Panopto video from the University of Pittsburgh, one needs to use a credentialless iframe.

# References