

# **Problems and Solutions in Applied Statistics (2nd ed)**

Ken Butler

2023-05-23

# Table of contents

<b>Introduction</b>	<b>3</b>
Packages used somewhere in this book . . . . .	4
<b>1 Getting used to R and R Studio</b>	<b>6</b>
1.1 Using R Studio online . . . . .	6
1.2 Using R Studio on your own computer . . . . .	12
1.3 Getting started . . . . .	12
1.4 Reading data from a file . . . . .	25
1.5 Reading files different ways . . . . .	29
<b>2 Reading in data</b>	<b>39</b>
2.1 Orange juice . . . . .	39
2.2 Making soap . . . . .	39
2.3 Handling shipments . . . . .	40
2.4 Orange juice . . . . .	40
2.5 Making soap . . . . .	43
2.6 Handling shipments . . . . .	44
<b>3 Data exploration</b>	<b>47</b>
3.1 North Carolina births . . . . .	47
3.2 More about the NC births . . . . .	47
3.3 Nenana, Alaska . . . . .	48
3.4 Computerized accounting . . . . .	48
3.5 Test scores in two classes . . . . .	49
3.6 Unprecedented rainfall . . . . .	50
3.7 Learning algebra . . . . .	50
3.8 North Carolina births . . . . .	51
3.9 More about the NC births . . . . .	63
3.10 Nenana, Alaska . . . . .	70
3.11 Computerized accounting . . . . .	78
3.12 Test scores in two classes . . . . .	87
3.13 Unprecedented rainfall . . . . .	100
3.14 Learning algebra . . . . .	109

# Introduction

[This book](#) contains a collection of problems, and my solutions to them, in applied statistics with R. These come from my courses STAC32, STAC33, and STAD29 at the University of Toronto Scarborough.

You will occasionally see question parts beginning with a \*; this means that other question parts refer back to this one. (One of my favourite question strategies is to ask how two different approaches lead to the same answer, or more generally to demonstrate that there are different ways to see the same thing.)

Thanks to Dann Sioson for spotting some errors and making some useful suggestions.

If *you* see anything, [file an issue](#) on the Github page for now. Likely problems include:

- some LaTeX construction that I didn't catch (eg. block quotes)
- disappeared footnotes (that will show up as an apparently missing sentence in the text)
- references to “in class” or a lecture or a course by course number, which need to be eliminated (in favour of wording like “a previous course”)
- references to other questions or question parts that are *wrong* (likely caused by *not* being “labels” or “refs” in the original LaTeX)
- my contorted English that is difficult to understand.

As I read through looking for problems like these, I realize that there ought to be a textbook that reflects my way of doing things. There isn't one (yet), though there are lecture notes. Current versions of these are at:

- [the STAC32 website](#)
- [the STAC33 website](#)
- [the STAD29 website](#)

A little background:

STAC32 is an introduction to R as applied to statistical methods that have (mostly) been learned in previous courses. This course is designed for students who have a second non-mathematical applied statistics course such as [this](#). The idea is that students have already seen a little of regression and analysis of variance (and the things that precede them), and need mainly an introduction of how to run them in R.

STAC33 is an introduction to R, and applied statistics in general, for students who have a background in mathematical statistics. The way our courses are structured, these students have a strong mathematical background, but not very much experience in applications, which this course is designed to provide. The material covered is similar to STAC32, with a planned addition of some ideas in bootstrap and practical Bayesian statistics. There are some questions on these here.

STAD29 is an overview of a number of advanced statistical methods. I start from regression and proceed to some regression-like methods (logistic regression, survival analysis, log-linear frequency table analysis), then I go a little further with analysis of variance and proceed with MANOVA and repeated measures. I finish with a look at classical multivariate methods such as discriminant analysis, cluster analysis, principal components and factor analysis. I cover a number of methods in no great depth; my aim is to convey an understanding of what these methods are for, how to run them and how to interpret the results. Statistics majors and specialists cannot take this course for credit (they have separate courses covering this material with the proper mathematical background). D29 is intended for students in other disciplines who find themselves wanting to learn more statistics; we have an [Applied Statistics Minor program](#) for which C32 and D29 are two of the last courses.

## Packages used somewhere in this book

The bottom lines are below used with the `conflicted` package: if a function by the name shown is in two or more packages, prefer the one from the package shown.

```
library(tidyverse)
library(smmr)
library(MASS)
library(nnet)
library(survival)
library(survminer)
library(car)
library(lme4)
library(ggbiplot)
library(ggrepel)
library(broom)
library(rpart)
library(bootstrap)
library(cmdstanr)
library(posterior)
library(bayesplot)
library(tmaptools)
```

```
library(leaflet)
library(conflicted)
conflict_prefer("summarize", "dplyr")
conflict_prefer("select", "dplyr")
conflict_prefer("filter", "dplyr")
conflict_prefer("mutate", "dplyr")
conflict_prefer("count", "dplyr")
conflict_prefer("arrange", "dplyr")
conflict_prefer("rename", "dplyr")
conflict_prefer("id", "dplyr")
```

All of these packages are on CRAN, and may be installed via the usual `install.packages`, with the exceptions of:

- `smmr` on Github: install with

```
devtools::install_github("nxskok/smmr")
```

- `ggbiplot` on Github: install with

```
devtools::install_github("vqv/ggbiplot")
```

- `cmdstanr`, `posterior`, and `bayesplot`: install with

```
install.packages("cmdstanr",
                 repos = c("https://mc-stan.org/r-packages/",
                           getOption("repos")))
install.packages("posterior",
                 repos = c("https://mc-stan.org/r-packages/",
                           getOption("repos")))
install.packages("bayesplot",
                 repos = c("https://mc-stan.org/r-packages/",
                           getOption("repos")))
```

# 1 Getting used to R and R Studio

Don't forget `library(tidyverse)` first if you need it (you probably will).

## 1.1 Using R Studio online

- (a) Point your web browser at <http://r.datatools.utoronto.ca>. Click on the button to the left of “R Studio” (it will show blue), click the orange Log in to Start, and log in using your UTorID and password.

Solution

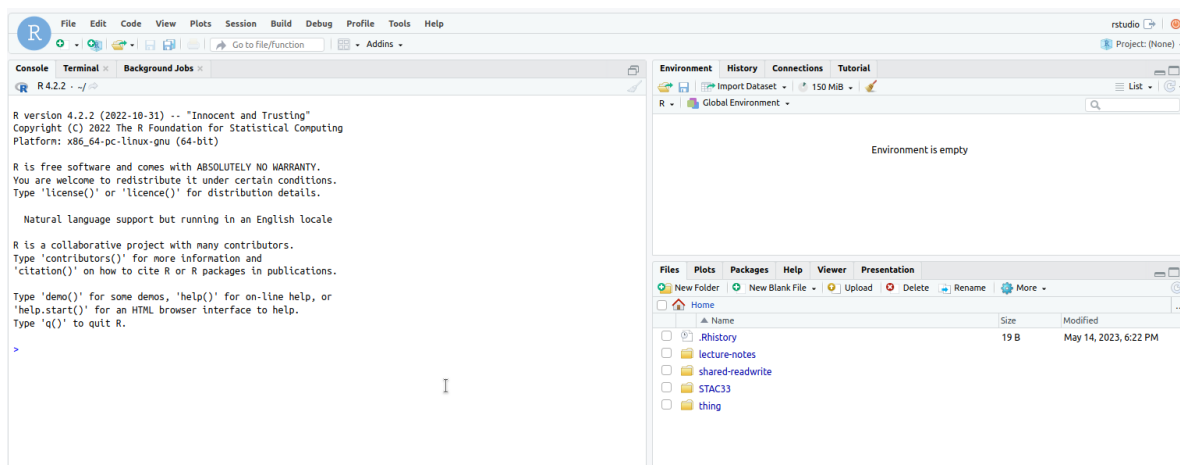
This is about what you should see first, before you click the orange thing:



After logging in, open: ☐ Jupyter Notebook ☒ RStudio ☐ JupyterLab

Log in to start

You will see a progress bar as things start up, and then you should see something like this:



This is R Studio, ready to go.

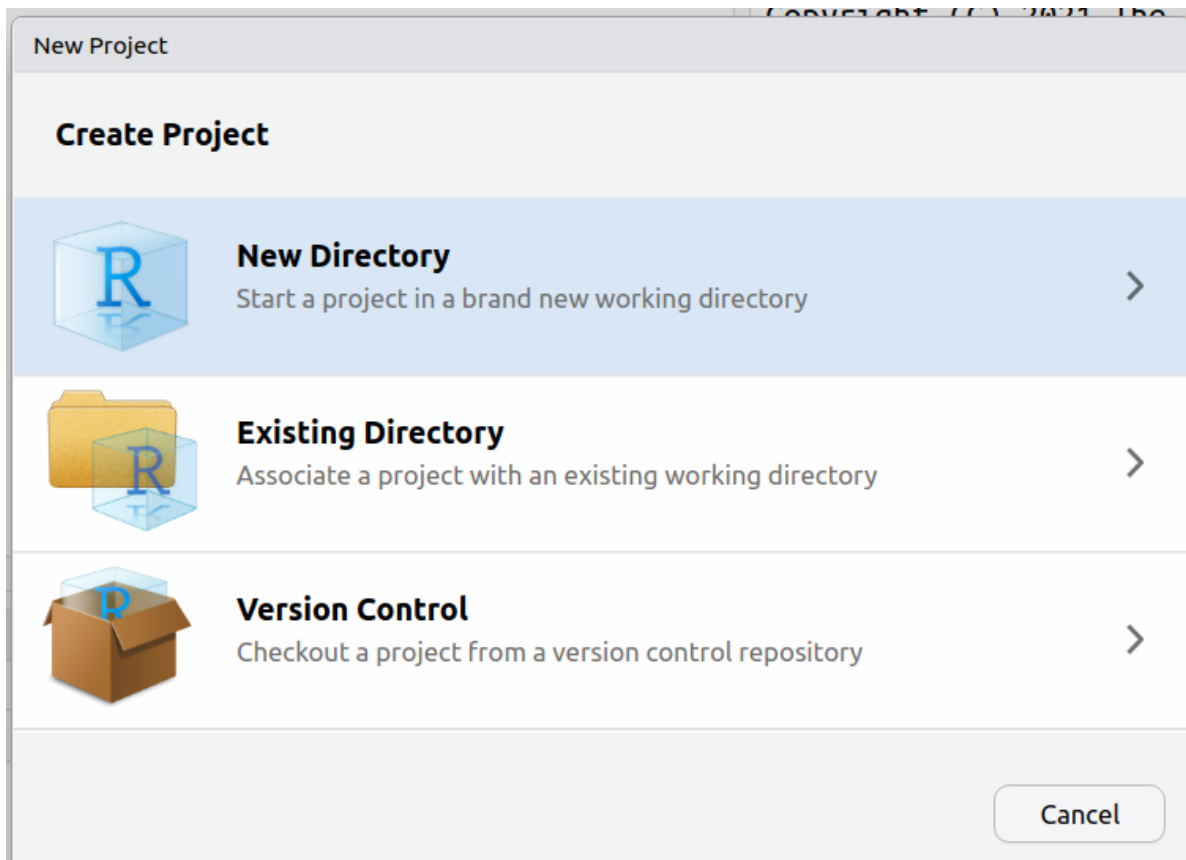
If you are already logged in to something else on the same browser that uses your UTorID and password, you may come straight here without needing to log in again.



(b) Take a look around, and create a new Project. Give the new project any name you like.

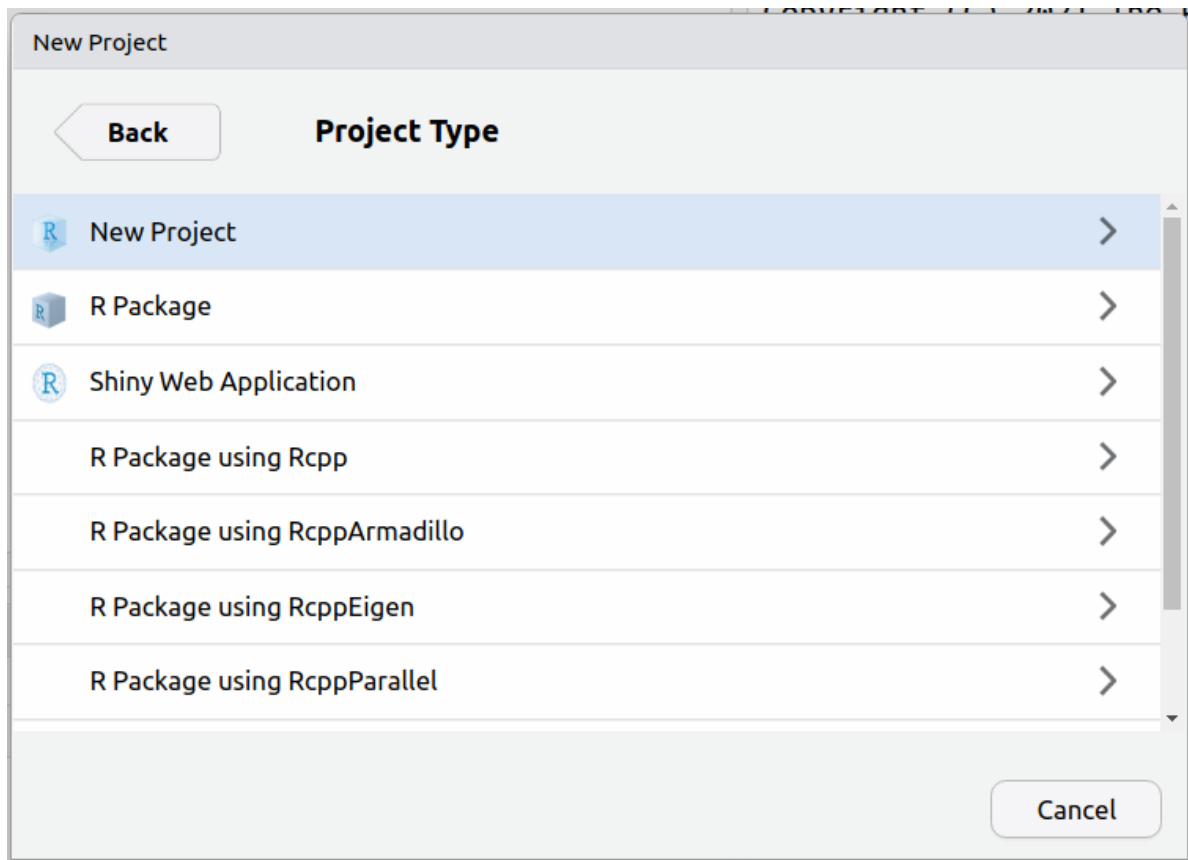
Solution

Select File and New Project to get this:

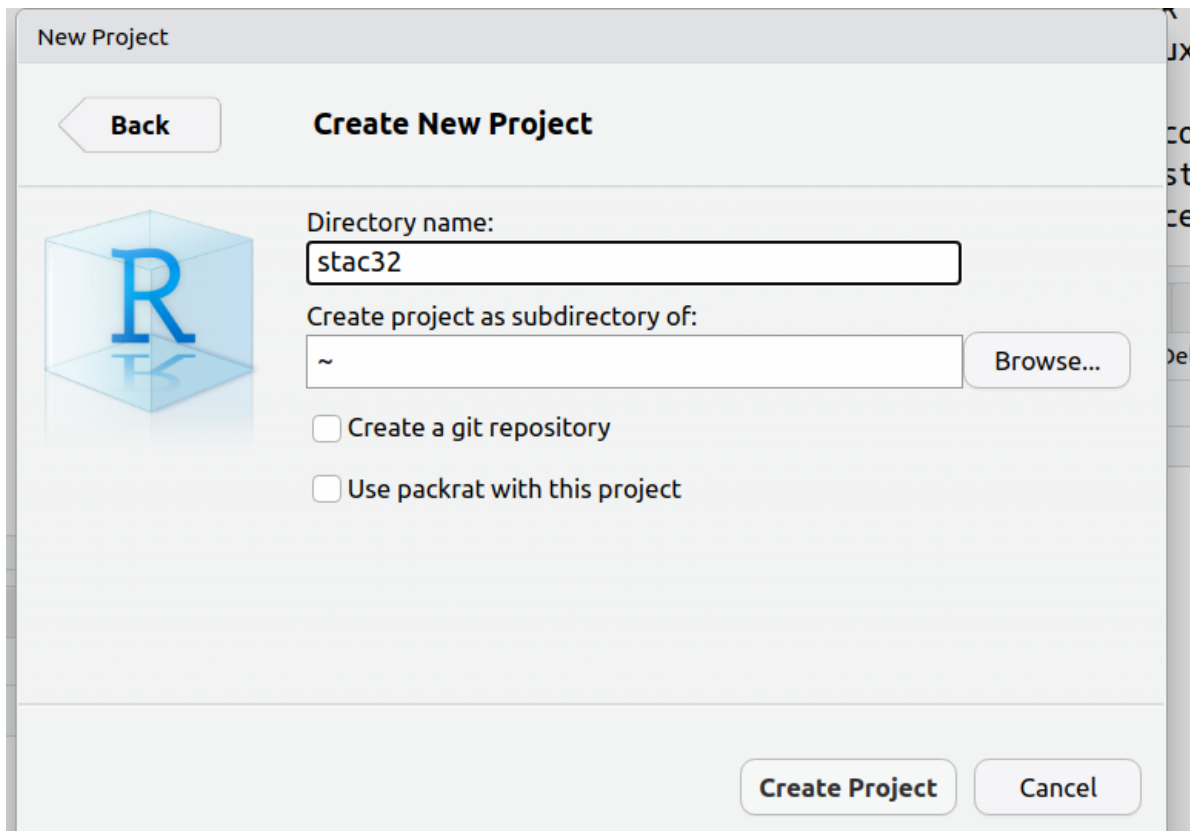


Click on New Directory (highlighted blue on mine). This will create a new folder to put your new project in, which is usually what you want to do. The idea is that a project is a container for a larger collection of work, such as all your assignments in this course. That brings you to this:

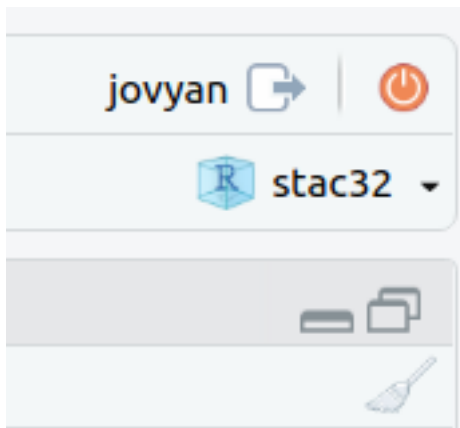




where you click on New Project (highlighted on mine), and:



Give your project a name, as I did. Then click Create Project. At this point, R Studio will be restarted in your new project. You can tell which project you are in by looking top right, and you'll see the name of your project next to the R symbol:



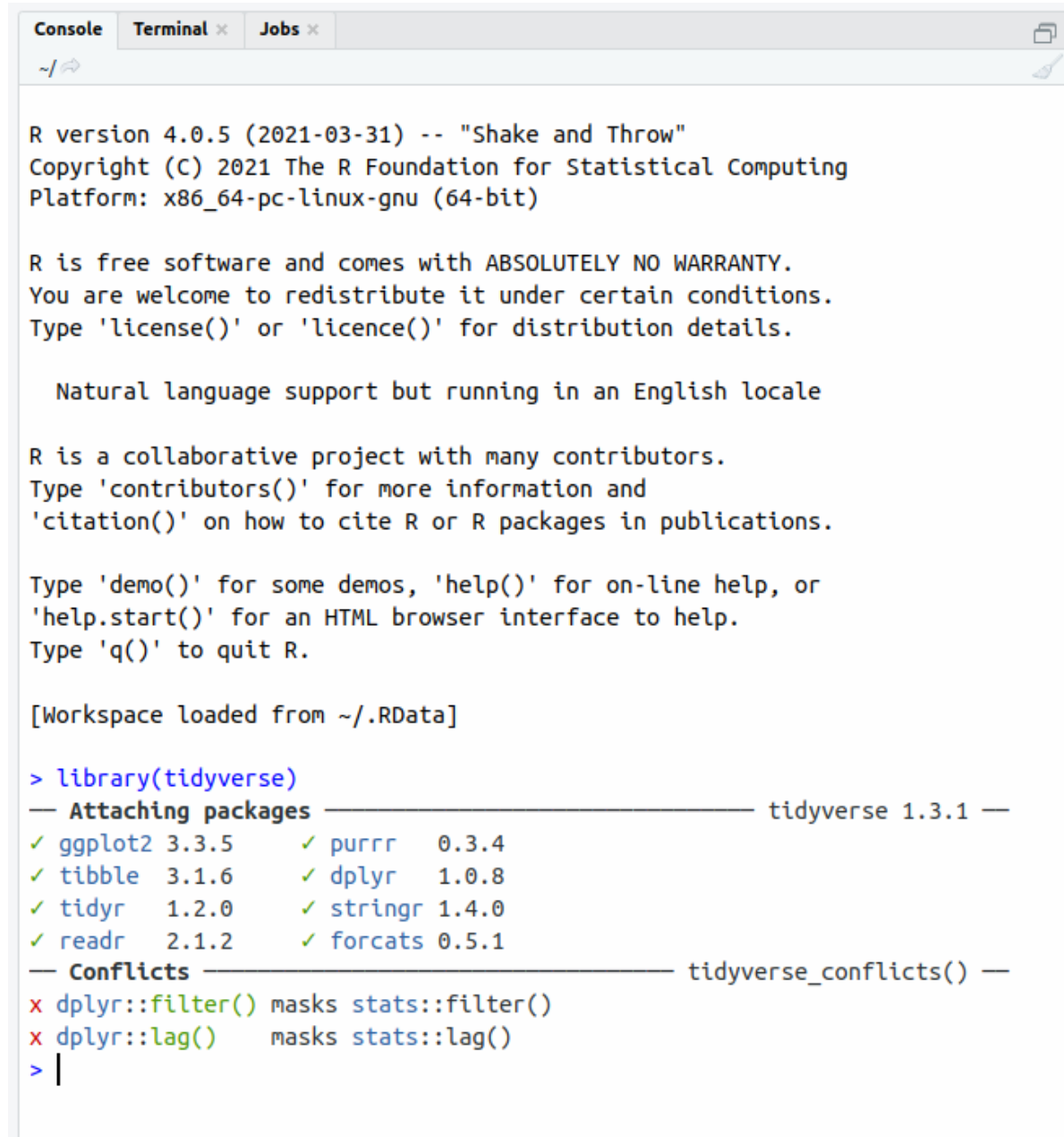
■

(c) One last piece of testing: find the Console window (which is probably on the left). Click

next to the blue `>`, and type `library(tidyverse)`. Press Enter.

Solution

It may think a bit, and then you'll see something like this:



```
Console Terminal x Jobs x
~/
R version 4.0.5 (2021-03-31) -- "Shake and Throw"
Copyright (C) 2021 The R Foundation for Statistical Computing
Platform: x86_64-pc-linux-gnu (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

Natural language support but running in an English locale

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

[Workspace loaded from ~/.RData]

> library(tidyverse)
— Attaching packages — tidyverse 1.3.1 —
✓ ggplot2 3.3.5    ✓ purrr  0.3.4
✓ tibble  3.1.6    ✓ dplyr  1.0.8
✓ tidyr   1.2.0    ✓ stringr 1.4.0
✓ readr   2.1.2    ✓ forcats 0.5.1
— Conflicts — tidyverse_conflicts() —
x dplyr::filter() masks stats::filter()
x dplyr::lag()    masks stats::lag()
> |
```

Aside: I used to use a cloud R Studio called `rstudio.cloud`. If you see or hear any references to that, it means the same thing as R Studio on `r.datatools` or `jupyter`. (You can still use

`rstudio.cloud` if you want; it used to be completely free, but now the free tier won't last you very long; the `utoronto.calink` is free as long as you are at U of T.) I'm trying to get rid of references to R Studio Cloud as I see them, but I am bound to miss some, and in the lecture videos they are rather hard to find.

Now we can get down to some actual work.



## 1.2 Using R Studio on your own computer

This is not required now, but you may wish to do this now or later so that you are not fighting for resources on the `r.datatools` server at busy times (eg. when an assignment is due).

Follow the instructions [here](#) to install R Studio on your computer, then start R Studio (which itself starts R).

Once you have this working, you can use it for any of the following questions, in almost exactly the same way as the online R (I will explain any differences).

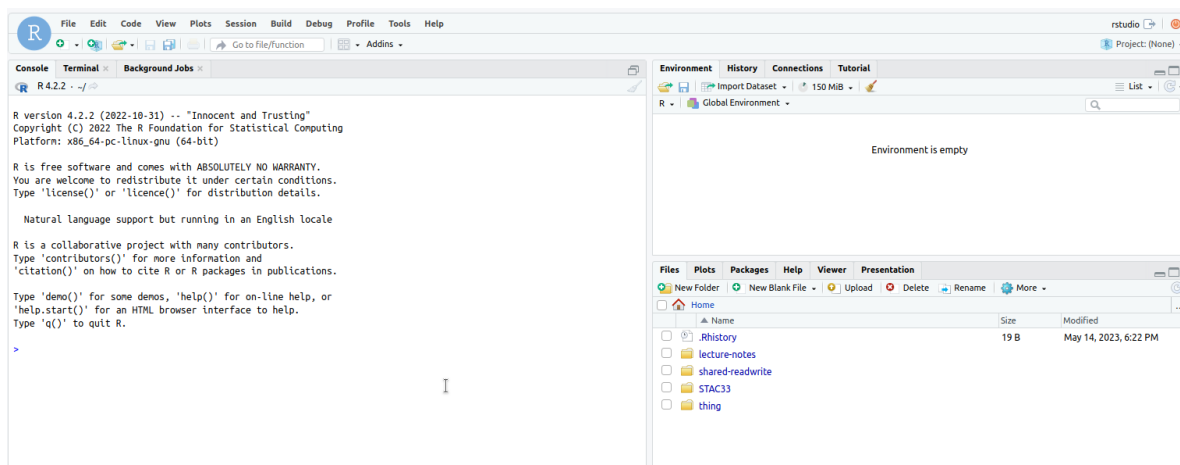
## 1.3 Getting started

This question is to get you started using R.

- (a) Start R Studio on `r.datatools` (or on your computer), in some project. (If you started up a new project in the previous question and are still logged in, use that; if not, create a new project with File, New Project, and New Directory. Then select New Project and give it a name. Click Create Project. This will give you an empty workspace to start from.)

Solution

You ought to see something like this:



There should be one thing on the left half, and at the top right it'll say "Environment is empty".

Extra: if you want to tweak things, select Tools (at the top of the screen) and from it Global Options, then click Appearance. You can make the text bigger or smaller via Editor Font Size, and choose a different colour scheme by picking one of the Editor Themes (which previews on the right). My favourite is Tomorrow Night Blue. Click Apply or OK when you have found something you like. (I spend a lot of time in R Studio, and I like having a dark background to be easier on my eyes.)

■

- (b) We're going to do some stuff in R here, just to get used to it. First, make a Quarto document by selecting File, New File and Quarto Document.

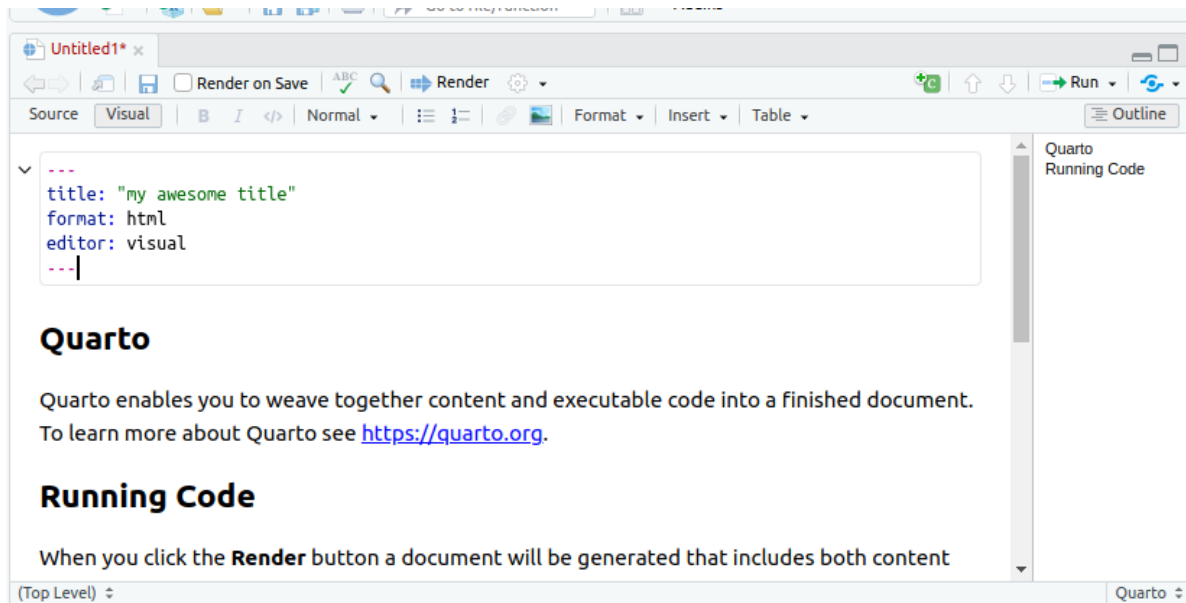
Solution

In the first box that pops up, you'll be invited to give your document a title. Make something up for now.

The first time, you might be invited to "install some packages" to make the document thing work.<sup>1</sup> Let it do that by clicking Yes. After that, you'll have this:

---

<sup>1</sup>Especially if you are on your own computer.



A couple of technical notes:

- this should be in the top left pane of your R Studio now, with the Console below it.
- At the top of the file, between the two lines with three hyphens (minus signs, whatever), is some information about the document, known in the jargon as a YAML block, any of which you can change:
  - the title is whatever title you gave your document
  - the **format** is what the *output* is going to be (in this case, HTML like a webpage, which is mostly what we'll be using)
  - there is a visual editor that looks like Notion or a bit like a Google doc (the default), and also a Source editor which gives you more control, and shows that underlying the document is a thing called R Markdown (which is a code for writing documents).
- My *document* is called “My awesome title”, but the *file* in which the document lives is still untitled because I haven’t saved it yet. See right at the top.

■

- (c) You can delete the template code below the YAML block now (that is, everything from the title “Quarto” to the end). Somewhere in the space opened up below the YAML block (it might say “Heading 2”, greyed out), type a /. This, like Notion, gives you a list of things to choose from to insert there. Pressing Enter will insert a “code chunk”, sometimes known as a “code cell”. We are going to use this in a moment.

Solution

Something like this:

```
---  
title: "my awesome title"  
format: html  
editor: visual  
---
```

```
{r}
```

The `{r}` at the top of the code chunk means that the code that will go in there will be R code (you can also have a Python code chunk, among others).



- (d) On the line below the `{r}`, type these two lines of code into the chunk in the Quarto document:

```
library(tidyverse)  
mtcars
```

Solution

What this will do: get hold of a built-in data set with information about some different models of car, and display it.

```
---  
title: "my awesome title"  
format: html  
editor: visual  
---
```

```
✓ {r}  
mtcars
```

In approximately five seconds, you'll be demonstrating that for yourself.



- (e) Run this command. To do that, look at the top right of your code chunk block (shaded in a slightly different colour). You should see a down arrow and a green “play button”. Click the play button. This will run the code, and show the output below the code chunk.

Solution

Here’s what I get (yours should be the same):

{r}
mtcars

	mpg <dbl>	cyl <dbl>	disp <dbl>	hp <dbl>	drat <dbl>	wt <dbl>	qsec <dbl>
Mazda...	21.0	6	160.0	110	3.90	2.620	16.46
Mazda...	21.0	6	160.0	110	3.90	2.875	17.02
Datsun...	22.8	4	108.0	93	3.85	2.320	18.61
Hornet...	21.4	6	258.0	110	3.08	3.215	19.44
Hornet...	18.7	8	360.0	175	3.15	3.440	17.02
Valiant	18.1	6	225.0	105	2.76	3.460	20.22
Duster...	14.3	8	360.0	245	3.21	3.570	15.84
Merc 2...	24.4	4	146.7	62	3.69	3.190	20.00
Merc 2...	22.8	4	140.8	95	3.92	3.150	22.90
Merc 2...	19.2	6	167.6	123	3.92	3.440	18.30

1-10 of 32 rows | 1-8 of 11 columns

Previous
1
2
3
4
Next

This is a rectangular array of rows and columns, with individuals (here, cars) in rows and variables in columns, known as a “dataframe”. When you display a dataframe in an Quarto document, you see 10 rows and as many columns as will fit on the screen. At the bottom, it says how many rows and columns there are altogether (here 32 rows and 11 columns), and which ones are being displayed.

You can see more rows by clicking on Next, and if there are more columns, you’ll see a little arrow next to the rightmost column (as here next to `am`) that you can click on to see more columns. Try it and see. Or if you want to go to a particular collection of rows, click one of the numbers between Previous and Next: 1 is rows 1–10, 2 is rows 11–20, and so on.

The column on the left without a header (containing the names of the cars) is called “row names”. These have a funny kind of status, kind of a column and kind of not a column; usually, if we need to use the names, we have to put them in a column first.

In future solutions, rather than showing you a screenshot, expect me to show you something like this:



```
library(tidyverse)
mtcars
```

```
# A tibble: 32 x 12
  car      mpg  cyl  disp    hp  drat    wt  qsec    vs    am  gear  carb
  <chr>    <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
1 Mazda RX4      21     6  160   110  3.9   2.62  16.5     0     1     4     4
2 Mazda RX4 ~    21     6  160   110  3.9   2.88  17.0     0     1     4     4
3 Datsun 710    22.8     4  108    93  3.85  2.32  18.6     1     1     4     1
4 Hornet 4 D~   21.4     6  258   110  3.08  3.22  19.4     1     0     3     1
5 Hornet Spo~   18.7     8  360   175  3.15  3.44  17.0     0     0     3     2
6 Valiant       18.1     6  225   105  2.76  3.46  20.2     1     0     3     1
7 Duster 360    14.3     8  360   245  3.21  3.57  15.8     0     0     3     4
8 Merc 240D     24.4     4  147.    62  3.69  3.19   20      1     0     4     2
9 Merc 230      22.8     4  141.    95  3.92  3.15  22.9     1     0     4     2
10 Merc 280     19.2     6  168.   123  3.92  3.44  18.3     1     0     4     4
# i 22 more rows
```

The top bit is the code, the bottom bit the output. In this kind of display, you only see the first ten rows (by default).<sup>2</sup>

If you don't see the "play button", make sure that what you have really is a code chunk. (I often accidentally delete one of the special characters above or below the code chunk). If you can't figure it out, delete this code chunk and make a new one. Sometimes R Studio gets confused.

On the code chunk, the other symbols are the settings for this chunk (you have the choice to display or not display the code or the output or to not actually run the code). The second one, the down arrow, runs all the chunks prior to this one (but not this one).

Your output has its own little buttons (as seen on the screenshot). The first one pops the output out into its own window; the second one shows or hides the output, and the third one deletes the output (so that you have to run the chunk again to get it back). Experiment. You can't do much damage here.



- (f) Something a little more interesting: **summary** obtains a summary of whatever you feed it (the five-number summary plus the mean for numerical variables). Obtain this for our data frame. To do this, create a new code chunk below the previous one, type **summary(mtcars)** into the code chunk, and run it.

<sup>2</sup>This document was actually produced by literally running this code, a process known as "rendering", which we will learn about shortly.

Solution

This is what you should see:

```
8
9- '''{r}
0 summary(mtcars)
1 '''
```

mpg	cyl	disp	hp	drat	wt
Min. :10.40	Min. :4.000	Min. : 71.1	Min. : 52.0	Min. :2.760	Min. :1.513
1st Qu.:15.43	1st Qu.:4.000	1st Qu.:120.8	1st Qu.: 96.5	1st Qu.:3.080	1st Qu.:2.581
Median :19.20	Median :6.000	Median :196.3	Median :123.0	Median :3.695	Median :3.325
Mean :20.09	Mean :6.188	Mean :230.7	Mean :146.7	Mean :3.597	Mean :3.217
3rd Qu.:22.80	3rd Qu.:8.000	3rd Qu.:326.0	3rd Qu.:180.0	3rd Qu.:3.920	3rd Qu.:3.610
Max. :33.90	Max. :8.000	Max. :472.0	Max. :335.0	Max. :4.930	Max. :5.424

qsec	vs	am	gear	carb
Min. :14.50	Min. :0.0000	Min. :0.0000	Min. :3.000	Min. :1.000
1st Qu.:16.89	1st Qu.:0.0000	1st Qu.:0.0000	1st Qu.:3.000	1st Qu.:2.000
Median :17.71	Median :0.0000	Median :0.0000	Median :4.000	Median :2.000
Mean :17.85	Mean :0.4375	Mean :0.4062	Mean :3.688	Mean :2.812
3rd Qu.:18.90	3rd Qu.:1.0000	3rd Qu.:1.0000	3rd Qu.:4.000	3rd Qu.:4.000
Max. :22.90	Max. :1.0000	Max. :1.0000	Max. :5.000	Max. :8.000

or the other way:

```
summary(mtcars)
```

mpg	cyl	disp	hp
Min. :10.40	Min. :4.000	Min. : 71.1	Min. : 52.0
1st Qu.:15.43	1st Qu.:4.000	1st Qu.:120.8	1st Qu.: 96.5
Median :19.20	Median :6.000	Median :196.3	Median :123.0
Mean :20.09	Mean :6.188	Mean :230.7	Mean :146.7
3rd Qu.:22.80	3rd Qu.:8.000	3rd Qu.:326.0	3rd Qu.:180.0
Max. :33.90	Max. :8.000	Max. :472.0	Max. :335.0

drat	wt	qsec	vs
Min. :2.760	Min. :1.513	Min. :14.50	Min. :0.0000
1st Qu.:3.080	1st Qu.:2.581	1st Qu.:16.89	1st Qu.:0.0000
Median :3.695	Median :3.325	Median :17.71	Median :0.0000
Mean :3.597	Mean :3.217	Mean :17.85	Mean :0.4375
3rd Qu.:3.920	3rd Qu.:3.610	3rd Qu.:18.90	3rd Qu.:1.0000
Max. :4.930	Max. :5.424	Max. :22.90	Max. :1.0000

am	gear	carb
Min. :0.0000	Min. :3.000	Min. :1.000
1st Qu.:0.0000	1st Qu.:3.000	1st Qu.:2.000
Median :0.0000	Median :4.000	Median :2.000
Mean :0.4062	Mean :3.688	Mean :2.812
3rd Qu.:1.0000	3rd Qu.:4.000	3rd Qu.:4.000

Max. :1.0000    Max. :5.000    Max. :8.000

For the gas mileage column `mpg`, the mean is bigger than the median, and the largest value is unusually large compared with the others, suggesting a distribution that is skewed to the right.

There are 11 numeric (quantitative) variables, so we get the five-number summary plus mean for each one. Categorical variables, if we had any here, would be displayed a different way.

■

(g) Let's make a histogram of the gas mileage data. Type the code below into another new code chunk, and run it:

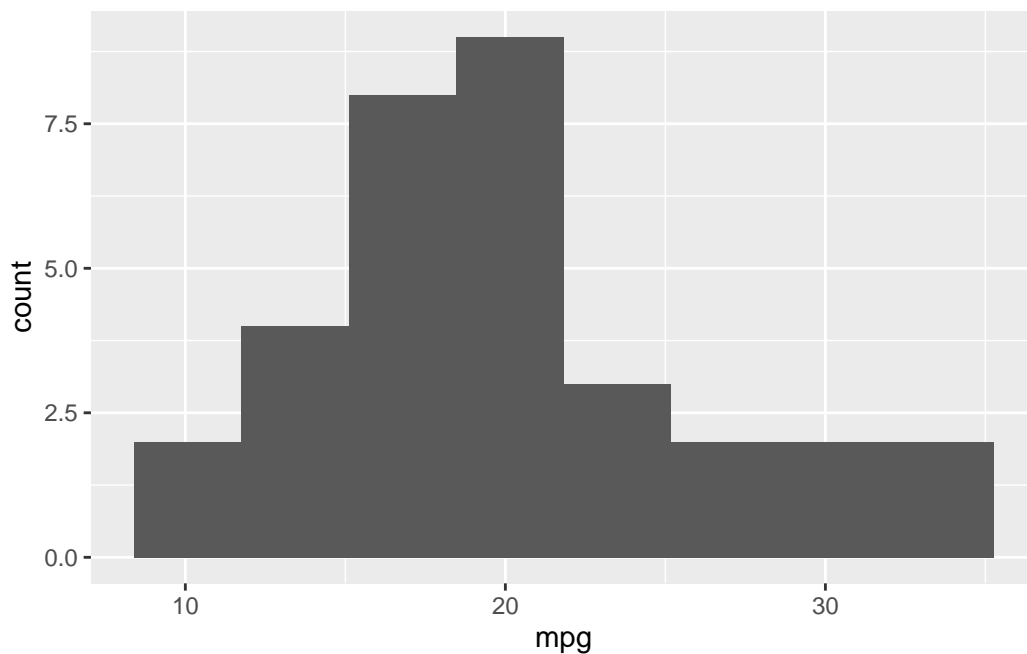
```
ggplot(mtcars, aes(x = mpg)) + geom_histogram(bins = 8)
```

The code looks a bit wordy, but we'll see what all those pieces do later in the course (like, maybe tomorrow).

Solution

This is what you should see:

```
ggplot(mtcars, aes(x = mpg)) + geom_histogram(bins = 8)
```



The long right tail supports our guess from before that the distribution is right-skewed.



- (h) Some aesthetics: Add some narrative text above and below your code chunks. Above the code chunk is where you say what you are going to do (and maybe why you are doing it), and below is where you say what you conclude from the output you just obtained. I find it looks better if you have a blank line above and below each code chunk.

Solution

This is what I wrote (screenshot), with none of the code run yet. My `library(tidyverse)` line seems to have disappeared, but yours should still be there:

```
---  
title: "my awesome title"  
format: html  
editor: visual  
---
```

Here is the built-in data set `mtcars`, with some information about 32 different cars:

```
{r}  
mtcars
```

We look at a numerical summary of each of the eleven (quantitative) variables:

```
{r}  
summary(mtcars)
```

For the `mpg` column, the mean is larger than the median, so we suspect that the distribution will be right-skewed. This can be confirmed by looking at a histogram:

```
{r}  
ggplot(mtcars, aes(x = mpg)) + geom_histogram(bins = 8)
```

The longer upper tail indicates a right skew.

---



- (i) Save your Quarto document (the usual way with File and Save). This saves it *on the jupyter servers* (and not on your computer). This means that when you come back to it later, even from another device, this document will still be available to you. (If you are running R Studio on your own computer, it is much simpler: the Quarto document is on that computer, in the folder associated with the project you created.)

Now click Render. This produces a pretty HTML version of your Quarto document. This will appear in a new tab of your web browser, which you might need to encourage to appear (if you have a pop-up blocker) by clicking a Try Again.

#### Solution

If there are any errors in the rendering process, these will appear in the Render tab. The error message will tell you where in your document your error was. Find it and correct it.<sup>3</sup> Otherwise, you should see your document.

Extra 1: the rendering process as you did it doesn't produce that nice display of a dataframe that I had in one of my screenshots. To get that, alter the YAML block to read:

```
format:
  html:
    df-print: paged
```

This way, anyone reading your document can actually page through the dataframes you display in the same way that you did, to check that they contain the right things.

Extra 2: you might prefer to have a preview of your document within R Studio. To make this happen, look for the gear wheel to the right of Render. Click the arrow beside it, and in the drop-down, click on Preview in Viewer Pane. Render again, and you'll see the rendered version of your document in a Viewer pane on the right. This puts the thing you're writing and what it will look like side by side.

Extra 3: you might be annoyed at having to remember to save things. If you are, you can enable auto-saving. To do this, go to Tools and select Global Options. Select Code (on the left) and Saving (at the top). Click on Automatically Save when editor loses focus, to put a check mark in the box on the left of it. Change the pull-down below that to Save and Write Changes. Click OK. Now, as soon as you pause for a couple of seconds, everything unsaved will be saved.



- (j) Practice handing in your rendered Quarto document, as if it were an assignment that was worth something. (It is good to get the practice in a low-stakes situation, so that you'll know what to do next week.)

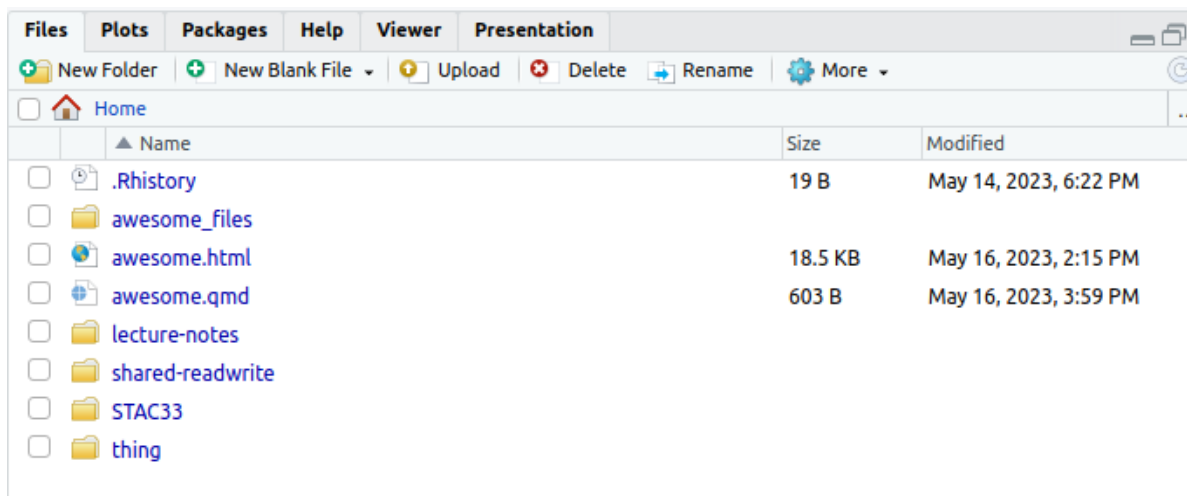
#### Solution

There are two steps: download the HTML file onto your computer, and then handing it in on Quercus. To download: find the HTML file that you want to download in the Files pane on

---

<sup>3</sup>A big part of coding is dealing with errors. You will forget things, and it is fine. In the same way that it doesn't matter how many times you get knocked down, it's key that you get up again each time: it doesn't matter how many errors you made, it's key that you fix them. If you want something to sing along with while you do this, I recommend [this](#).

the right. You might need to click on Files at the top, especially if you had a Viewer open there before:



I called my Quarto document `awesome` and the file I was working on was called `awesome.qmd` (which stands for “Quarto Markdown”). That’s the file I had to render to produce the output. My output file itself is called `awesome.html`. That’s the file I want to hand in. If you called your file something different when you saved it, that’s the thing to look for: there should be something ending in `.qmd` and something with the same first part ending in `.html`.

Click the checkbox to the left of the HTML file. Now click on More above the bottom-right pane. This pops up a menu from which you choose Export. This will pop up another window called Export Files, where you put the name that the file will have on your computer. (I usually leave the name the same.) Click Download. The file will go to your Downloads folder, or wherever things you download off the web go.

Now, to hand it in. Open up Quercus at [q.utoronto.ca](http://q.utoronto.ca), log in and navigate to this course. Click Assignments. Click (the title of) Assignment 0. There is a big blue Start Assignment button top right. Click it. You’ll get a File Upload at the bottom of the screen. Click Choose File and find the HTML file that you downloaded. Click Open (or equivalent on your system). The name of the file should appear next to Choose File. Click Submit Assignment. You’ll see Submitted at the top right, and below that is a Submission Details window and the file you uploaded.

---

New Attempt

## Submission

---

✓ Submitted!

May 16 at 4:23p.m.

[Submission Details](#)

[Download awesome.html](#)

Comments:

No Comments

---

pdf

---

You should be in the habit of *always* checking what you hand in, by downloading it again and looking at it to make sure it's what you thought you had handed in.

If you want to try this again, you can try again as many times as you like, by making a New Attempt. (For the real thing, you can use this if you realize you made a mistake in something you submitted. The graders' instructions, for the real thing, are to grade the *last* file submitted, so in that case you need to make sure that the last thing submitted before the due date includes *everything* that you want graded. Here, though, it doesn't matter.)

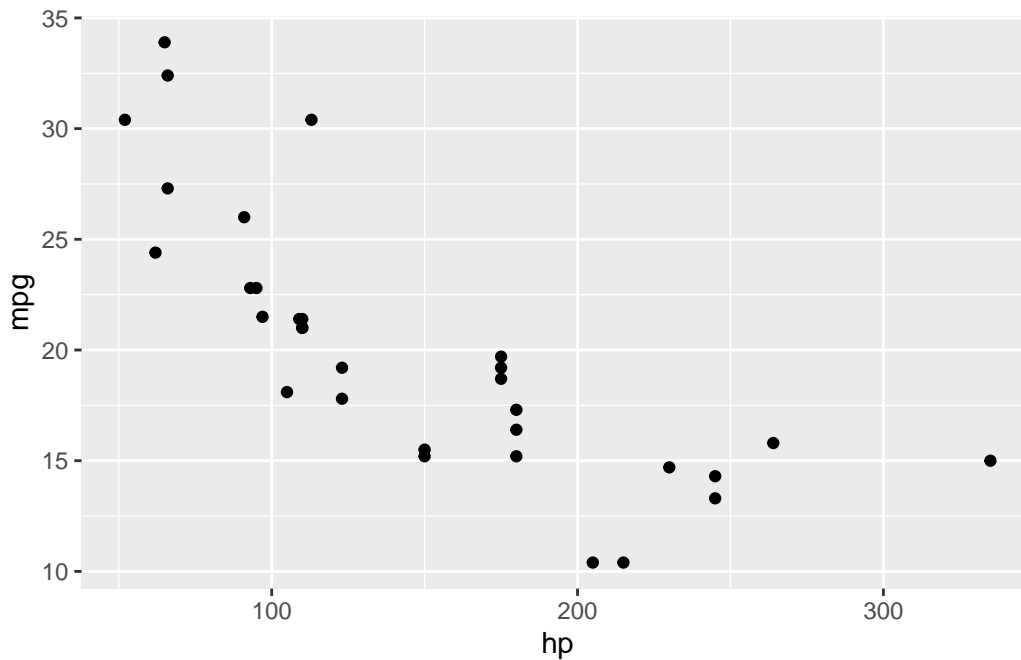
■

- (k) Optional extra. Something more ambitious: make a scatterplot of gas mileage `mpg`, on the  $y$  axis, against horsepower, `hp`, on the  $x$ -axis.

Solution

That goes like this. I'll explain the steps below.

```
library(tidyverse)
ggplot(mtcars, aes(x=hp, y=mpg)) + geom_point()
```



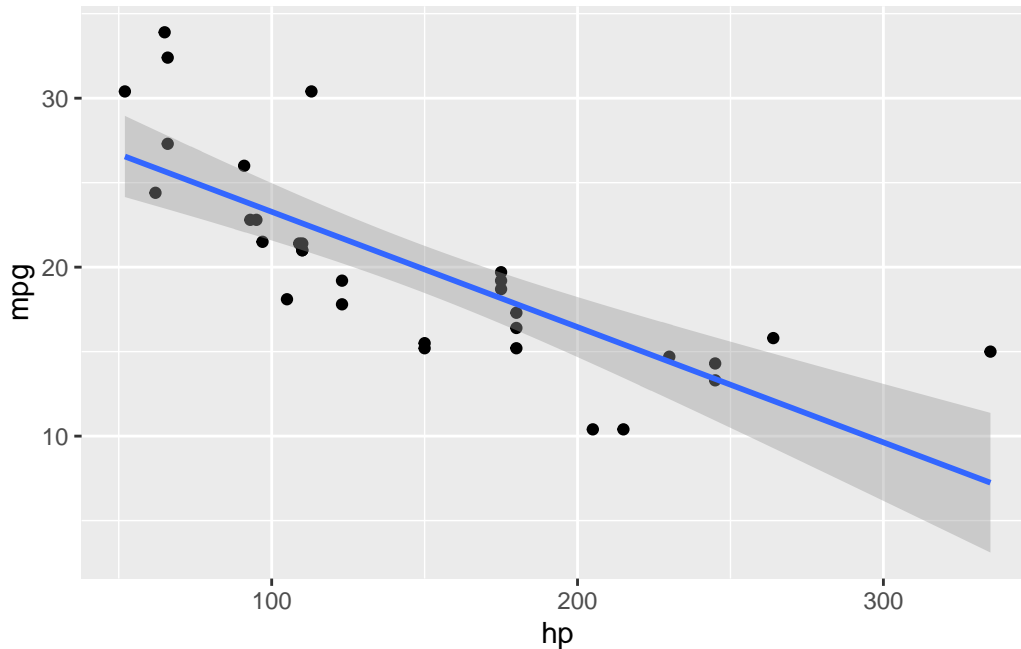
This shows a somewhat downward trend, which is what you'd expect, since a larger `hp` value means a more powerful engine, which will probably consume more gas and get *fewer* miles per gallon. As for the code: to make a `ggplot` plot, as we will shortly see in class, you first need a `ggplot` statement that says what to plot. The first thing in a `ggplot` is a data frame (`mtcars` here), and then the `aes` says that the plot will have `hp` on the *x*-axis and `mpg` on the *y*-axis, taken from the data frame that you specified. That's all of the what-to-plot. The last thing is how to plot it; `geom_point()` says to plot the data values as points.

You might like to add a regression line to the plot. That is a matter of adding this to the end of the plotting command:

```
ggplot(mtcars, aes(x=hp, y=mpg)) + geom_point() + geom_smooth(method="lm")
```

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





The line definitely goes downhill. Decide for yourself how well you think a line fits these data.

■

## 1.4 Reading data from a file

In this question, we read a file from the web and do some descriptive statistics and a graph. This is very like what you will be doing on future assignments, so it's good to practice it now.

Take a look at the data file at <http://ritsokiguess.site/datafiles/jumping.txt>. These are measurements on 30 rats that were randomly made to do different amounts of jumping by group (we'll see the details later in the course). The control group did no jumping, and the other groups did “low jumping” and “high jumping”. The first column says which jumping group each rat was in, and the second is the rat's bone density (the experimenters' supposition was that more jumping should go with higher bone density).

(a) What are the two columns of data separated by? (The fancy word is “delimited”).

Solution

Exactly one space. This is true all the way down, as you can check.

■

- (b) Make a new Quarto document. Leave the YAML block, but get rid of the rest of the template document. Start with a code chunk containing `library(tidyverse)`. Run it.

Solution

You will get either the same message as before or nothing. (I got nothing because I had already loaded the `tidyverse` in this session.)

■

- (c) Put the URL of the data file in a variable called `my_url`. Then use `read_delim` to read in the file. (See solutions for how.) `read_delim` reads data files where the data values are always separated by the same single character, here a space. Save the data frame in a variable `rats`.

Solution

Like this:

```
my_url <- "http://ritsokiguess.site/datafiles/jumping.txt"
rats <- read_delim(my_url, " ")
```

Rows: 30 Columns: 2

-- Column specification -----

Delimiter: " "

chr (1): group

dbl (1): density

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.

The second thing in `read_delim` is the thing that separates the data values. Often when you use `read_delim` it'll be a space.

Hint: to get the file name into `my_url`, the best way is to *right-click* on the link, and select Copy Link Address (or equivalent in your browser). That puts in on your clipboard. Then make a code chunk and put this in it (you'll probably only need to type one quote symbol, because R Studio will supply the other one):

```
my_url <- ""
```

then put the cursor between the two quote symbols and paste. This is better than selecting the URL in my text and then copy-pasting that because odd things happen if it happens to span two lines on your screen. (URLs tend to be rather long, so this is not impossible.)

■

- (d) Take a look at your data frame, by making a new code chunk and putting the data frame's name in it (as we did with `mtcars`).

Solution

```
rats

# A tibble: 30 x 2
  group density
  <chr>   <dbl>
1 Control    611
2 Control    621
3 Control    614
4 Control    593
5 Control    593
6 Control    653
7 Control    600
8 Control    554
9 Control    603
10 Control   569
# i 20 more rows
```

There are 30 rows and two columns, as there should be.



- (e) Find the mean bone density for rats that did each amount of jumping.

Solution

This is something you'll see a lot: `group_by` followed by `summarize`. Reminder: to get that funny thing with the percent signs (called the “pipe symbol”), type control-shift-M (or equivalent on a Mac):

```
rats %>% group_by(group) %>%
  summarize(m = mean(density))

# A tibble: 3 x 2
  group      m
  <chr>   <dbl>
1 Control  601.
2 Highjump 639.
3 Lowjump  612.
```

The mean bone density is clearly highest for the high jumping group, and not much different between the low-jumping and control groups.

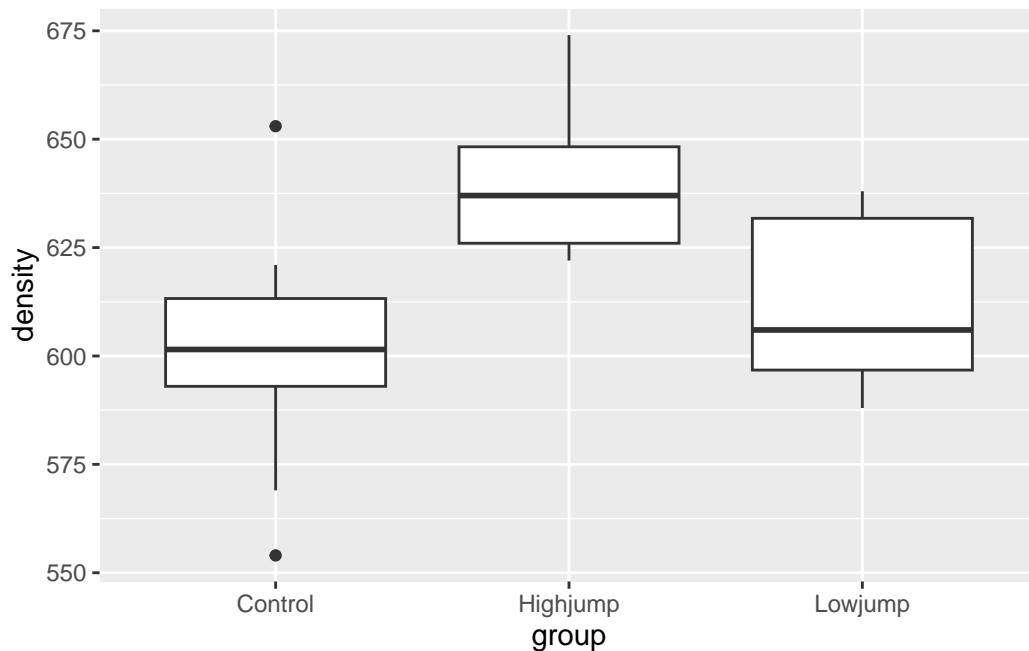
■

(f) Make a boxplot of bone density for each jumping group.

Solution

On a boxplot, the groups go across and the values go up and down, so the right syntax is this:

```
ggplot(rats, aes(x=group, y=density)) + geom_boxplot()
```



Given the amount of variability, the control and low-jump groups are very similar (with the control group having a couple of outliers), but the high-jump group seems to have a consistently higher bone density than the others.

This is more or less in line with what the experimenters were guessing, but it seems that it has to be high jumping to make a difference.

You might recognize that this is the kind of data where we would use analysis of variance, which we will do later on in the course: we are comparing several (here three) groups.

■

## 1.5 Reading files different ways

This question is about different ways of reading data files. If you're working online (using `r.datatools` or similar), start at the beginning. If you're using R Studio running on your own computer, start at part (here).

- (a) Log in to `r.datatools.utoronto.ca`. Open up a project (or start a new one), and watch the spinning circles for a few minutes. When that's done, create a new Quarto Document with File, New File, Quarto Document. Delete the "template" document, but not the top lines with `title:` and `output:` in them. Add a code chunk that contains `library(tidyverse)` and run it.

Solution

So far you (with luck) have something that looks like this:

The screenshot shows a web-based RStudio interface. At the top, there's a menu bar with 'File', 'Edit', 'Code', 'View', 'Plots', 'Session', 'Build', 'Debug', 'Profile', 'Tools', and 'Help'. Below the menu is a toolbar with icons for file operations and a search bar labeled 'Go to file/function'. The main editor area shows a file named 'Untitled1\*' with the following R code:

```

1 ---
2 title: "R Notebook"
3 output: html_notebook
4 ---
5
6 ## Packages
7
8 ```{r}
9 library(tidyverse)
10 ```

```

Below the code editor, a console window displays the output of the code chunk. It shows a list of attached packages and their versions, followed by a conflicts section.

```

— Attaching packages —
✓ ggplot2 3.0.0    ✓ purrr  0.2.5
✓ tibble  1.4.2    ✓ dplyr   0.7.6
✓ tidyr   0.8.1    ✓ stringr 1.3.1
✓ readr   1.1.1    ✓ forcats 0.3.0
— Conflicts —
✗ dplyr::filter() masks stats::filter()
✗ dplyr::lag()    masks stats::lag()

```

The console output is displayed in a light gray box. The code editor has line numbers 1 through 12 on the left margin.

If you have an error rather than that output, you probably need to install the `tidyverse` first. Make another code chunk, containing

```
install.packages("tidyverse")
```

and run it. Wait for it to finish. It may take a while. If it completes successfully (you might see the word `DONE` at the end), delete that code chunk (you don't need it any more) and try again with the `library(tidyverse)` chunk. It should run properly this time.



- (b) \* The easiest kind of files to read in are ones on the Internet, with a URL address that begins with `http` or `https`. I have a small file at [link](http://ritsokiguess.site/datafiles/testing.txt). Click the link to see it, and keep the tab open for the next part of this question. This is a text file with three things on each line, each separated by exactly one space. Read the data file into a data frame, and display your data frame.

Solution

Data values separated by exactly one space is the kind of thing that `read_delim` reads, so make another code chunk and fill it with this:

```
my_url <- "http://ritsokiguess.site/datafiles/testing.txt"
testing <- read_delim(my_url, " ")
```

Rows: 6 Columns: 3

-- Column specification -----

Delimiter: " "

chr (1): g

dbl (2): x, 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.

```
testing
```

# A tibble: 6 x 3

	x	y	g
	<dbl>	<dbl>	<chr>
1	1	10	a
2	2	11	b
3	3	14	a
4	4	13	b
5	5	18	a
6	6	21	b

When you run that, you'll see something like my output. The first part is `read_delim` telling you what it saw in the file: two columns of (whole) numbers and one column of text. The top line of the file is assumed to contain names, which are used as the names of the columns of your data frame. The bottom part of the output, obtained by putting the name of the data frame on a line by itself in your code chunk, is what the data frame actually looks like. You

ought to get into the habit of eyeballing it and checking that it looks like the values in the data file.

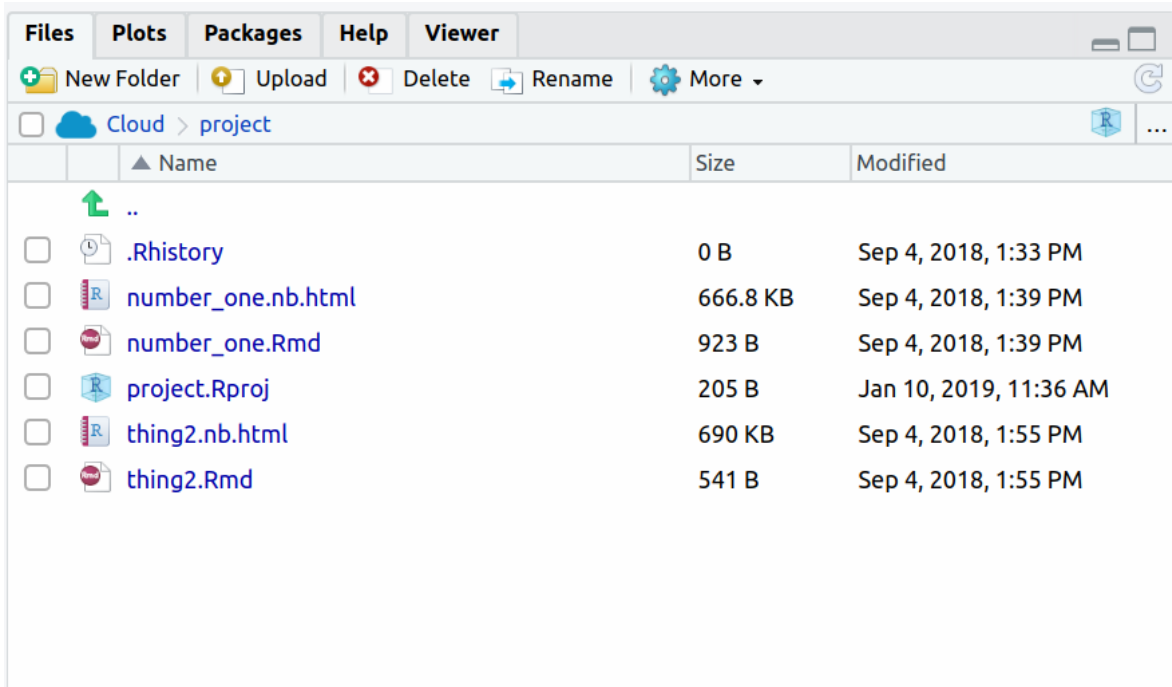
The things on the left side of the `<-` symbol (that is meant to look like an arrow pointing left) are variables that you are creating in R. You get to choose the names for them. My habit is to use `my_url` for URLs of files that I am going to read in, and (usually) to give my data frames names that say something about what they contain, but this is your choice to make.

■

- (c) You might have data in a file on your own computer. To read data from such a file, you first have to *upload* it to `r.datatools`, and then read it from there. To practice this: open a text editor (like Notepad or TextEdit). Go back to the web browser tab containing the data you used in the previous part. Copy the data from there and paste it into the text editor. Save it somewhere on your computer (like the Desktop). Upload that file, read in the data and verify that you get the right thing. (For this last part, see the Solution.)

#### Solution

I copied and pasted the data, and saved it in a file called `testing.txt` on my computer. I'm assuming that you've given it a similar name. Then go back to `r.datatools`. You should have a Files pane bottom right. If you don't see a pane bottom right at all, press Control-Shift-0 to show all the panes. If you see something bottom right but it's not Files (for example a plot), click the Files tab, and you should see a list of things that look like files, like this:





Click the Upload button (next to New Folder), click Choose File. Use the file finder to track down the file you saved on your computer, then click OK. The file should be uploaded to the same folder on `r.datatools` that your project is, and appear in the Files pane bottom right.

To read it in, you supply the file name to `read_delim` thus:

```
testing2 <- read_delim("testing.txt", " ")
```

Rows: 6 Columns: 3

-- Column specification -----

Delimiter: " "

chr (1): g

dbl (2): x, 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.

and then you look at it in the same way as before:

```
testing2
```

```
# A tibble: 6 x 3
```

	x	y	g
	<dbl>	<dbl>	<chr>
1	1	10	a
2	2	11	b
3	3	14	a
4	4	13	b
5	5	18	a
6	6	21	b

Check.



- (d) You might have a spreadsheet on your computer. Create a `.csv` file from it, and use the ideas of the last part to read it into R Studio.

Solution

Open the spreadsheet containing the data you want to read into R. If there are several sheets in the workbook, make sure you're looking at the right one. Select File, Save As, select "CSV" or "comma-separated values" and give it a name. Save the resulting file somewhere.

Then follow the same steps as the previous part to upload it to your project on R Studio Cloud. (If you look at the actual file, it will be plain text with the data values having commas between them, as the name suggests. You can open the file in R Studio by clicking on it in the Files pane; it should open top left.)

The final step is to read it into an R data frame. This uses `read_csv`; there are several `read_` functions that read in different types of file, and you need to use an appropriate one.

My spreadsheet got saved as `cars.csv`, so:

```
cars <- read_csv("cars.csv")
```

```
Rows: 38 Columns: 6
```

```
-- Column specification -----
```

```
Delimiter: ","
```

```
chr (2): Car, Country
```

```
dbl (4): MPG, Weight, Cylinders, Horsepower
```

```
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.
```

```
cars
```

```
# A tibble: 38 x 6
```

	Car	MPG	Weight	Cylinders	Horsepower	Country
	<chr>	<dbl>	<dbl>	<dbl>	<dbl>	<chr>
1	Buick Skylark	28.4	2.67	4	90	U.S.
2	Dodge Omni	30.9	2.23	4	75	U.S.
3	Mercury Zephyr	20.8	3.07	6	85	U.S.
4	Fiat Strada	37.3	2.13	4	69	Italy
5	Peugeot 694 SL	16.2	3.41	6	133	France
6	VW Rabbit	31.9	1.92	4	71	Germany
7	Plymouth Horizon	34.2	2.2	4	70	U.S.
8	Mazda GLC	34.1	1.98	4	65	Japan
9	Buick Estate Wagon	16.9	4.36	8	155	U.S.
10	Audi 5000	20.3	2.83	5	103	Germany

```
# i 28 more rows
```

Some information about different types of cars.

You are now done with this question.



- (e) \* Start here if you downloaded R and R Studio and they are running on your own computer. Open a web browser and point it at [link](http://ritsokiguess.site/datafiles/testing.txt). Click the link to see it, and keep the tab open for the next part of this question. This is a text file with three things on each line, each separated by exactly one space. Read the data file into a data frame, and display your data frame.

#### Solution

Data values separated by exactly one space is the kind of thing that `read_delim` reads, so make another code chunk and fill it with this:

```
my_url <- "http://ritsokiguess.site/datafiles/testing.txt"
testing <- read_delim(my_url, " ")
```

Rows: 6 Columns: 3

-- Column specification -----

Delimiter: " "

chr (1): g

dbl (2): x, 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.

```
testing
```

# A tibble: 6 x 3

	x	y	g
	<dbl>	<dbl>	<chr>
1	1	10	a
2	2	11	b
3	3	14	a
4	4	13	b
5	5	18	a
6	6	21	b

When you run that, you'll see something like my output. The first part is `read_delim` telling you what it saw in the file: two columns of (whole) numbers and one column of text. The top line of the file is assumed to contain names, which are used as the names of the columns of your data frame. The bottom part of the output, obtained by putting the name of the data frame on a line by itself in your code chunk, is what the data frame actually looks like. You

ought to get into the habit of eyeballing it and checking that it looks like the values in the data file.

The things on the left side of the `<-` symbol (that is meant to look like an arrow pointing left) are variables that you are creating in R. You get to choose the names for them. My habit is to use `my_url` for URLs of files that I am going to read in, and (usually) to give my data frames names that say something about what they contain, but this is your choice to make.

(This part is exactly the same whether you are running R Studio on `r.datatools` or have R Studio running on your computer. A remote file is obtained in exactly the same way regardless.)



- (f) You might have data in a file on your own computer. To read data from such a file, R has to know where to find it. Each R project lives in a folder, and one way of specifying where a data file is is to give its complete path relative to the folder that R Studio is running its current project in. This is rather complicated, so we will try a simpler way. To set this up, open a text editor (like Notepad or TextEdit). Go back to the web browser tab containing the data you used in the previous part. Copy the data from there and paste it into the text editor. Save it somewhere on your computer (like the Desktop). Follow the steps in the solution below to read the data into R.

#### Solution

I copied and pasted the data, and saved it in a file called `testing.txt` on my computer. I'm assuming that you've given it a similar name. Go back to R Studio. Create a new code chunk containing this:

```
f <- file.choose()
```

Run this code chunk. You'll see a file chooser. Find the file you saved on your computer, and click Open (or OK or whatever you see). This saves what R needs to access the file in the variable `f`. If you want to, you can look at it:

```
f
```

and you'll see what looks like a file path in the appropriate format for your system (Windows, Mac, Linux). To read the data in, you supply the file path to `read_delim` thus:

```
testing2 <- read_delim(f, " ")
```

and then you look at it in the same way as before:

```
testing2
```

```
# A tibble: 6 x 3
      x     y g
  <dbl> <dbl> <chr>
1     1     10 a
2     2     11 b
3     3     14 a
4     4     13 b
5     5     18 a
6     6     21 b
```

Check.



- (g) You might have a spreadsheet on your computer. Create a `.csv` file from it, and use the ideas of the last part to read it into R Studio.

Solution

Open the spreadsheet containing the data you want to read into R. If there are several sheets in the workbook, make sure you're looking at the right one. Select File, Save As, select "CSV" or "comma-separated values" and give it a name. Save the resulting file somewhere.

Then read it into an R data frame. This uses `read_csv`; there are several `read_` functions that read in different types of file, and you need to use an appropriate one. Before that, though, again run

```
f <- file.choose()
```

to find the `.csv` file on your computer, and then

```
cars <- read_csv(f)
```

to read it in. My spreadsheet was

```
cars
```

```
# A tibble: 38 x 6
  Car           MPG Weight Cylinders Horsepower Country
  <chr>         <dbl>   <dbl>     <dbl>     <dbl> <chr>
1 Buick Skylark  28.4    2.67         4         90 U.S.
2 Dodge Omni    30.9    2.23         4         75 U.S.
3 Mercury Zephyr 20.8    3.07         6         85 U.S.
4 Fiat Strada    37.3    2.13         4         69 Italy
```

5	Peugeot 694 SL	16.2	3.41	6	133 France
6	VW Rabbit	31.9	1.92	4	71 Germany
7	Plymouth Horizon	34.2	2.2	4	70 U.S.
8	Mazda GLC	34.1	1.98	4	65 Japan
9	Buick Estate Wagon	16.9	4.36	8	155 U.S.
10	Audi 5000	20.3	2.83	5	103 Germany

# i 28 more rows

Some information about different types of cars.



## 2 Reading in data

In this chapter, and from here on, the questions are first, and then my answers are in the second appearances of the problems with the same name. It is much better for your learning to spend some time thinking about how you would solve these problems, and after *that* you can see how I did it.

```
library(tidyverse)
```

### 2.1 Orange juice

The quality of orange juice produced by a manufacturer (identity unknown) is constantly being monitored. The manufacturer has developed a “sweetness index” for its orange juice, for which a higher value means sweeter juice. Is the sweetness index related to a chemical measure such as the amount of water-soluble pectin (parts per million) in the orange juice? Data were obtained from 24 production runs, and the sweetness and pectin content were measured for each run. The data are in [link](#). Open that link up now. You can click on that link just above to open the file.

- (a) The data values are separated by a space. Use the appropriate Tidyverse function to read the data directly from the course website into a “tibble”.
- (b) Take a look at what got read in. Do you have data for all 24 runs?
- (c) In your data frame, where did the column (variable) names come from? How did R know where to get them from?

### 2.2 Making soap

A company operates two production lines in a factory for making soap bars. The production lines are labelled A and B. A production line that moves faster may produce more soap, but may possibly also produce more “scrap” (that is, bits of soap that can no longer be made into soap bars and will have to be thrown away).

The data are in [link](#).

- (a) Read the data into R. Display the data.
- (b) There should be 27 rows. Are there? What columns are there?

## 2.3 Handling shipments

A company called Global Electronics from time to time imports shipments of a certain large part used as a component in several of its products. The size of the shipment varies each time. Each shipment is sent to one of two warehouses (labelled A and B) for handling. The data in [link](#) show the **size** of each shipment (in thousands of parts) and the direct **cost** of handling it, in thousands of dollars. Also shown is the **warehouse** (A or B) that handled each shipment.

- (a) Read the data into R and display your data frame.
- (b) Describe how many rows and columns your data frame has, and what they contain.

My solutions follow:

## 2.4 Orange juice

The quality of orange juice produced by a manufacturer (identity unknown) is constantly being monitored. The manufacturer has developed a “sweetness index” for its orange juice, for which a higher value means sweeter juice. Is the sweetness index related to a chemical measure such as the amount of water-soluble pectin (parts per million) in the orange juice? Data were obtained from 24 production runs, and the sweetness and pectin content were measured for each run. The data are in [link](#). Open that link up now. You can click on that link just above to open the file.

- (a) The data values are separated by a space. Use the appropriate Tidyverse function to read the data directly from the course website into a “tibble”.

Solution

Start with this (almost always):

```
library(tidyverse)
```

The appropriate function, the data values being separated by a space, will be `read_delim`. Put the URL as the first thing in `read_delim`, or (better) define it into a variable first:<sup>1</sup>

---

<sup>1</sup>I say “better” because otherwise the read line gets rather long. This way you read it as “the URL is some long thing that I don’t care about especially, and I what I need to do is to read the data from that URL, separated by spaces.”



```
url <- "http://ritsokiguess.site/datafiles/ojuice.txt"
juice <- read_delim(url, " ")
```

Rows: 24 Columns: 3

```
-- Column specification -----
Delimiter: " "
dbl (3): run, sweetness, pectin
```

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.

`read_delim` (or `read_csv` or any of the others) tell you what variables were read in, and also tell you about any “parsing errors” where it couldn’t work out what was what. Here, we have three variables, which is entirely consistent with the three columns of data values in the file.

`read_delim` can handle data values separated by *any* character, not just spaces, but the separating character, known as a “delimiter”, does *not* have a default, so you have to say what it is, every time.



(b) Take a look at what got read in. Do you have data for all 24 runs?

Solution

Type the name of the data frame in a code chunk (a new one, or add it to the end of the previous one). Because this is actually a “tibble”, which is what `read_delim` reads in, you’ll only actually see the first 10 lines, but it will tell you how many lines there are altogether, and you can click on the appropriate thing to see the rest of it.

```
juice
```

```
# A tibble: 24 x 3
  run sweetness pectin
  <dbl>      <dbl> <dbl>
1     1         5.2   220
2     2         5.5   227
3     3          6   259
4     4         5.9   210
5     5         5.8   224
6     6          6   215
7     7         5.8   231
```

```

8      8      5.6    268
9      9      5.6    239
10     10     5.9    212
# i 14 more rows

```

I appear to have all the data. If you want further convincing, click Next a couple of times to be sure that the runs go down to number 24.



- (c) In your data frame, where did the column (variable) names come from? How did R know where to get them from?

#### Solution

They came from the top line of the data file, so we didn't have to specify them. This is the default behaviour of all the `read_` functions, so we don't have to ask for it specially.

Extra: in fact, if the top line of your data file is *not* variable names, *that's* when you have to say something special. The `read_` functions have an option `col_names` which can either be `TRUE` (the default), which means "read them in from the top line", `FALSE` ("they are not there, so make some up") or a list of column names to use. You might use the last alternative when the column names that are in the file are *not* the ones you want to use; in that case, you would also say `skip=1` to skip the first line. For example, with file `a.txt` thus:

```

a b
1 2
3 4
5 6

```

you could read the same data but call the columns `x` and `y` thus:

```
read_delim("a.txt", " ", col_names = c("x", "y"), skip = 1)
```

```
Rows: 3 Columns: 2
```

```
-- Column specification -----
```

```
Delimiter: " "
```

```
dbl (2): x, 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.
```

```
# A tibble: 3 x 2
      x     y
  <dbl> <dbl>
1     1     2
2     3     4
3     5     6
```



## 2.5 Making soap

A company operates two production lines in a factory for making soap bars. The production lines are labelled A and B. A production line that moves faster may produce more soap, but may possibly also produce more “scrap” (that is, bits of soap that can no longer be made into soap bars and will have to be thrown away).

The data are in [link](#).

(a) Read the data into R. Display the data.

Solution

Read directly from the URL, most easily:

```
url <- "http://ritsokiguess.site/datafiles/soap.txt"
soap <- read_delim(url, " ")
```

```
Rows: 27 Columns: 4
```

```
-- Column specification -----
```

```
Delimiter: " "
```

```
chr (1): line
```

```
dbl (3): case, scrap, speed
```

```
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.
```

```
soap
```

```
# A tibble: 27 x 4
  case scrap speed line
  <dbl> <dbl> <dbl> <chr>
1     1    218   100 a
2     2    248   125 a
3     3    360   220 a
4     4    351   205 a
5     5    470   300 a
6     6    394   255 a
7     7    332   225 a
8     8    321   175 a
9     9    410   270 a
10    10    260   170 a
# i 17 more rows
```



(b) There should be 27 rows. Are there? What columns are there?

Solution

There are indeed 27 rows, one per observation. The column called **case** identifies each particular run of a production line (scroll down to see that it gets to 27 as well). Though it is a number, it is an identifier variable and so should not be treated quantitatively. The other columns (variables) are **scrap** and **speed** (quantitative) and **line** (categorical). These indicate which production line was used for each run, the speed it was run at, and the amount of scrap produced.

This seems like an odd place to end this question, but later we'll be using these data to draw some graphs.



## 2.6 Handling shipments

A company called Global Electronics from time to time imports shipments of a certain large part used as a component in several of its products. The size of the shipment varies each time. Each shipment is sent to one of two warehouses (labelled A and B) for handling. The data in [link](#) show the **size** of each shipment (in thousands of parts) and the direct **cost** of handling it, in thousands of dollars. Also shown is the **warehouse** (A or B) that handled each shipment.

(a) Read the data into R and display your data frame.

## Solution

If you open the data file in your web browser, it will probably open as a spreadsheet, which is not really very helpful, since then it is not clear what to do with it. You could, I suppose, save it and upload it to R Studio Cloud, but it requires much less brainpower to open it directly from the URL:

```
url <- "http://ritsokiguess.site/datafiles/global.csv"
shipments <- read_csv(url)
```

Rows: 10 Columns: 3

-- Column specification -----

Delimiter: ","

chr (1): warehouse

dbl (2): size, cost

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.

If you display your data frame and it looks like this, you are good (you can give the data frame any name):

```
shipments
```

```
# A tibble: 10 x 3
  warehouse size cost
  <chr>      <dbl> <dbl>
1 A          225 12.0
2 B          350 14.1
3 A          150  8.93
4 A          200 11.0
5 A          175 10.0
6 A          180 10.1
7 B          325 13.8
8 B          290 13.3
9 B          400 15
10 A         125  7.97
```



(b) Describe how many rows and columns your data frame has, and what they contain.

Solution

It has 10 rows and 3 columns. *You need to say this.*

That is, there were 10 shipments recorded, and for each of them, 3 variables were noted: the size and cost of the shipment, and the warehouse it was handled at.

We will also be making some graphs of these data later.



## 3 Data exploration

```
library(tidyverse)
```

### 3.1 North Carolina births

The data in file [link](#) are about 500 randomly chosen births of babies in North Carolina. There is a lot of information: not just the weight at birth of the baby, but whether the baby was born prematurely, the ages of the parents, whether the parents are married, how long (in weeks) the pregnancy lasted (this is called the “gestation”) and so on.

- (a) Read in the data from the file into R, bearing in mind what type of file it is.
- (b) From your output, verify that you have the right number of observations and that you have several variables. Which of your variables correspond to birthweight, prematurity and length of pregnancy? (You might have to make guesses based on the names of the variables.)
- (c) The theory behind the  $t$ -test (which we do later) says that the distribution of birth weights should be (approximately) normally distributed. Obtain a histogram of the birth weights. Does it look approximately normal? Comment briefly. (You’ll have to pick a number of bins for your histogram first. I don’t mind very much what you pick, as long as it’s not obviously too many or too few bins.)

### 3.2 More about the NC births

This is an exploration of some extra issues around the North Carolina births data set.

- (a) How short does a pregnancy have to be, for the birth to be classified as “premature”? Deduce this from the data, by drawing a suitable graph or otherwise.
- (b) Explore the relationship between birth weight and length of pregnancy (“gestation”) using a suitable graph. What do you see?

- (c) Do a web search to find the standard (North American) definition of a premature birth. Does that correspond to what you saw in the data? Cite the website you used, for example by saying “according to URL, ...”, with URL replaced by the address of the website you found.

### 3.3 Nenana, Alaska

Nenana, Alaska, is about 50 miles west of Fairbanks. Every spring, there is a contest in Nenana. A wooden tripod is placed on the frozen river, and people try to guess the exact minute when the ice melts enough for the tripod to fall through the ice. The contest started in 1917 as an amusement for railway workers, and has taken place every year since. Now, hundreds of thousands of people enter their guesses on the Internet and the prize for the winner can be as much as \$300,000.

Because so much money is at stake, and because the exact same tripod is placed at the exact same spot on the ice every year, the data are consistent and accurate. The data are in [link](#).

- (a) Read the data into R. Note that the values are separated by *tabs* rather than spaces, so you’ll need an appropriate `read_` to read it in.
- (b) Find a way of displaying how many rows and columns your data frame has, and some of the values. Describe the first and last of the variables that you appear to have.
- (c) Dates and times are awkward to handle with software. (We see more ways later in the course.) The column `JulianDate` expresses the time that the tripod fell through the ice as a fractional number of days since December 31. This enables the time (as a fraction of the way through the day) to be recorded as well, the whole thing being an ordinary number. Make a histogram of the Julian dates. Comment briefly on its shape.
- (d) Plot `JulianDate` against `Year` on a scatterplot. What recent trends, if any, do you see? Comment briefly.

### 3.4 Computerized accounting

Beginning accounting students need to learn to learn to audit in a computerized environment. A sample of beginning accounting students took each of two tests: the Computer Attitude Scale (CAS) and the Computer Anxiety Rating Scale (CARS). A higher score in each indicates greater anxiety around computers. The test scores are scaled to be between 0 and 5. Also noted was each student’s gender. The data are in <http://ritsokiguess.site/datafiles/compatt.txt>. The data values are separated by spaces.

- (a) Read the data into R. Do you have what you expected? Explain briefly.



- (b) How many males and females were there in the sample?
- (c) Do the CAS scores tend to be higher for females or for males? Draw a suitable graph to help you decide, and come to a conclusion.
- (d) Find the median CAS scores for each gender. Does this support what you saw on your plot? Explain briefly.
- (e) Find the mean and standard deviation of both CAS and CARS scores (for all the students combined, ie. not separated by gender) *without* naming those columns explicitly.

### 3.5 Test scores in two classes

Open R Studio. Create a new Text File by selecting File, New File and Text File. You should see a new empty, untitled window appear at the top left. In that window, type or copy the data below (which are scores on a test for students in two different classes):

```
class score
ken 78
ken 62
ken 59
ken 69
ken 81
thomas 83
thomas 77
thomas 63
thomas 61
thomas 79
thomas 72
```

Save the file, using a filename of your choice (with, perhaps, extension `.txt`). Or, if you prefer, use the one at [link](#).

- (a) Read the data into a data frame called `marks`, using `read_delim`, and list the data frame (by typing its name) to confirm that you read the data values properly. Note that the top line of the data file contains the names of the variables, as it ought to.
- (b) \* Obtain side-by-side boxplots of the scores for each class.
- (c) Do the two classes appear to have similar or different scores, on average? Explain briefly.
- (d) Obtain a boxplot of all the scores together, regardless of which class they came from.

- (e) Compute the median score (of all the scores together). Does this seem about right, looking at the boxplot? Explain briefly.

### 3.6 Unprecedented rainfall

In 1997, a company in Davis, California, had problems with odour in its wastewater facility. According to a company official, the problems were caused by “unprecedented weather conditions” and “because rainfall was at 170 to 180 percent of its normal level, the water in the holding ponds took longer to exit for irrigation, giving it more time to develop an odour.”

Annual rainfall data for the Davis area is [here](#). Note that clicking on the link will display the file, and *right*-clicking on the link will give you some options, one of which is Copy Link Address, which you can then paste into your R Notebook.

The rainfall is measured in inches.

- (a) Read in and display (some of) the data.
- (b) Summarize the data frame.
- (c) Make a suitable plot of the rainfall values. (We are not, for the time being, concerned about the years.)
- (d) How would you describe the shape of the distribution of rainfall values?
- (e) In the quote at the beginning of the question, where do you think the assertion that the 1997 rainfall was “at 170 to 180 percent of its normal level” came from? Explain briefly.
- (f) Do you think the official’s calculation was reasonable? Explain briefly. (Note that this is not the same as asking whether the official’s calculation was *correct*. This is an important distinction for you to make.)
- (g) Do you think that the official was right to use the word “unprecedented” to describe the 1997 rainfall? Justify your answer briefly.

### 3.7 Learning algebra

At a high school in New Jersey, teachers were interested in what might help students to learn algebra. One idea was laptops as a learning aid, to see whether having access to one helped with algebra scores. (This was some time ago.) The 20 students in one class were given laptops to use in school and at home, while the 27 students in another class were not given laptops. For all of these students, the final exam score in algebra was recorded. The data are in <http://ritsokiguess.site/datafiles/algebra.txt>, with two columns, one indicating whether the student received a laptop or not, and the other giving their score on the algebra final exam.

- (a) Read in and display (some of) the data. Do you have (i) the correct number of observations, and (ii) the correct *type* of columns? Explain briefly.
- (b) Make a suitable graph of these data.
- (c) Comment briefly on your graph, thinking about what the teachers would like to know.
- (d) Work out the median and inter-quartile range for the students who did and who did not have laptops, and compare with the boxplot. (In R, the inter-quartile range is `IQR` in uppercase.)

My solutions follow:

### 3.8 North Carolina births

The data in file [link](#) are about 500 randomly chosen births of babies in North Carolina. There is a lot of information: not just the weight at birth of the baby, but whether the baby was born prematurely, the ages of the parents, whether the parents are married, how long (in weeks) the pregnancy lasted (this is called the “gestation”) and so on.

- (a) Read in the data from the file into R, bearing in mind what type of file it is.

Solution

This is a `.csv` file (it came from a spreadsheet), so it needs reading in accordingly. Work directly from the URL:

```
myurl <- "http://ritsokiguess.site/datafiles/ncbirths2.csv"
bw <- read_csv(myurl)
```

```
Rows: 500 Columns: 10
-- Column specification -----
Delimiter: ","
dbl (10): father_age, mother_age, weeks_gestation, pre_natal_visits, marital...

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.
```

This shows you which variables the data set has (some of the names got a bit mangled), and it shows you that they are all integers except for the birth weight (a decimal number).

The easiest way to find out how many rows and columns there are is simply to list the data frame:

```
bw
```

```
# A tibble: 500 x 10
  father_age mother_age weeks_gestation pre_natal_visits marital_status
    <dbl>      <dbl>      <dbl>          <dbl>          <dbl>
1         27         26         38             14             1
2         35         33         40             11             1
3         34         22         37             10             2
4         NA         16         38              9             2
5         35         33         39             12             1
6         32         24         36             12             1
7         33         33         38             15             2
8         38         35         38             16             1
9         28         29         40              5             1
10        NA         19         34             10             2
# i 490 more rows
# i 5 more variables: mother_weight_gained <dbl>, low_birthweight <dbl>,
#   weight_pounds <dbl>, premie <dbl>, few_visits <dbl>
```

or you can take a “glimpse” of it, which is good if you have a lot of columns:

```
glimpse(bw)
```

```
Rows: 500
Columns: 10
$ father_age      <dbl> 27, 35, 34, NA, 35, 32, 33, 38, 28, NA, 28, 34, N~
$ mother_age      <dbl> 26, 33, 22, 16, 33, 24, 33, 35, 29, 19, 26, 31, 1~
$ weeks_gestation <dbl> 38, 40, 37, 38, 39, 36, 38, 38, 40, 34, 39, 39, 3~
$ pre_natal_visits <dbl> 14, 11, 10, 9, 12, 12, 15, 16, 5, 10, 15, 15, 0, ~
$ marital_status  <dbl> 1, 1, 2, 2, 1, 1, 2, 1, 1, 2, 1, 1, 2, 2, 2, 2, 2~
$ mother_weight_gained <dbl> 32, 23, 50, NA, 15, 12, 60, 2, 20, NA, 45, 22, 20~
$ low_birthweight <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0~
$ weight_pounds   <dbl> 6.8750, 6.8125, 7.2500, 8.8125, 8.8125, 5.8125, 6~
$ premie          <dbl> 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0~
$ few_visits      <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0~
```

Either of these displays show that there are 500 rows (observations, here births) and 10 columns (variables), and they both show what the variables are called. So they’re both good as an answer to the question.

Extra: As is rather too often the way, the original data weren't like this, and I had to do some tidying first. Here's the original:

```
my_old_url <- "http://ritsokiguess.site/datafiles/ncbirths_original.csv"
bw0 <- read_csv(my_old_url)
```

Rows: 500 Columns: 10

-- Column specification -----

Delimiter: ","

dbl (10): Father Age, Mother Age, Weeks Gestation, Pre-natal Visits, Marital...

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.

```
bw0
```

# A tibble: 500 x 10

	`Father Age` <dbl>	`Mother Age` <dbl>	`Weeks Gestation` <dbl>	`Pre-natal Visits` <dbl>
1	27	26	38	14
2	35	33	40	11
3	34	22	37	10
4	NA	16	38	9
5	35	33	39	12
6	32	24	36	12
7	33	33	38	15
8	38	35	38	16
9	28	29	40	5
10	NA	19	34	10

# i 490 more rows

# i 6 more variables: `Marital Status` <dbl>, `Mother Weight Gained` <dbl>,

# `Low Birthweight?` <dbl>, `Weight (pounds)` <dbl>, `Premie?` <dbl>,

# `Few Visits?` <dbl>

What you'll notice is that the variables have *spaces* in their names, which would require special handling later. The `glimpse` output shows you what to do about those spaces in variable names:

```
glimpse(bw0)
```

```

Rows: 500
Columns: 10
$ `Father Age`      <dbl> 27, 35, 34, NA, 35, 32, 33, 38, 28, NA, 28, 34, ~
$ `Mother Age`      <dbl> 26, 33, 22, 16, 33, 24, 33, 35, 29, 19, 26, 31, ~
$ `Weeks Gestation` <dbl> 38, 40, 37, 38, 39, 36, 38, 38, 40, 34, 39, 39, ~
$ `Pre-natal Visits` <dbl> 14, 11, 10, 9, 12, 12, 15, 16, 5, 10, 15, 15, 0~
$ `Marital Status`  <dbl> 1, 1, 2, 2, 1, 1, 2, 1, 1, 2, 1, 1, 2, 2, 2, 2, ~
$ `Mother Weight Gained` <dbl> 32, 23, 50, NA, 15, 12, 60, 2, 20, NA, 45, 22, ~
$ `Low Birthweight?` <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, ~
$ `Weight (pounds)` <dbl> 6.8750, 6.8125, 7.2500, 8.8125, 8.8125, 5.8125, ~
$ `Premie?`         <dbl> 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, ~
$ `Few Visits?`     <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, ~

```

What you have to do is to surround the variable name with “backticks”. (On my keyboard, that’s on the key to the left of number 1, where the squiggle is, that looks like a backwards apostrophe. Probably next to `Esc`, depending on the layout of your keyboard.) For example, to get the mean mother’s age, you have to do this:

```
bw0 %>% summarize(mom_mean = mean(`Mother Age`))
```

```

# A tibble: 1 x 1
  mom_mean
  <dbl>
1      26.9

```

Although almost all of the variables are stored as integers, the ones that have a question mark in their name are actually “logical”, true or false, with 1 denoting true and 0 false. We could convert them later if we want to. A question mark is not a traditional character to put in a variable name either, so we have to surround these variables with backticks too.

In fact, all the variables have “illegal” names in one way or another: they contain spaces, or question marks, or brackets. So they *all* need backticks, which, as you can imagine, is rather awkward. The Capital Letters at the start of each word are also rather annoying to type every time.

People who collect data are not always the people who analyze it, so there is not always a lot of thought given to column names in spreadsheets.

So how did I get you a dataset with much more sane variable names? Well, I used the `janitor` package, which has a function in it called `clean_names`. This is what it does:

```

library(janitor)
bw0 %>% clean_names() %>% glimpse()

```

```

Rows: 500
Columns: 10
$ father_age      <dbl> 27, 35, 34, NA, 35, 32, 33, 38, 28, NA, 28, 34, N~
$ mother_age      <dbl> 26, 33, 22, 16, 33, 24, 33, 35, 29, 19, 26, 31, 1~
$ weeks_gestation <dbl> 38, 40, 37, 38, 39, 36, 38, 38, 40, 34, 39, 39, 3~
$ pre_natal_visits <dbl> 14, 11, 10, 9, 12, 12, 15, 16, 5, 10, 15, 15, 0, ~
$ marital_status  <dbl> 1, 1, 2, 2, 1, 1, 2, 1, 1, 2, 1, 1, 2, 2, 2, 2~
$ mother_weight_gained <dbl> 32, 23, 50, NA, 15, 12, 60, 2, 20, NA, 45, 22, 20~
$ low_birthweight <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0~
$ weight_pounds   <dbl> 6.8750, 6.8125, 7.2500, 8.8125, 8.8125, 5.8125, 6~
$ premie          <dbl> 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0~
$ few_visits      <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0~

```

All the spaces have been replaced by underscores, the question marks and brackets have been removed, and all the uppercase letters have been made lowercase. The spaces have been replaced by underscores because an underscore is a perfectly legal thing to have in a variable name. I saved this dataset into the file you read in.



- (b) From your output, verify that you have the right number of observations and that you have several variables. Which of your variables correspond to birthweight, prematurity and length of pregnancy? (You might have to make guesses based on the names of the variables.)

Solution

As a reminder:

```
glimpse(bw)
```

```

Rows: 500
Columns: 10
$ father_age      <dbl> 27, 35, 34, NA, 35, 32, 33, 38, 28, NA, 28, 34, N~
$ mother_age      <dbl> 26, 33, 22, 16, 33, 24, 33, 35, 29, 19, 26, 31, 1~
$ weeks_gestation <dbl> 38, 40, 37, 38, 39, 36, 38, 38, 40, 34, 39, 39, 3~
$ pre_natal_visits <dbl> 14, 11, 10, 9, 12, 12, 15, 16, 5, 10, 15, 15, 0, ~
$ marital_status  <dbl> 1, 1, 2, 2, 1, 1, 2, 1, 1, 2, 1, 1, 2, 2, 2, 2~
$ mother_weight_gained <dbl> 32, 23, 50, NA, 15, 12, 60, 2, 20, NA, 45, 22, 20~
$ low_birthweight <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0~
$ weight_pounds   <dbl> 6.8750, 6.8125, 7.2500, 8.8125, 8.8125, 5.8125, 6~
$ premie          <dbl> 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0~
$ few_visits      <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0~

```

I do indeed have 500 observations (rows) on 10 variables (columns; “several”). (If you don’t have several variables, check to see that you didn’t use `read_delim` or something by mistake.) After that, you see all the variables by name, with what type of values they have,<sup>1</sup> and the first few of the values.<sup>2</sup>

The variable `weight_pounds` is the birthweight (in pounds), `premie` is 1 for a premature baby and 0 for a full-term baby, and `weeks_gestation` is the number of weeks the pregnancy lasted.



- (c) The theory behind the  $t$ -test (which we do later) says that the birth weights should be (approximately) normally distributed. Obtain a histogram of the birth weights. Does it look approximately normal? Comment briefly. (You’ll have to pick a number of bins for your histogram first. I don’t mind very much what you pick, as long as it’s not obviously too many or too few bins.)

Solution

You’ll have seen that I often start with 10 bins, or maybe not quite that many if I don’t have much data, and this is a decent general principle. That would give

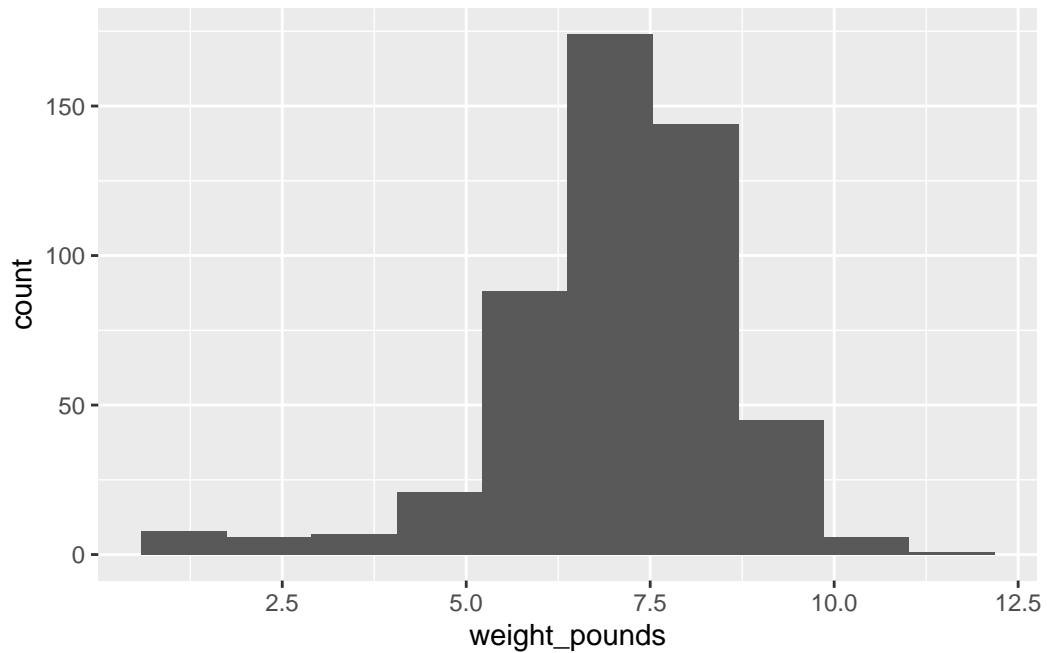
```
ggplot(bw, aes(x = weight_pounds)) + geom_histogram(bins = 10)
```

---

<sup>1</sup>these are mostly `int`, that is, integer.

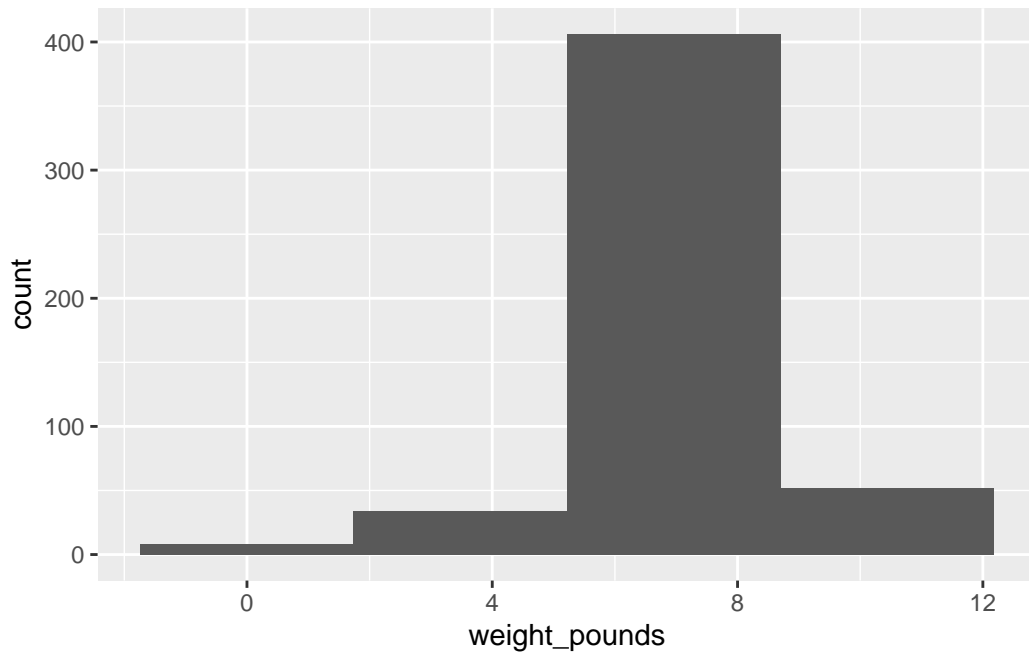
<sup>2</sup>Other possible variable types are `num` for (real, decimal) numbers such as birth weight, `chr` for text, and `Factor` (with the number of levels) for factors/categorical variables. We don’t have any of the last two here. There is also `lgl` for *logical*, things that were actually recorded as TRUE or FALSE. We have some variables that are actually logical ones, but they are recorded as integer values.





which is perfectly acceptable with 500 observations. You can try something a bit more or a bit less, and see how you like it in comparison. What you are looking for is a nice clear picture of *shape*. If you have too few bins, you'll lose the shape:

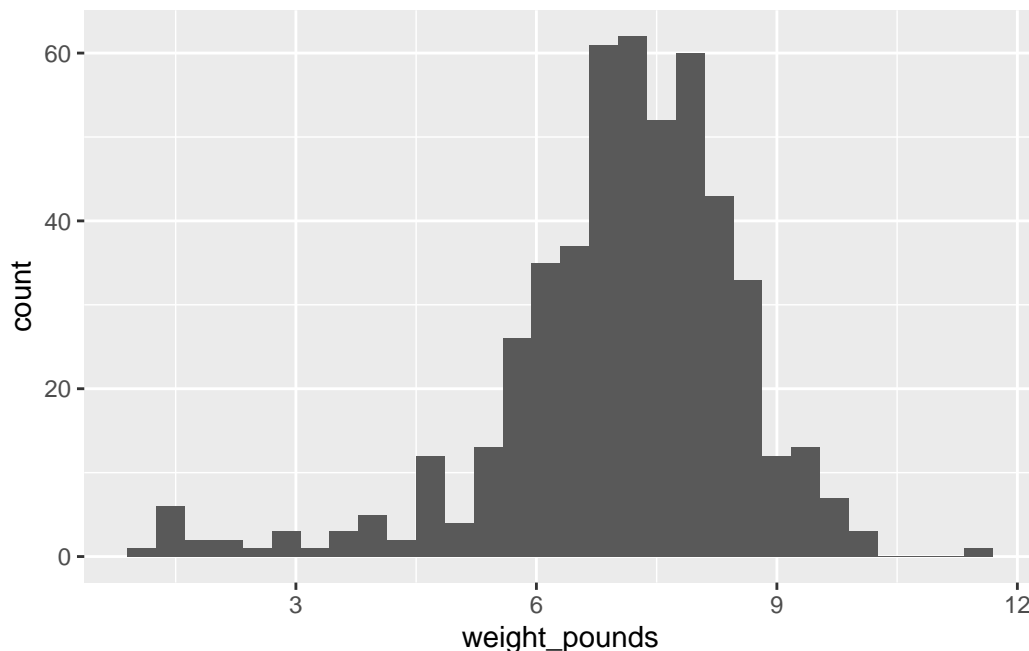
```
ggplot(bw, aes(x = weight_pounds)) + geom_histogram(bins = 4)
```



(is that leftmost bin an indication of skewness or some observations that happen to be small-ish?)

And if you have too many, the shape will be there, but it will be hard to make out in all the noise, with frequencies going up and down:

```
ggplot(bw, aes(x = weight_pounds)) + geom_histogram(bins = 30)
```



I generally am fairly relaxed about the number of bins you use, as long as it's not clearly too few or too many. You might have done exercises in the past that illustrate that the choice of number of bins (or the class intervals where you move from one bin to the next, which is another issue that I won't explore here) can make an appreciable difference to how a histogram looks.

Extra: I had some thoughts about this issue that I put in a blog post, that you might like to read: [link](#). The nice thing about Sturges' rule, mentioned there, is that you can almost get a number of bins for your histogram in your head (as long as you know the powers of 2, that is). What you do is to start with your sample size, here  $n = 500$ . You find the next power of 2 above that, which is here  $512 = 2^9$ . You then take that power and add 1, to get 10 bins. If you don't like that, you can get R to calculate it for you:

```
nclass.Sturges(bw$weight_pounds)
```

```
[1] 10
```

The place where Sturges' rule comes from is an assumption of normal data (actually a binomial approximation to the normal, backwards though that sounds). If you have less than 30 observations, you'll get fewer than 6 bins, which won't do much of a job of showing the shape. Rob Hyndman wrote a [critical note](#) about Sturges' rule in which he asserts that it is just plain wrong (if you have taken B57, this note is very readable).

So what to use instead? Well, judgment is still better than something automatic, but if you want a place to start from, something with a better foundation than Sturges is the Freedman-Diaconis rule. This, in its original formulation, gives a bin width rather than a number of bins:

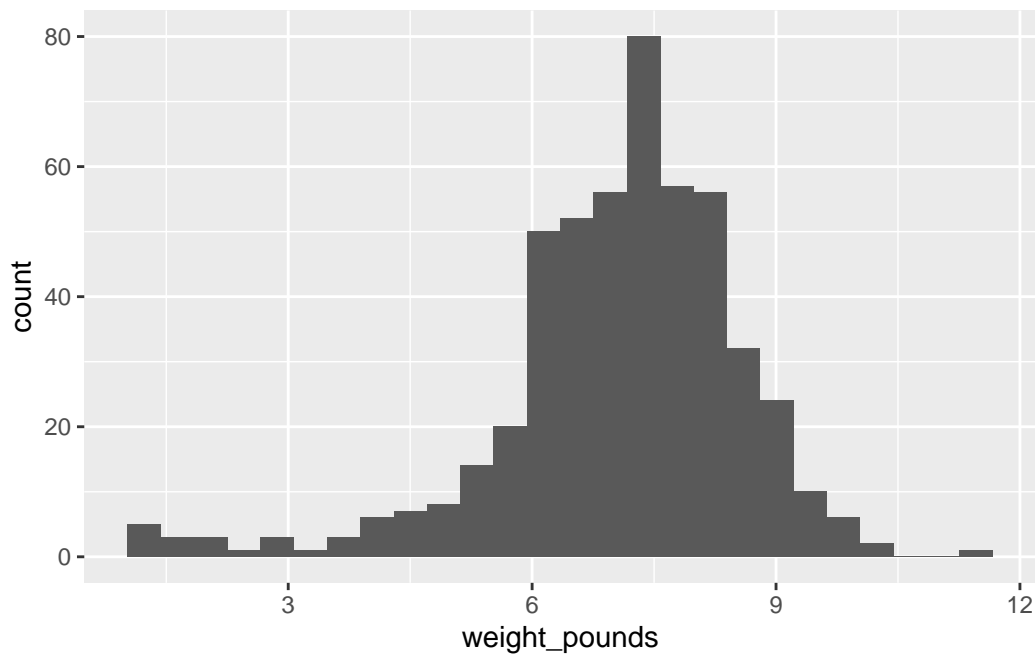
$$w = 2(IQR)n^{-1/3}$$

The nice thing about this is that it uses the interquartile range, so it won't be distorted by outliers. `geom_histogram` can take a bin width, so we can use it as follows:

```
w <- 2 * IQR(bw$weight_pounds) * 500^(-1 / 3)
w
```

```
[1] 0.4094743
```

```
ggplot(bw, aes(x = weight_pounds)) + geom_histogram(binwidth = w)
```



R also has

```
nc <- nclass.FD(bw$weight_pounds)
nc
```

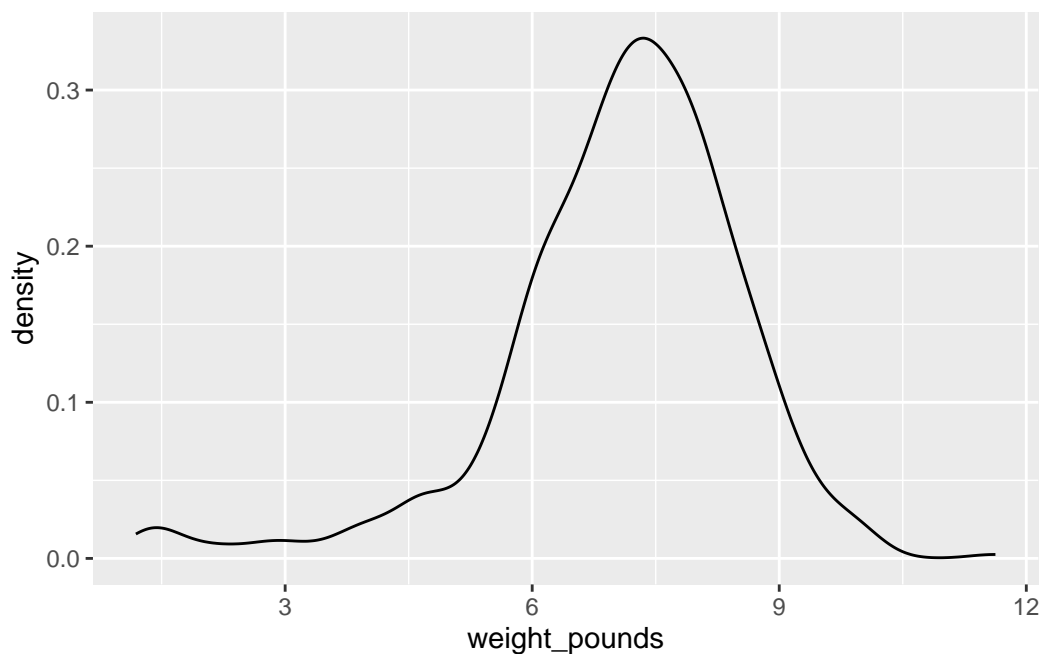
[1] 26

which turns the Freedman-Diaconis rule into a number of bins rather than a binwidth; using that gives the same histogram as we got with `binwidth`.

In my opinion, Freedman-Diaconis tends to give too many bins (here there are 26 rather than the 10 of Sturges). But I put it out there for you to make your own call.

Another way to go is a “density plot”. This is a smoothed-out version of a histogram that is not obviously frequencies in bins, but which does have a theoretical basis. It goes something like this:

```
ggplot(bw, aes(x = weight_pounds)) + geom_density()
```



`geom_density` has an optional parameter that controls how smooth or wiggly the picture is, but the default is usually good.

Alright, before we got distracted, we were assessing normality. What about that?

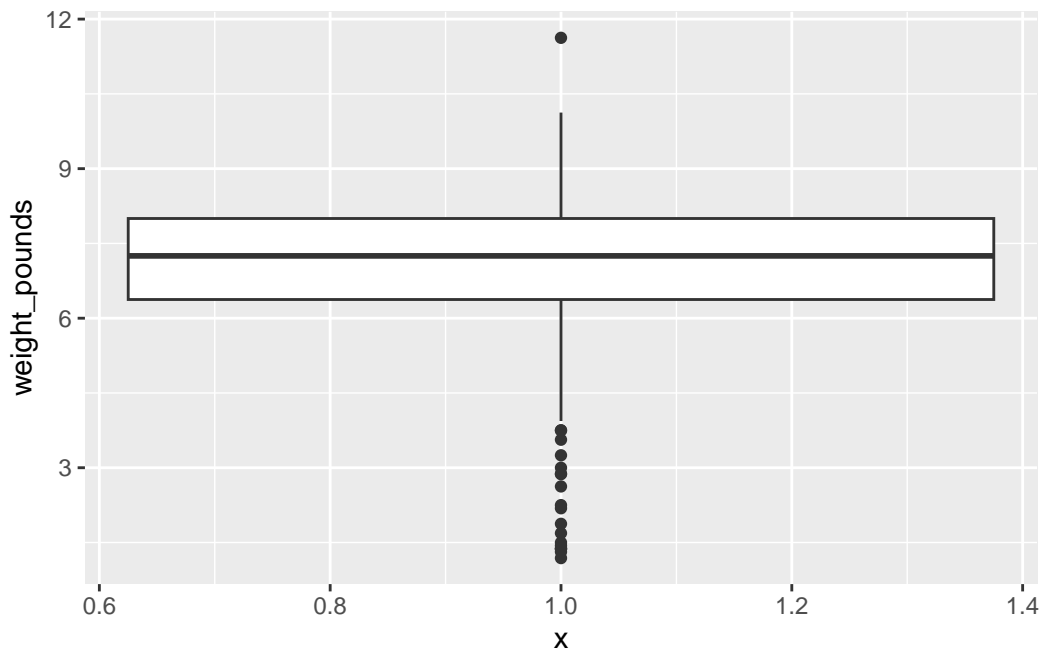
It is mostly normal-looking, but I am suspicious about those *very* low birth weights, the ones below about 4 pounds. There are a few too many of those, as I see it.

If you think this is approximately normal, you need to make some comment along the lines of “the shape is approximately symmetric with no outliers”. I think my first answer is better,

but this answer is worth something, since it is a not completely unreasonable interpretation of the histogram.

I have been making the distinction between a histogram (for one quantitative variable) and side-by-side boxplots (for one quantitative variable divided into groups by one categorical variable). When you learned the boxplot, you probably learned it in the context of one quantitative variable. You can draw a boxplot for that, too, but the `ggplot` boxplot has an `x` as well as a `y`. What you do to make a single boxplot is to set the `x` equal 1, which produces a weird `x`-axis (that you ignore):

```
ggplot(bw, aes(x = 1, y = weight_pounds)) + geom_boxplot()
```



The high weight is actually an outlier, but look at all those outliers at the bottom!<sup>3</sup>

I think the reason for those extra very low values is that they are the premature births (that can result in *very* small babies). Which leads to the additional question coming up later.

■

---

<sup>3</sup>When Tukey, a name we will see again, invented the boxplot in the 1950s, 500 observations would have been considered a big data set. He designed the boxplot to produce a sensible number of outliers for the typical size of data set of his day, but a boxplot of a large data set tends to have a lot of outliers that are probably not really outliers at all.

### 3.9 More about the NC births

This is an exploration of some extra issues around the North Carolina births data set.

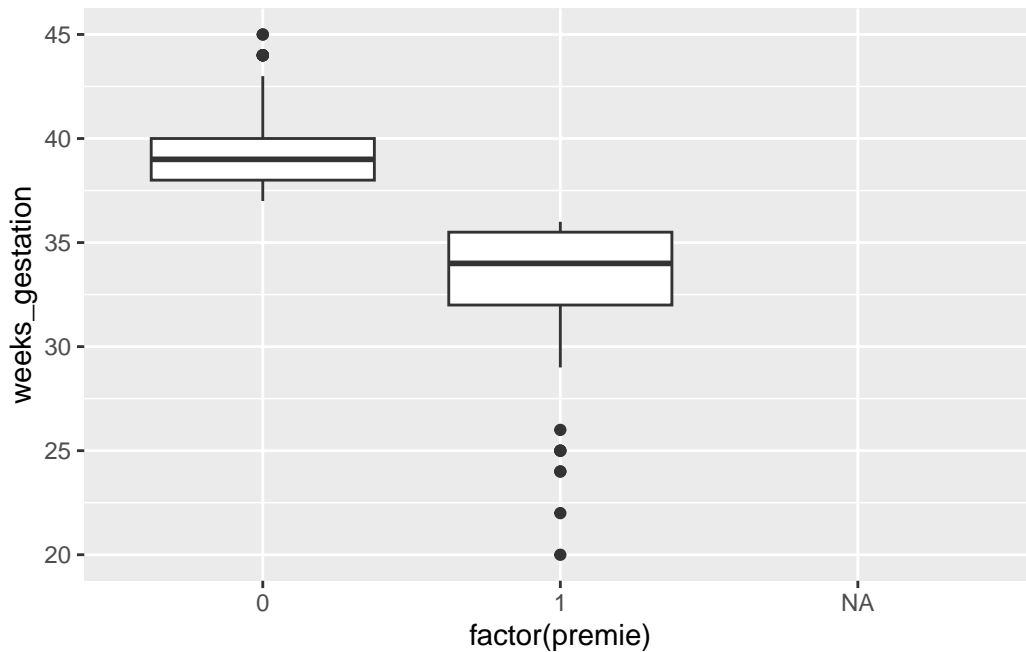
- (a) How short does a pregnancy have to be, for the birth to be classified as “premature”? Deduce this from the data, by drawing a suitable graph or otherwise.

Solution

To figure it out from the data, we can see how `weeks_gestation` depends on `premie`. Some possibilities are boxplots or a scatterplot. Either of the first two graphs would get full credit (for the graphing part: you still have to do the explanation) if this were being marked:

```
ggplot(bw,aes(x=factor(premie), y=weeks_gestation)) + geom_boxplot()
```

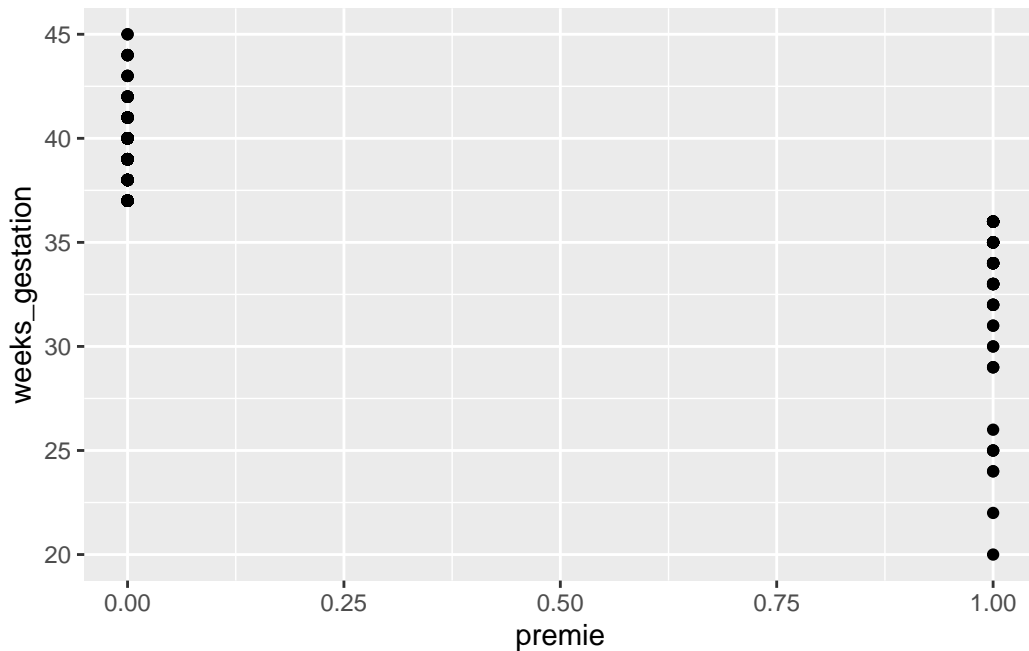
Warning: Removed 1 rows containing non-finite values (``stat_boxplot()``).



The warning is because the prematurity of one of the babies is not known. Or

```
ggplot(bw,aes(x=premie, y=weeks_gestation)) + geom_point()
```

Warning: Removed 1 rows containing missing values (``geom_point()``).



The same warning again, for the same reason.

Notice how the graphs are similar in syntax, because the what-to-plot is the same (apart from the **factor** thing) and we just make a small change in how-to-plot-it. In the boxplot, the thing on the *x*-scale needs to be categorical, and **premie** is actually a number, so we'd better make it into a **factor**, which is R's version of a categorical variable. **premie** is actually a categorical variable ("premature" or "not premature") masquerading as a quantitative one (1 or 0). It is an "indicator variable", if you're familiar with that term.

It looks as if the breakpoint is 37 weeks: a pregnancy at least that long is considered normal, but a shorter one ends with a premature birth. Both plots show the same thing: the **premie=1** births all go with short pregnancies, shorter than 37 weeks. This is completely clear cut.

Another way to attack this is to use **summarize**, finding the max and min:

```
bw %>% summarize( n=n(),
  min=min(weeks_gestation),
  max=max(weeks_gestation))
```

```
# A tibble: 1 x 3
      n   min   max
<int> <dbl> <dbl>
1   500   NA   NA
```



only this is for *all* the babies, premature or not.<sup>4</sup> So we want it by prematurity, which means a `group_by` first:

```
bw %>% group_by(premie) %>%  
  summarize( n=n(),  
    min=min(weeks_gestation),  
    max=max(weeks_gestation))
```

```
# A tibble: 3 x 4  
  premie      n    min    max  
  <dbl> <int> <dbl> <dbl>  
1      0   424     37     45  
2      1    75     20     36  
3     NA     1     NA     NA
```

`group_by` with a number works, even though using the number in `premie` in a boxplot didn't. `group_by` just uses the distinct values, whether they are numbers, text or factor levels.

Any of these graphs or summaries will help you answer the question, in the same way. The ultimate issue here is “something that will get the job done”: it doesn't matter so much what.

Extra: In R, NA means “missing”. When you try to compute something containing a missing value, the answer is usually missing (since you don't know what the missing value is). That's why the first `summarize` gave us missing values: there was one missing weeks of gestation in with all the ones for which we had values, so the max and min had to be missing as well. In the second `summarize`, the one by whether a baby was born prematurely or not, we learn a bit more about that missing `premie`: evidently its weeks of gestation was missing as well, since the min and max of that were missing.<sup>5</sup>

Here's that baby. I'm doing a bit of fiddling to show all the columns (as rows, since there's only one actual row). Don't worry about the second line of code below; we will investigate that later in the course. Its job here is to show the values nicely:

```
bw %>%  
  filter(is.na(premie)) %>%  
  pivot_longer(everything(), names_to="name", values_to="value")
```

---

<sup>4</sup>I explain the missing values below.

<sup>5</sup>If there had been a weeks of gestation, we could have figured out whether it was premature or not, according to whether the weeks of gestation was less than 37.

```
# A tibble: 10 x 2
  name          value
  <chr>        <dbl>
1 father_age    33
2 mother_age    32
3 weeks_gestation NA
4 pre_natal_visits 9
5 marital_status 1
6 mother_weight_gained 25
7 low_birthweight 0
8 weight_pounds  7.19
9 premie        NA
10 few_visits    0
```

The *only* thing that was missing was its weeks of gestation, but that prevented anyone from figuring out whether it was premature or not.



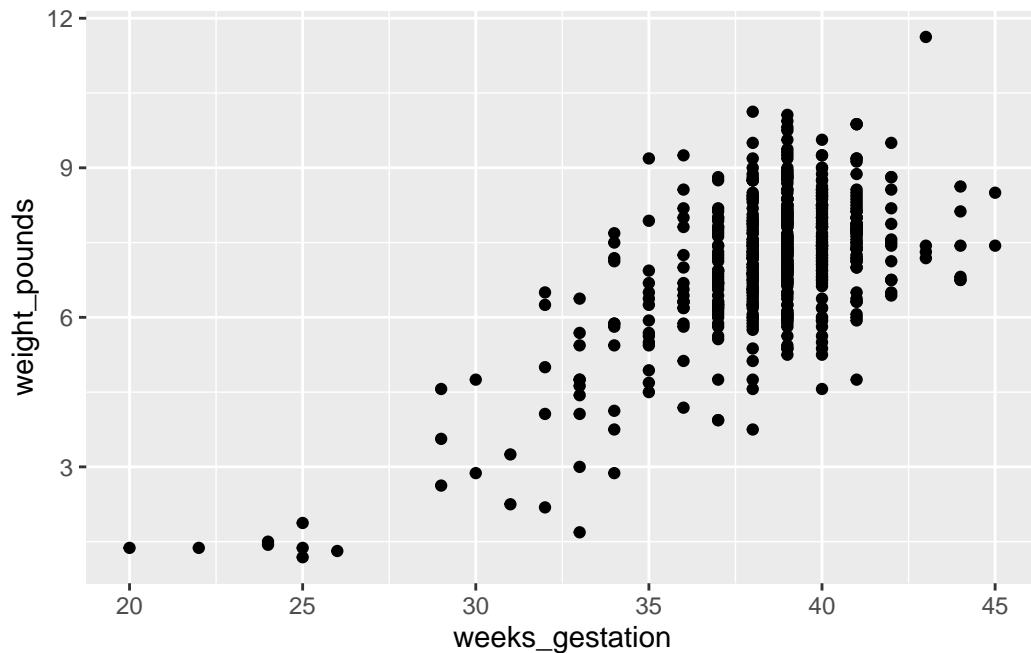
- (b) Explore the relationship between birth weight and length of pregnancy (“gestation”) using a suitable graph. What do you see?

Solution

This needs to be a scatterplot because these are both quantitative variables:

```
ggplot(bw,aes(x=weeks_gestation, y=weight_pounds)) + geom_point()
```

Warning: Removed 1 rows containing missing values (`geom\_point()`).

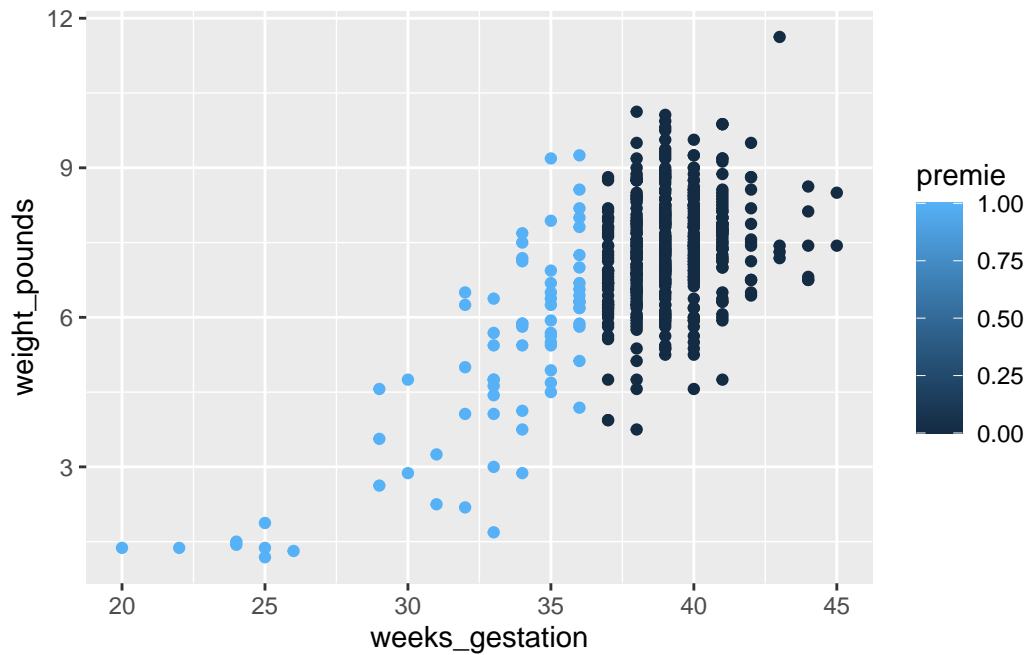


You see a rather clear upward trend. Those very underweight babies came from very short pregnancies, but the vast majority of pregnancies were of more or less normal length (40 weeks is normal) and resulted in babies of more or less normal birth weight.

Extra: I want to illustrate something else: how about *colouring* the births that were premature? Piece of cake with `ggplot`:

```
ggplot(bw,aes(x=weeks_gestation, y=weight_pounds, colour = premie)) +  
  geom_point()
```

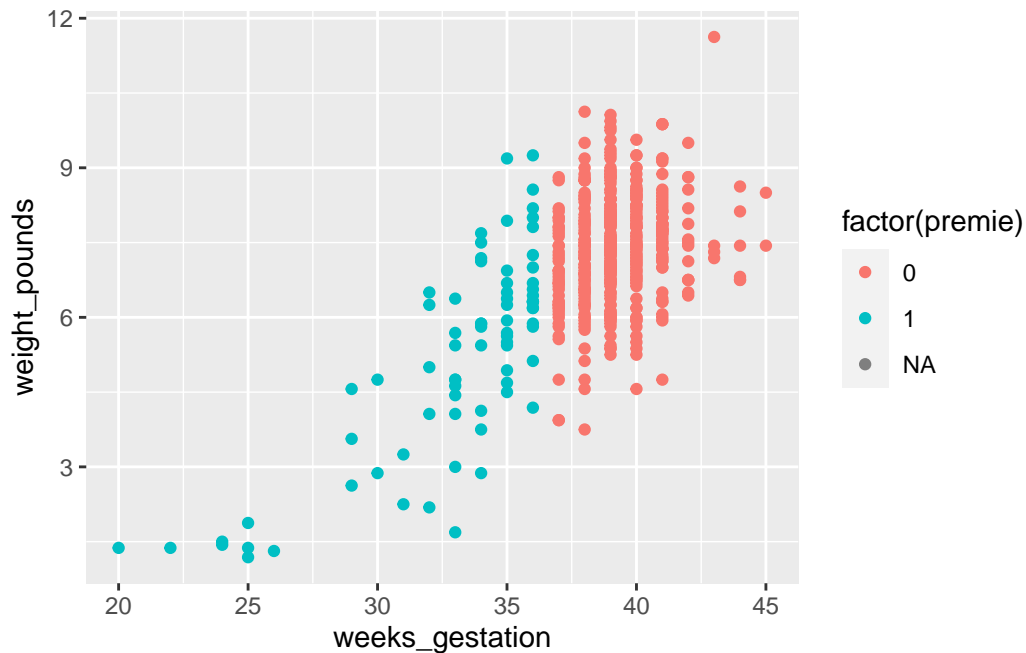
Warning: Removed 1 rows containing missing values (`geom\_point()`).



That was rather silly because `ggplot` treated prematureness as a *continuous* variable, and plotted the values on a dark blue-light blue scale. This is the same issue as on the boxplot above, and has the same solution:

```
ggplot(bw,aes(x=weeks_gestation, y=weight_pounds,
              colour = factor(premie))) + geom_point()
```

Warning: Removed 1 rows containing missing values (``geom_point()``).



Better.

With the normal-length pregnancies (red), there seems to be no relationship between length of pregnancy and birth weight, just a random variation. But with the premature births, a shorter pregnancy typically goes with a *lower* birth weight. This would be why the birth weights for the premature births were more variable.

■

- (c) Do a web search to find the standard (North American) definition of a premature birth. Does that correspond to what you saw in the data? Cite the website you used, for example by saying “according to URL, ...”, with URL replaced by the address of the website you found.

Solution

The website <http://www.mayoclinic.org/diseases-conditions/premature-birth/basics/definition/con-20020050> says that “a premature birth is one that occurs before the start of the 37th week of pregnancy”, which is exactly what we found. (Note that I am citing the webpage on which I found this, and I even made it into a link so that you can check it.) The Mayo Clinic is a famous hospital system with locations in several US states, so I think we can trust what its website says.

■

### 3.10 Nenana, Alaska

Nenana, Alaska, is about 50 miles west of Fairbanks. Every spring, there is a contest in Nenana. A wooden tripod is placed on the frozen river, and people try to guess the exact minute when the ice melts enough for the tripod to fall through the ice. The contest started in 1917 as an amusement for railway workers, and has taken place every year since. Now, hundreds of thousands of people enter their guesses on the Internet and the prize for the winner can be as much as \$300,000.

Because so much money is at stake, and because the exact same tripod is placed at the exact same spot on the ice every year, the data are consistent and accurate. The data are in [link](#).

- (a) Read the data into R. Note that the values are separated by *tabs* rather than spaces, so you'll need an appropriate `read_` to read it in.

Solution

These are “tab-separated values”, so `read_tsv` is the thing, as for the Australian athletes:

```
myurl <- "http://ritsokiguess.site/datafiles/nenana.txt"
nenana <- read_tsv(myurl)
```

```
Rows: 87 Columns: 3
```

```
-- Column specification -----
```

```
Delimiter: "\t"
```

```
chr (1): Date&Time
```

```
dbl (2): Year, JulianDate
```

```
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.
```

Use whatever name you like for the data frame. One that is different from any of the column headers is smart; then it is clear whether you mean the whole data frame or one of its columns. `ice` or `melt` or anything like that would also be good.

I haven't asked you to display or check the data (that's coming up), but if you look at it and find that it didn't work, you'll know to come back and try this part again. R usually gets it right or gives you an error.

If you look at the data, they do appear to be separated by spaces, but the text version of the date and time *also* have spaces in them, so things might go astray if you try and read the values in without recognizing that the actual separator is a tab:

```
x <- read_delim(myurl, " ")
```

Warning: One or more parsing issues, call ``problems()`` on your data frame for details, e.g.:

```
dat <- vroom(...)
problems(dat)
```

Rows: 87 Columns: 1

-- Column specification -----

Delimiter: " "

chr (1): Year JulianDate Date&Time

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.

Ouch! A hint as to what went wrong comes from looking at the read-in data frame:

```
x
```

```
# A tibble: 87 x 1
```

```
  `Year\tJulianDate\tDate&Time`
```

```
  <chr>
```

```
1 "1917\t120.4795\tApril 30 at 11:30 AM"
2 "1918\t131.3983\tMay 11 at 9:33 AM"
3 "1919\t123.6066\tMay 3 at 2:33 PM"
4 "1920\t132.4490\tMay 11 at 10:46 AM"
5 "1921\t131.2795\tMay 11 at 6:42 AM"
6 "1922\t132.5559\tMay 12 at 1:20 PM"
7 "1923\t129.0837\tMay 9 at 2:00 AM"
8 "1924\t132.6323\tMay 11 at 3:10 PM"
9 "1925\t127.7726\tMay 7 at 6:32 PM"
10 "1926\t116.6691\tApril 26 at 4:03 PM"
```

```
# i 77 more rows
```

Those `t` symbols mean “tab character”, which is our hint that the values were separated by tabs rather than spaces.

More detail (if you can bear to see it) is here:

```
problems(x)
```

```
# A tibble: 87 x 5
```

```
  row col expected actual file
```

```

      <int> <int> <chr>      <chr>      <chr>
1         2      5 1 columns 5 columns ""
2         3      5 1 columns 5 columns ""
3         4      5 1 columns 5 columns ""
4         5      5 1 columns 5 columns ""
5         6      5 1 columns 5 columns ""
6         7      5 1 columns 5 columns ""
7         8      5 1 columns 5 columns ""
8         9      5 1 columns 5 columns ""
9        10      5 1 columns 5 columns ""
10       11      5 1 columns 5 columns ""
# i 77 more rows

```

The first line of the data file (with the variable names in it) had no spaces, only tabs, so `read_delim` thinks there is *one* column with a very long name, but in the actual data, there are *five* space-separated columns. The text date-times are of the form “April 30 at 11:30 AM”, which, if you think it’s all separated by spaces, is actually 5 things: April, 30, at and so on. These are the only things that are separated by spaces, so, from that point of view, there are five columns.

■

- (b) Find a way of displaying how many rows and columns your data frame has, and some of the values. Describe the first and last of the variables that you appear to have.

Solution

The easiest is just to display the tibble: `:: { .cell }`

nenana

```

# A tibble: 87 x 3
   Year JulianDate `Date&Time`
  <dbl>      <dbl> <chr>
1  1917      120. April 30 at 11:30 AM
2  1918      131. May 11 at 9:33 AM
3  1919      124. May 3 at 2:33 PM
4  1920      132. May 11 at 10:46 AM
5  1921      131. May 11 at 6:42 AM
6  1922      133. May 12 at 1:20 PM
7  1923      129. May 9 at 2:00 AM
8  1924      133. May 11 at 3:10 PM
9  1925      128. May 7 at 6:32 PM

```



```
10 1926          117. April 26 at 4:03 PM
# i 77 more rows
```

```
...
```

Alternatively, you can take a `glimpse` of it:

```
glimpse(nenana)
```

```
Rows: 87
```

```
Columns: 3
```

```
$ Year      <dbl> 1917, 1918, 1919, 1920, 1921, 1922, 1923, 1924, 1925, 1926~
$ JulianDate <dbl> 120.4795, 131.3983, 123.6066, 132.4490, 131.2795, 132.5559~
$ `Date&Time` <chr> "April 30 at 11:30 AM", "May 11 at 9:33 AM", "May 3 at 2:3~
```

There are 87 years, and 3 columns (variables). The first column is year, and the last column is the date and time that the tripod fell into the river, written as a piece of text. I explain the second column in a moment.

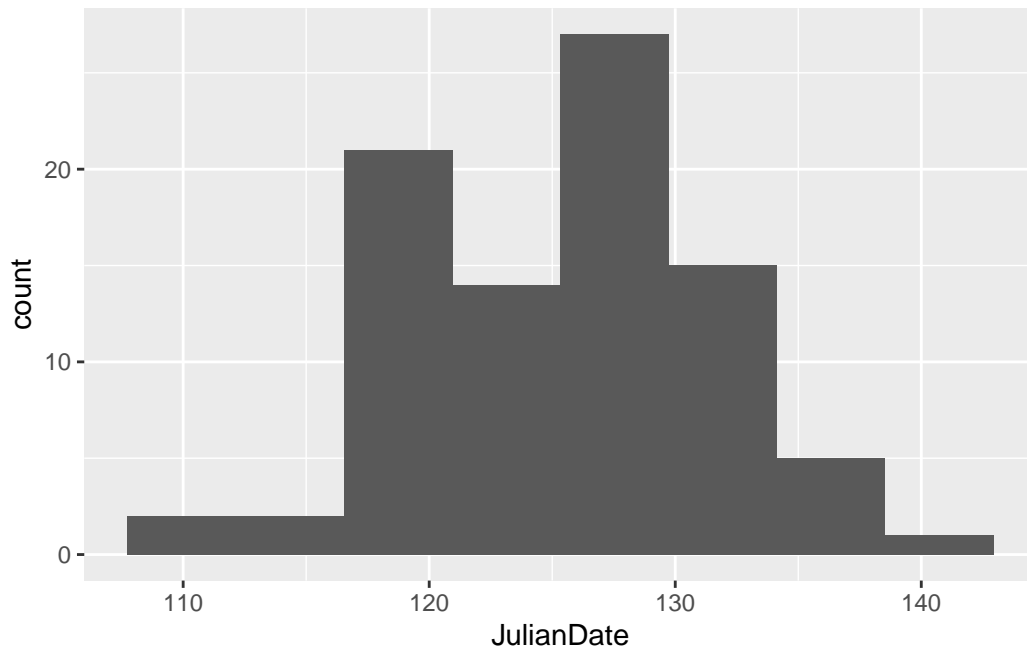


- (c) Dates and times are awkward to handle with software. (We see more ways later in the course.) The column `JulianDate` expresses the time that the tripod fell through the ice as a fractional number of days since December 31. This enables the time (as a fraction of the way through the day) to be recorded as well, the whole thing being an ordinary number. Make a histogram of the Julian dates. Comment briefly on its shape.

Solution

With a `ggplot` histogram, we need a number of bins first. I can do Sturges' rule in my head: the next power of 2 up from 87 (our  $n$ ) is 128, which is  $2^7$ , so the base 2 log of 87 rounds up to 7. That plus one is 8, so we need 8 bins. For you, any not-insane number of bins will do, or any not-insane bin width, if you want to go that way: `::: {.cell}`

```
ggplot(nenana, aes(x = JulianDate)) + geom_histogram(bins = 8)
```



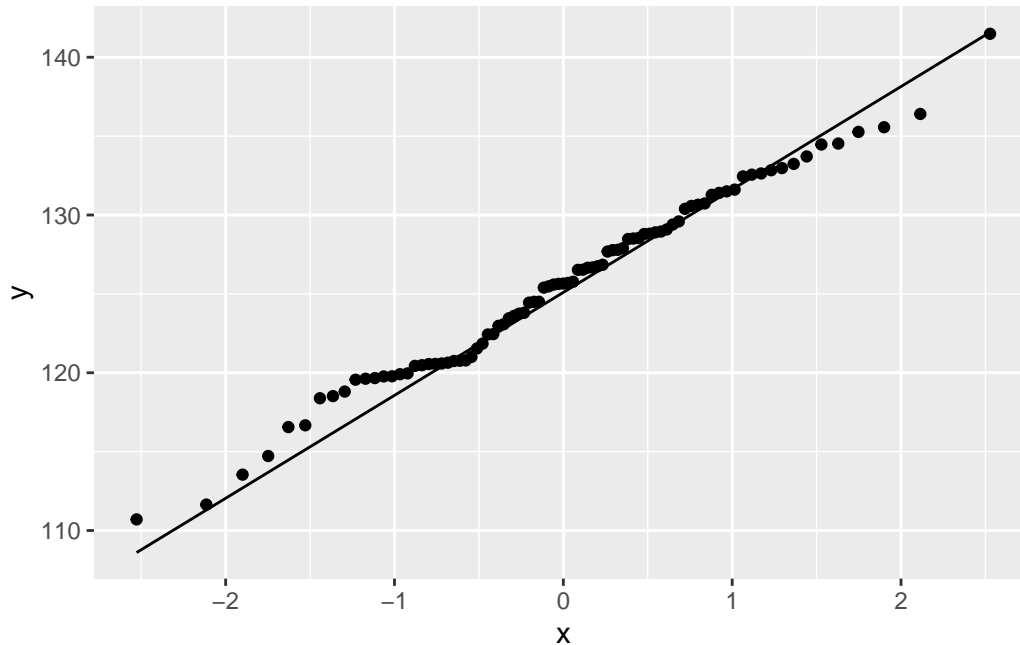
...

Note that you need to type `JulianDate` exactly as it appears, capital letters and all. R is case-sensitive.

This histogram looks more or less symmetric (and, indeed, normal). I really don't think you can justify an answer other than "symmetric" here. (Or "approximately normal": that's good too.) If your histogram is different, say so. I think that "hole" in the middle is not especially important.

We haven't done normal quantile plots yet, but looking ahead:

```
ggplot(nenana, aes(sample = JulianDate)) + stat_qq() + stat_qq_line()
```



That hugs the line pretty well, so I would call it close to normally-distributed. It bulges away from the line because there are more values just below 120 than you would expect for a normal. This corresponds to the histogram bar centred just below 120 being taller than you would have expected.<sup>6</sup>

Extra: looking *way* ahead (to almost the end of the R stuff), this is how you handle the dates and times:

```
library(lubridate)
nenana %>%
  mutate(longdt = str_c(Year, " ", `Date&Time`)) %>%
  mutate(datetime = ymd_hm(longdt, tz = "America/Anchorage"))
```

# A tibble: 87 x 5

	Year	JulianDate	`Date&Time`	longdt	datetime
	<dbl>	<dbl>	<chr>	<chr>	<dtm>
1	1917	120.	April 30 at 11:30 AM	1917 April 30 at 1~	1917-04-30 11:30:00
2	1918	131.	May 11 at 9:33 AM	1918 May 11 at 9:3~	1918-05-11 09:33:00
3	1919	124.	May 3 at 2:33 PM	1919 May 3 at 2:33~	1919-05-03 14:33:00
4	1920	132.	May 11 at 10:46 AM	1920 May 11 at 10:~	1920-05-11 10:46:00
5	1921	131.	May 11 at 6:42 AM	1921 May 11 at 6:4~	1921-05-11 06:42:00

<sup>6</sup>That is to say, the principal deviation from normality is not the hole on the histogram, the bar centred around 123 being too short, but that the bar centred just below 120 is too *tall*.

```

6 1922      133. May 12 at 1:20 PM    1922 May 12 at 1:2~ 1922-05-12 13:20:00
7 1923      129. May 9 at 2:00 AM     1923 May 9 at 2:00~ 1923-05-09 02:00:00
8 1924      133. May 11 at 3:10 PM    1924 May 11 at 3:1~ 1924-05-11 15:10:00
9 1925      128. May 7 at 6:32 PM     1925 May 7 at 6:32~ 1925-05-07 18:32:00
10 1926     117. April 26 at 4:03 PM   1926 April 26 at 4~ 1926-04-26 16:03:00
# i 77 more rows

```

I am not doing any further analysis with these, so just displaying them is good.

I have to do a preliminary step to get the date-times *with* their year in one place. `str_c` glues pieces of text together: in this case, the year, a space, and then the rest of the `Date&Time`. I stored this in `longdt`. The second `mutate` is the business end of it: `ymd_hm` takes a piece of text containing a year, month (by name or number), day, hours, minutes *in that order*, and extracts those things from it, storing the whole thing as an R date-time. Note that the AM/PM was handled properly. The benefit of doing that is we can extract anything from the dates, such as the month or day of week, or take differences between the dates. Or even check that the Julian dates were calculated correctly (the `lubridate` function is called `yday` for “day of year”):

```

nenana %>%
  mutate(longdt = str_c(Year, " ", `Date&Time`)) %>%
  mutate(datetime = ymd_hm(longdt, tz = "America/Anchorage")) %>%
  mutate(jd = yday(datetime)) ->
nenana2
nenana2 %>% select(JulianDate, jd, datetime)

```

```

# A tibble: 87 x 3
  JulianDate   jd datetime
      <dbl> <dbl> <dtm>
1      120.   120 1917-04-30 11:30:00
2      131.   131 1918-05-11 09:33:00
3      124.   123 1919-05-03 14:33:00
4      132.   132 1920-05-11 10:46:00
5      131.   131 1921-05-11 06:42:00
6      133.   132 1922-05-12 13:20:00
7      129.   129 1923-05-09 02:00:00
8      133.   132 1924-05-11 15:10:00
9      128.   127 1925-05-07 18:32:00
10     117.   116 1926-04-26 16:03:00
# i 77 more rows

```

The Julian days as calculated are the same. Note that these are not rounded; the Julian day begins at midnight and lasts until the next midnight. Thus Julian day 132 is May 12 (in a

non-leap year like 1922) and the reason that the Julian date given in the file for that year would round to 133 is that it is after noon (1:20pm as you see).

■

- (d) Plot `JulianDate` against `Year` on a scatterplot. What recent trends, if any, do you see? Comment briefly.

Solution

`geom_point`:

```
ggplot(nenana, aes(x = Year, y = JulianDate)) + geom_point()
```

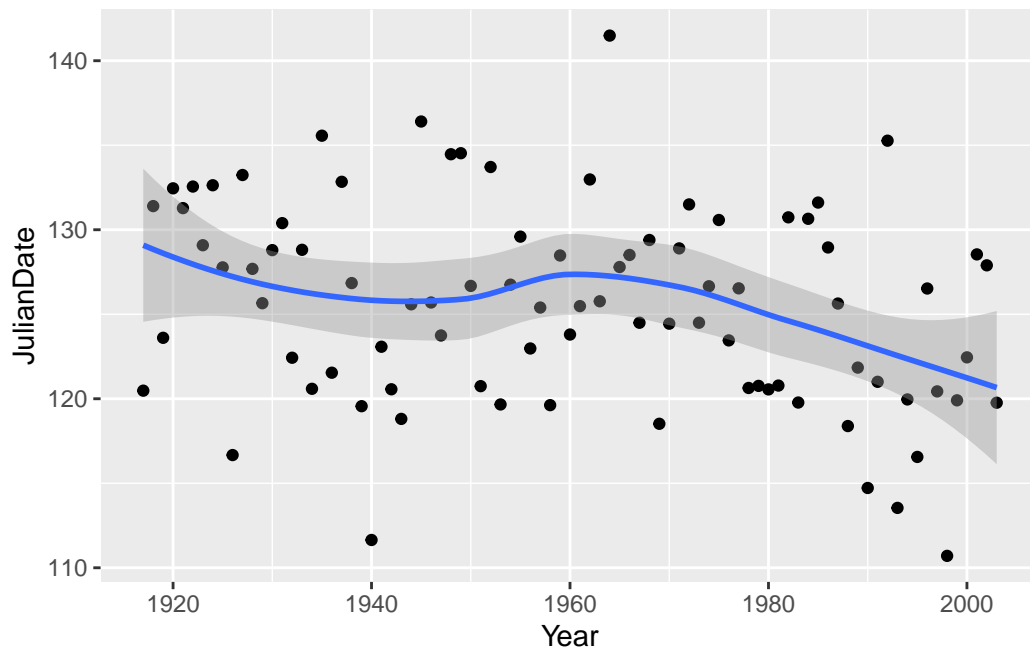


This is actually a small-but-real downward trend, especially since about 1960, but the large amount of variability makes it hard to see, so I'm good with either “no trend” or “weak downward trend” or anything roughly like that. There is definitely not much trend before 1960, but most of the really early break-ups (less than about 118) have been since about 1990.

You can even add to the `ggplot`, by putting a smooth trend on it:

```
ggplot(nenana, aes(x = Year, y = JulianDate)) + geom_point() + geom_smooth()
```

```
`geom_smooth()` using method = 'loess' and formula = 'y ~ x'
```



This is R's version of a trend that is not constrained to be linear (so that it “lets the data speak for itself”).

Now there is something obvious to see: after about 1960, there is a clear downward trend: the ice is breaking up earlier on average every year. Even though there is a lot of variability, the overall trend, viewed this way, is clear.

What does this mean, in practice? This notion of the ice melting earlier than it used to is consistent all over the Arctic, and is one more indication of climate change. Precisely, it is an indication that climate change is happening, but we would have to delve further to make any statements about the *cause* of that climate change.



### 3.11 Computerized accounting

Beginning accounting students need to learn to learn to audit in a computerized environment. A sample of beginning accounting students took each of two tests: the Computer Attitude Scale (CAS) and the Computer Anxiety Rating Scale (CARS). A higher score in each indicates greater anxiety around computers. The test scores are scaled to be between 0 and 5. Also noted

was each student's gender. The data are in <http://ritsokiguess.site/datafiles/compatt.txt>. The data values are separated by spaces.

(a) Read the data into R. Do you have what you expected? Explain briefly.

Solution

Read in and display the data. This, I think, is the easiest way.

```
my_url <- "https://raw.githubusercontent.com/nxskok/datafiles/master/compatt.txt"
anxiety=read_delim(my_url," ")
```

```
Rows: 35 Columns: 3
```

```
-- Column specification -----
```

```
Delimiter: " "
```

```
chr (1): gender
```

```
dbl (2): CAS, CARS
```

```
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.
```

```
anxiety
```

```
# A tibble: 35 x 3
```

	gender	CAS	CARS
	<chr>	<dbl>	<dbl>
1	female	2.85	2.9
2	male	2.6	2.32
3	female	2.2	1
4	male	2.65	2.58
5	male	2.6	2.58
6	male	3.2	3.05
7	male	3.65	3.74
8	female	2.55	1.9
9	male	3.15	3.32
10	male	2.8	2.74

```
# i 25 more rows
```

There is a total of 35 students with a CAS score, a CARS score and a gender recorded for each. This is in line with what I was expecting. (You can also note that the genders appear to be a mixture of males and females.)

■

(b) How many males and females were there in the sample?

Solution

Most easily count: ::: {.cell}

```
anxiety %>% count(gender)
```

```
# A tibble: 2 x 2
  gender      n
  <chr>  <int>
1 female    15
2 male     20
```

:::

This also works (and is therefore good):

```
anxiety %>% group_by(gender) %>% summarize(count=n())
```

```
# A tibble: 2 x 2
  gender count
  <chr>  <int>
1 female    15
2 male     20
```

I want you to use R to do the counting (that is, don't just go through the whole data set and count the males and females yourself). This is because you might have thousands of data values and you need to learn how to get R to count them for you.

15 females and 20 males, *which you should say*. I made a point of *not* saying that it is enough to get the output with the answers on it, so you need to tell me what the answer is.



(c) Do the CAS scores tend to be higher for females or for males? Draw a suitable graph to help you decide, and come to a conclusion.

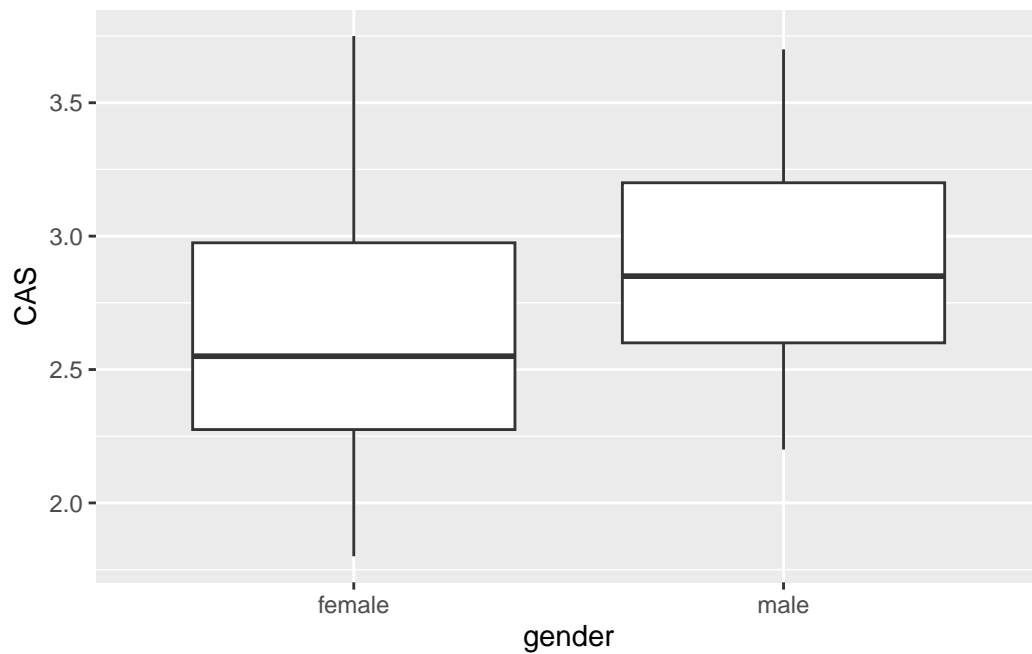
Solution

Gender is categorical and CAS score is quantitative, so a boxplot would appear to be the thing:

::: {.cell}



```
ggplot(anxiety,aes(x=gender,y=CAS))+geom_boxplot()
```

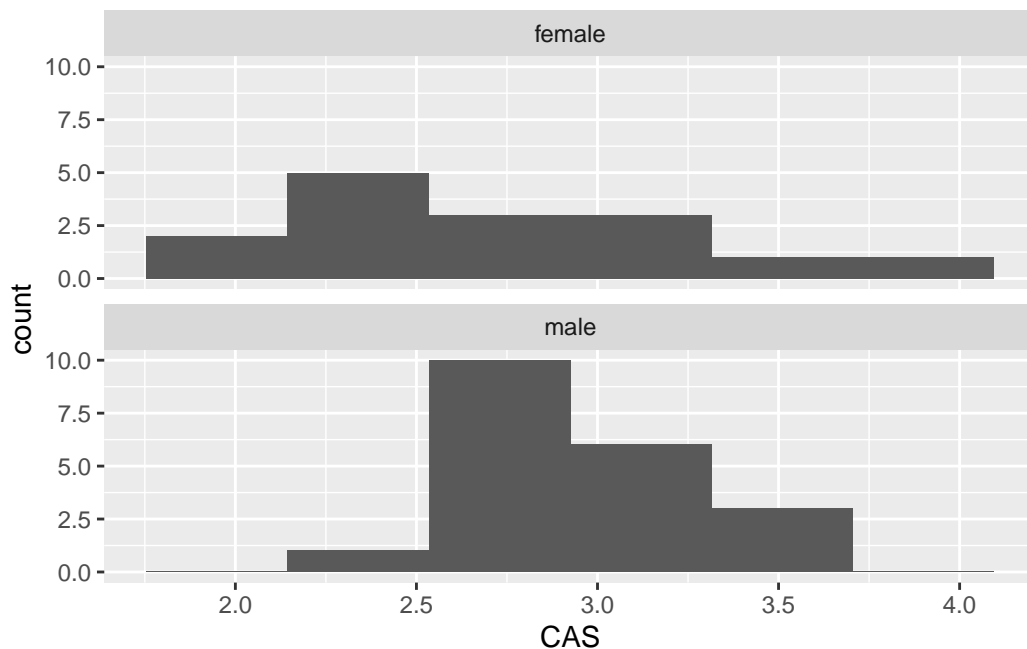


∴

The median for males is slightly higher, so male accountants are more anxious around computers than female accountants are.

If you wish, side-by-side (or, better, above-and-below) histograms would also work:

```
ggplot(anxiety,aes(x=CAS))+geom_histogram(bins=6)+  
facet_wrap(~gender,ncol=1)
```



If you go this way, you have to make a call about where the centres of the histograms are. I guess the male one is slightly further to the right, but it's not so easy to tell. (Make a call.)

■

- (d) Find the median CAS scores for each gender. Does this support what you saw on your plot? Explain briefly.

Solution

Group-by and summarize: `::: {.cell}`

```
anxiety %>% group_by(gender) %>% summarize(med=median(CAS))
```

```
# A tibble: 2 x 2
```

```
  gender  med
  <chr> <dbl>
1 female 2.55
2 male   2.85
```

```
:::
```

The median is a bit higher for males, which is what I got on my boxplot (and is apparently the same thing as is on the histograms, but it's harder to be sure there).



- (e) Find the mean and standard deviation of both CAS and CARS scores (for all the students combined, ie. not separated by gender) *without* naming those columns explicitly.

Solution

Without naming them explicitly means using some other way to pick them out of the data frame, **summarize** with **across**.

The basic **across** comes from asking yourself what the *names* of those columns have in common: they start with C and the gender column doesn't:

```
anxiety %>% summarize(across(starts_with("C"), list(m = ~mean(.), s = ~sd(.))))
```

```
# A tibble: 1 x 4
  CAS_m CAS_s CARS_m CARS_s
  <dbl> <dbl> <dbl> <dbl>
1  2.82 0.484  2.77  0.671
```

Another way is to ask what *property* these two columns have in common: they are the only two numeric (quantitative) columns. This means using an **across** with a **where** inside it, thus:

```
anxiety %>% summarize(across(where(is.numeric), list(m = ~mean(.), s = ~sd(.))))
```

```
# A tibble: 1 x 4
  CAS_m CAS_s CARS_m CARS_s
  <dbl> <dbl> <dbl> <dbl>
1  2.82 0.484  2.77  0.671
```

Read the first one as “across all the columns whose names start with S, find the mean and SD of them.” The second one is a little clunkier: “across all the columns for which **is.numeric** is true, find the mean and SD of them”. A shorter way for the second one is “across all the numeric (quantitative) columns, find their mean and SD”, but then you have to remember exactly how to code that. The reason for the **list** is that we are calculating two statistics for each column that we find. I am using a “named list” so that the mean gets labelled with an **m** on the end of the column name, and the SD gets an **s** on the end.

Either of these is good, or anything equivalent (like noting that the two anxiety scales both **ends\_with S**):

```
anxiety %>% summarize(across(ends_with("S"), list(m = ~mean(.), s = ~sd(.))))
```

```
# A tibble: 1 x 4
  CAS_m CAS_s CARS_m CARS_s
  <dbl> <dbl> <dbl> <dbl>
1  2.82 0.484   2.77 0.671
```

Because I didn't say otherwise, you should tell me what the means and SDs are, rounding off suitably: the CAS scores have mean 2.82 and SD 0.48, and the CARS scores have mean 2.77 and SD 0.67.

Yet another way to do it is to select the columns you want first (which you can do by number so as not to name them), and then find the mean and SD of all of them:

```
anxiety %>% select(2:3) %>%
  summarize(across(everything(), list(m = ~mean(.), s = ~sd(.))))
```

```
# A tibble: 1 x 4
  CAS_m CAS_s CARS_m CARS_s
  <dbl> <dbl> <dbl> <dbl>
1  2.82 0.484   2.77 0.671
```

This doesn't work:

```
summary(anxiety)
```

gender	CAS	CARS
Length:35	Min. :1.800	Min. :1.000
Class :character	1st Qu.:2.575	1st Qu.:2.445
Mode :character	Median :2.800	Median :2.790
	Mean :2.816	Mean :2.771
	3rd Qu.:3.150	3rd Qu.:3.290
	Max. :3.750	Max. :4.000

because, although it gets the means, it does not get the standard deviations. (I added the SD to the original question to make you find a way other than this.)

In summary, find a way to get those answers without naming those columns in your code, and I'm good.

In case you were wondering about how to do this separately by gender, well, put the `group\_by` in like you did before:

```
anxiety %>% group_by(gender) %>%
  summarize(across(where(is.numeric), list(m = ~mean(.), s = ~sd(.))))
```

```
# A tibble: 2 x 5
  gender CAS_m CAS_s CARS_m CARS_s
  <chr>   <dbl> <dbl>   <dbl>   <dbl>
1 female  2.64 0.554    2.51    0.773
2 male    2.94 0.390    2.96    0.525
```

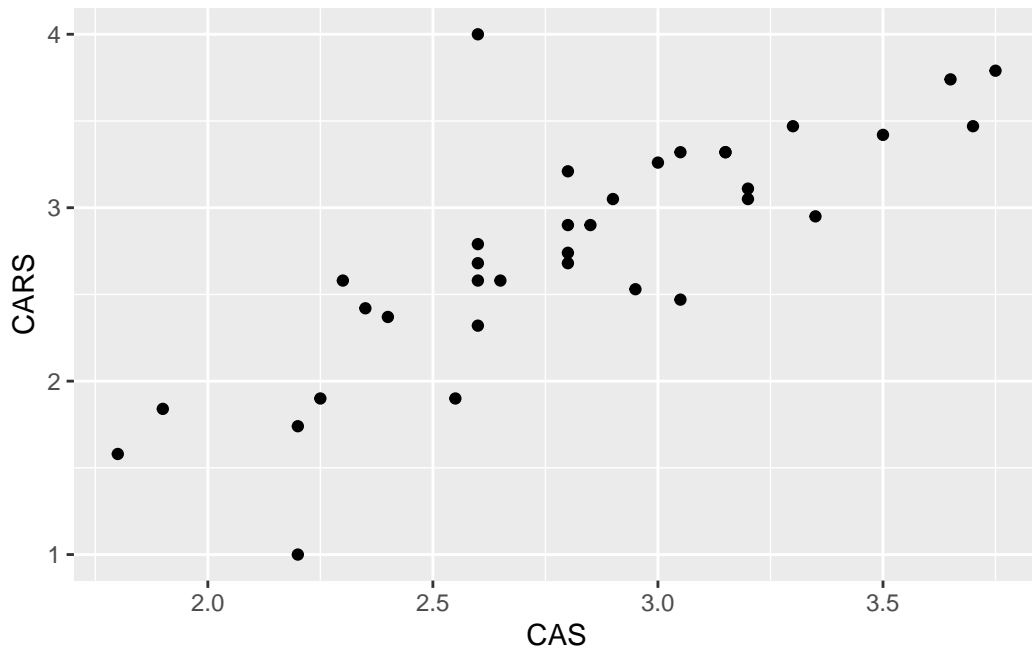
or

```
anxiety %>% group_by(gender) %>%
  summarize(across(starts_with("C"), list(m = ~mean(.), s = ~sd(.))))
```

```
# A tibble: 2 x 5
  gender CAS_m CAS_s CARS_m CARS_s
  <chr>   <dbl> <dbl>   <dbl>   <dbl>
1 female  2.64 0.554    2.51    0.773
2 male    2.94 0.390    2.96    0.525
```

The male means are slightly higher on both tests, but the male standard deviations are a little smaller. You might be wondering whether the test scores are related. They are both quantitative, so the obvious way to find out is a scatterplot:

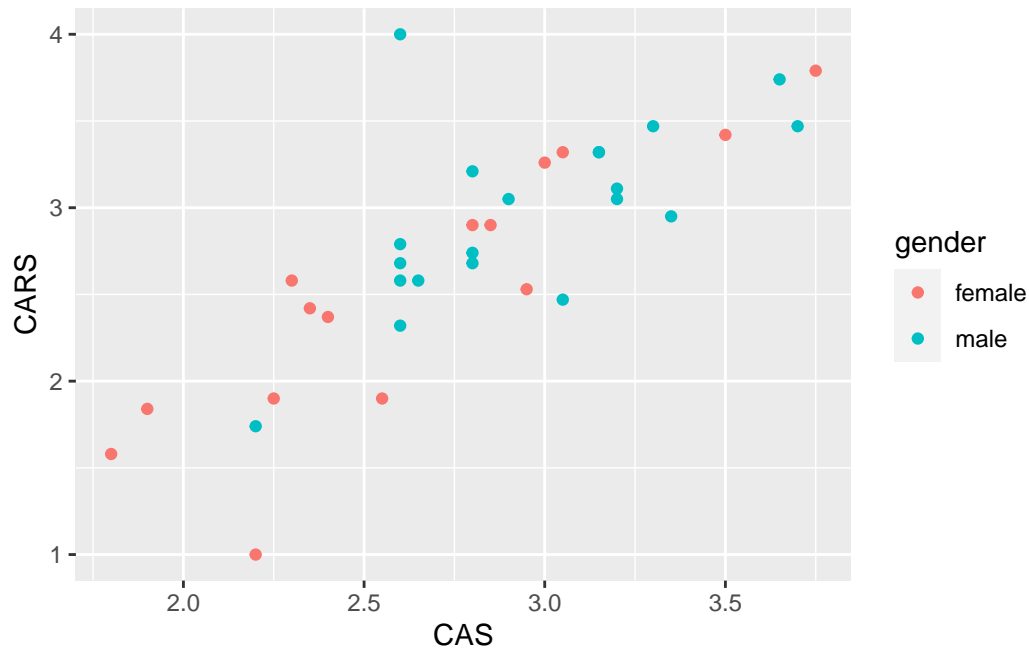
```
ggplot(anxiety, aes(x=CAS, y=CARS)) + geom_point()
```



The two variables can be on either axis, since there is no obvious response or explanatory variable. A higher score on one scale goes with a higher score on the other, suggesting that the two scales are measuring the same thing.

This plot mixes up the males and females, so you might like to distinguish them, which goes like this:

```
ggplot(anxiety,aes(x=CAS,y=CARS,colour=gender))+geom_point()
```



There is a slight (but only slight) tendency for the males to be up and to the right, and for the females to be down and to the left. This is about what you would expect, given that the male means are slightly bigger on both scores, but the difference in means is not that big compared to the SD.

■

### 3.12 Test scores in two classes

Open R Studio. Create a new Text File by selecting File, New File and Text File. You should see a new empty, untitled window appear at the top left. In that window, type or copy the data below (which are scores on a test for students in two different classes):

```
class score
ken 78
ken 62
ken 59
ken 69
ken 81
thomas 83
thomas 77
```

```
thomas 63
thomas 61
thomas 79
thomas 72
```

Save the file, using a filename of your choice (with, perhaps, extension `.txt`). Or, if you prefer, use the one at [link](#).

- (a) Read the data into a data frame called `marks`, using `read_delim`, and list the data frame (by typing its name) to confirm that you read the data values properly. Note that the top line of the data file contains the names of the variables, as it ought to.

#### Solution

I was lazy and used the one on the web, the values being separated (“delimited”) by exactly one space: `::: {.cell}`

```
my_url <- "http://ritsokiguess.site/datafiles/marks.txt"
marks <- read_delim(my_url, " ")
```

```
Rows: 11 Columns: 2
```

```
-- Column specification -----
```

```
Delimiter: " "
```

```
chr (1): class
```

```
dbl (1): score
```

```
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.
```

```
marks
```

```
# A tibble: 11 x 2
```

```
  class  score
  <chr>  <dbl>
1 ken    78
2 ken    62
3 ken    59
4 ken    69
5 ken    81
6 thomas 83
7 thomas 77
```



```

8 thomas    63
9 thomas    61
10 thomas   79
11 thomas   72

```

```
:::
```

If you copied and pasted, or typed in, the data values yourself, use the local file name (such as `marks.txt`) in place of the URL.

Extra: in the old days, when we used `read.table` (which actually also works here), we needed to also say `header=T` to note that the top line of the data file was variable names. With `read_delim`, that's the default, and if the top line is *not* variable names, that's when you have to say so. If I cheat, by skipping the first line and saying that I then have no column names, I get:

```
read_delim(my_url, " ", col_names = F, skip = 1)
```

```
Rows: 11 Columns: 2
```

```
-- Column specification -----
```

```
Delimiter: " "
```

```
chr (1): X1
```

```
dbl (1): X2
```

```
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.
```

```
# A tibble: 11 x 2
```

```

  X1      X2
<chr> <dbl>
1 ken     78
2 ken     62
3 ken     59
4 ken     69
5 ken     81
6 thomas  83
7 thomas  77
8 thomas  63
9 thomas  61
10 thomas 79
11 thomas 72

```

Column names are supplied (X1 and X2). I could also supply my own column names, in which case the file needs not to have any, so I need the `skip` again:

```
read_delim(my_url, " ", col_names = c("instructor", "mark"), skip = 1)
```

Rows: 11 Columns: 2

-- Column specification -----

Delimiter: " "

chr (1): instructor

dbl (1): mark

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.

# A tibble: 11 x 2

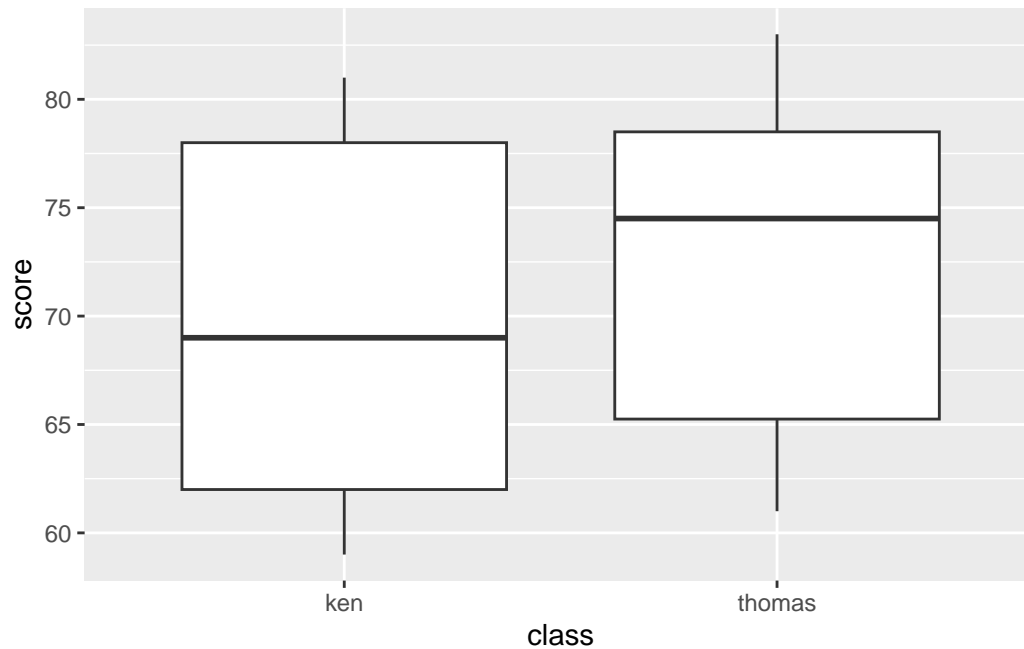
	instructor	mark
	<chr>	<dbl>
1	ken	78
2	ken	62
3	ken	59
4	ken	69
5	ken	81
6	thomas	83
7	thomas	77
8	thomas	63
9	thomas	61
10	thomas	79
11	thomas	72

■

(b) \* Obtain side-by-side boxplots of the scores for each class.

Solution

```
library(tidyverse)
ggplot(marks, aes(x = class, y = score)) + geom_boxplot()
```



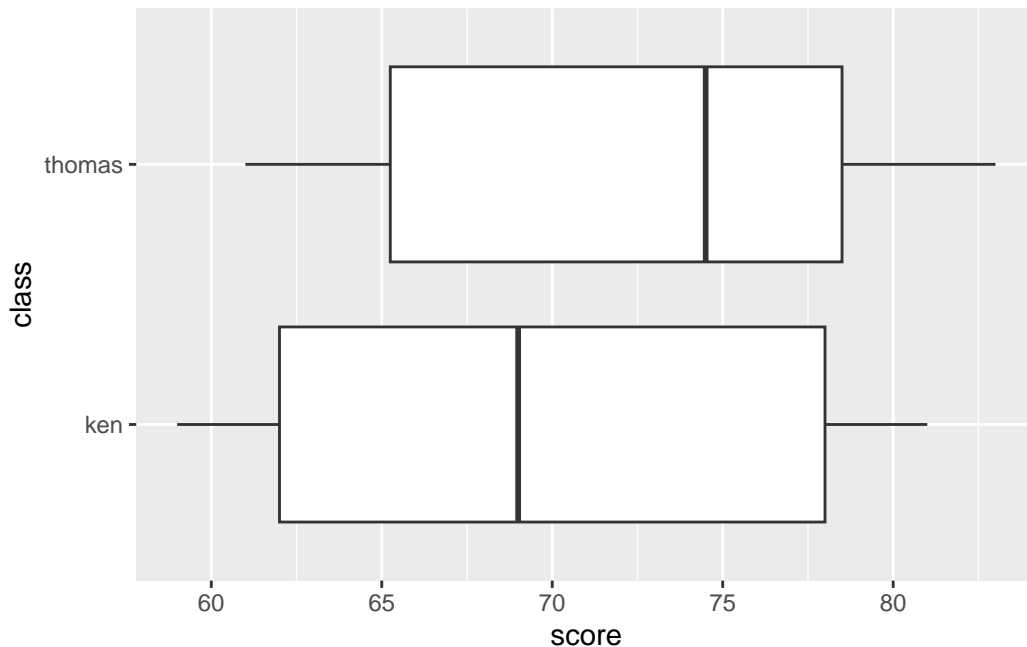
Remember: on a regular boxplot,<sup>7</sup> the groups go across ( $x$ ), the variable measured goes up ( $y$ ).

Extra: this might work:

```
ggplot(marks, aes(x = class, y = score)) + geom_boxplot() +  
  coord_flip()
```

---

<sup>7</sup>Boxplots can also go across the page, but for us, they don't.



It does. That was a guess. So if you want sideways boxplots, this is how you can get them. Long group names sometimes fit better on the  $y$ -axis, in which case flipping the axes will help. (The  $x$  and  $y$  happen *before* the coordinate-flip, so they are the same as above, not the same way they come out.)

■

(c) Do the two classes appear to have similar or different scores, on average? Explain briefly.

Solution

The median for Thomas's class appears to be quite a bit higher than for Ken's class (the difference is actually about 6 marks). It's up to you whether you think this is a big difference or not: I want you to have *an* opinion, but I don't mind so much what that opinion is. Having said that the medians are quite a bit different, note that the boxes overlap substantially, so that the *distributions* of scores are pretty similar (or, the quartiles of scores are similar, or, the IQR of scores is similar for the two groups). If you say that, it's good, but I'm not insisting that you do.

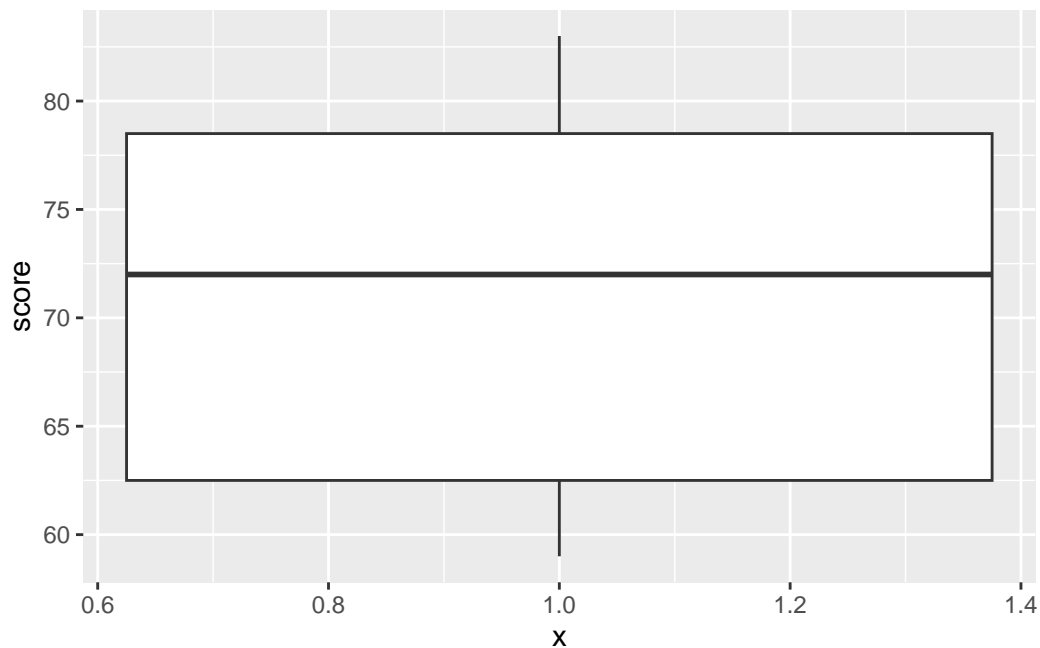
■

(d) Obtain a boxplot of all the scores together, regardless of which class they came from.

Solution

Replace your  $x$ -coordinate by some kind of dummy thing like 1 (`factor(1)` also works): `::: {.cell}`

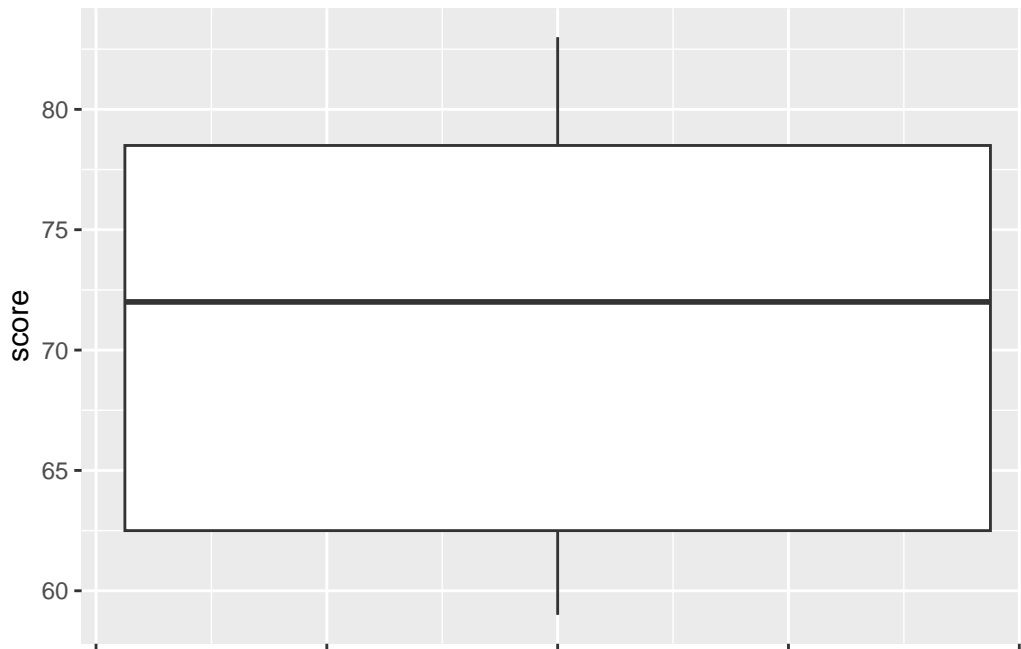
```
ggplot(marks, aes(x = 1, y = score)) + geom_boxplot()
```



⋮

The *x*-axis is kind of dopey, so you just ignore it. It is possible to remove it, but that is more work than it's worth, and I didn't get rid of the ticks below:

```
ggplot(marks, aes(x = 1, y = score)) + geom_boxplot() +  
  theme(  
    axis.text.x = element_blank(),  
    axis.title.x = element_blank()  
  )
```



■

- (e) Compute the median score (of all the scores together). Does this seem about right, looking at the boxplot? Explain briefly.

Solution

Three ways to get the median score. I like the first one best:

```
marks %>% summarize(med = median(score))
```

```
# A tibble: 1 x 1
  med
<dbl>
1    72
```

```
with(marks, median(score))
```

```
[1] 72
```

```
median(marks$score)
```

[1] 72

`summarize` is the `tidyverse` “verb” that does what you want here. (The same idea gets the mean score for each class, below.)

The other ways use the basic function `median`. To make that work, you need to say that the variable `score` whose median you want lives in the data frame `marks`. These are two ways to do that.

Extra: if you wanted median by group, this is the approved (`tidyverse`) way:

```
marks %>%
  group_by(class) %>%
  summarize(med = median(score))
```

```
# A tibble: 2 x 2
  class    med
  <chr>  <dbl>
1 ken      69
2 thomas  74.5
```

To get something by group, the extra step is `group_by`, and then whatever you do after that is done for *each* group.

You can now go back and compare these medians with the ones on the boxplots in (here). They should be the same. Or you can even do this:

```
marks %>%
  group_by(class) %>%
  summarize(
    q1 = quantile(score, 0.25),
    med = median(score),
    q3 = quantile(score, 0.75)
  )
```

```
# A tibble: 2 x 4
  class    q1    med    q3
  <chr>  <dbl> <dbl> <dbl>
1 ken      62     69     78
2 thomas  65.2  74.5  78.5
```

You can calculate as many summaries as you like. These ones should match up with the top and bottom of the boxes on the boxplots. The only restriction is that the things on the right side of the equals should return a *single* number. If you have a function like `quantile` without anything extra that returns more than one number:

```
quantile(marks$score)
```

```
0%  25%  50%  75% 100%
59.0 62.5 72.0 78.5 83.0
```

you're in trouble. Only read on if you *really* want to know how to handle this. Here's step 1:

```
marks %>%
  nest_by(class)
```

```
# A tibble: 2 x 2
# Rowwise:  class
  class      data
  <chr> <list<tibble[,1]>>
1 ken      [5 x 1]
2 thomas    [6 x 1]
```

This is kind of a funky `group_by`. The things in the `data` column are the *whole* rest of the data frame: there were 5 students in Ken's class and 6 in Thomas's, and they each had a `score`, so 5 or 6 rows and 1 column. The column `data` is known in the trade as a "list-column".

Now, for each of those mini-data-frames, we want to calculate the quantiles of `score`. This is `rowwise`: for each of our mini-data-frames `data`, calculate the five-number summary of the column called `score` in *it*:

```
marks %>%
  nest_by(class) %>%
  rowwise() %>%
  mutate(qq = list(quantile(data$score)))
```

```
# A tibble: 2 x 3
# Rowwise:
  class      data qq
  <chr> <list<tibble[,1]>> <list>
```



```
1 ken          [5 x 1] <dbl [5]>
2 thomas       [6 x 1] <dbl [5]>
```

I have to be a little bit careful about which data frame I want the `score` to come from: the ones hidden in `data`, which are the things we're for-eaching over.

This obtains a new list-column called `qq`, with the five-number summary for each group.<sup>8</sup>

Now we want to display the quantiles. This is the easiest way:

```
marks %>%
  nest_by(class) %>%
  rowwise() %>%
  mutate(qq = list(quantile(data$score))) %>%
  unnest(qq)
```

```
# A tibble: 10 x 3
  class      data      qq
  <chr>   <list<tibble[,1]>> <dbl>
1 ken      [5 x 1]    59
2 ken      [5 x 1]    62
3 ken      [5 x 1]    69
4 ken      [5 x 1]    78
5 ken      [5 x 1]    81
6 thomas   [6 x 1]    61
7 thomas   [6 x 1]   65.2
8 thomas   [6 x 1]   74.5
9 thomas   [6 x 1]   78.5
10 thomas  [6 x 1]    83
```

The `unnest` turns the list-column back into actual data, so we get the five quantiles for each class.

The only thing this doesn't do is to show us which quantile is which (we know, of course, that the first one is the minimum, the last one is the max and the quartiles and median are in between). It would be nice to see which is which, though. A trick to do that is to use `enframe`, thus:

```
quantile(marks$score) %>% enframe()
```

---

<sup>8</sup>It's actually a coincidence that the five-number summary and Ken's class both have five values in them.

```
# A tibble: 5 x 2
  name value
  <chr> <dbl>
1 0%    59
2 25%   62.5
3 50%   72
4 75%   78.5
5 100%  83
```

or thus:

```
enframe(quantile(marks$score))
```

```
# A tibble: 5 x 2
  name value
  <chr> <dbl>
1 0%    59
2 25%   62.5
3 50%   72
4 75%   78.5
5 100%  83
```

I don't normally like the second way with all the brackets, but we'll be using it later.

The idea here is that the output from a `quantile` is a vector, but one with “names”, namely the percentiles themselves. `enframe` makes a two-column data frame with the names and the values. (You can change the names of the columns it creates, but here we'll keep track of which is which.)

So we have a *two*-column data frame with a column saying which quantile is which. So let's rewrite our code to use this:

```
marks %>%
  nest_by(class) %>%
  rowwise() %>%
  mutate(qq = list(enframe(quantile(data$score))))
```

```
# A tibble: 2 x 3
# Rowwise:
  class data qq
  <chr> <list<tibble[,1]>> <list>
```

```
1 ken          [5 x 1] <tibble [5 x 2]>
2 thomas       [6 x 1] <tibble [5 x 2]>
```

Note that the qq data frames in the list-column now themselves have two columns.

And finally unnest qq:

```
marks %>%
  nest_by(class) %>%
  rowwise() %>%
  mutate(qq = list(enframe(quantile(data$score)))) %>%
  unnest(qq)
```

```
# A tibble: 10 x 4
  class      data name value
  <chr> <list<tibble[,1]>> <chr> <dbl>
1 ken      [5 x 1] 0%      59
2 ken      [5 x 1] 25%      62
3 ken      [5 x 1] 50%      69
4 ken      [5 x 1] 75%      78
5 ken      [5 x 1] 100%     81
6 thomas   [6 x 1] 0%      61
7 thomas   [6 x 1] 25%     65.2
8 thomas   [6 x 1] 50%     74.5
9 thomas   [6 x 1] 75%     78.5
10 thomas  [6 x 1] 100%     83
```

Success! Or even:

```
marks %>%
  nest_by(class) %>%
  rowwise() %>%
  mutate(qq = list(enframe(quantile(data$score)))) %>%
  unnest(qq) %>%
  mutate(qn = parse_number(name)) %>%
  select(-name) %>%
  pivot_wider(names_from = qn, values_from = value)
```

```
# A tibble: 2 x 7
  class      data `0` `25` `50` `75` `100`
  <chr> <list<tibble[,1]>> <dbl> <dbl> <dbl> <dbl> <dbl>
```

1 ken	[5 x 1]	59	62	69	78	81
2 thomas	[6 x 1]	61	65.2	74.5	78.5	83

This deliberately untidies the final answer to make it nicer to look at. (The lines before that create a numeric quantile, so that it sorts into the right order, and then get rid of the original quantile percents. Investigate what happens if you do a similar `pivot_wider` without doing that.)

### 3.13 Unprecedented rainfall

In 1997, a company in Davis, California, had problems with odour in its wastewater facility. According to a company official, the problems were caused by “unprecedented weather conditions” and “because rainfall was at 170 to 180 percent of its normal level, the water in the holding ponds took longer to exit for irrigation, giving it more time to develop an odour.”

Annual rainfall data for the Davis area is [here](#). Note that clicking on the link will display the file, and *right*-clicking on the link will give you some options, one of which is Copy Link Address, which you can then paste into your R Notebook.

The rainfall is measured in inches.

(a) Read in and display (some of) the data.

Solution

Look at the data file, and see that the values are separated by a single space, so `read_delim` will do it. Read straight from the URL; the hint above tells you how to copy it, which would even work if the link spans two lines.

```
my_url <- "http://ritsokiguess.site/datafiles/rainfall.txt"
rain <- read_delim(my_url, " ")
```

```
Rows: 47 Columns: 2
```

```
-- Column specification -----
```

```
Delimiter: " "
```

```
dbl (2): Year, Rainfall
```

```
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.
```

```
rain
```

```
# A tibble: 47 x 2
  Year Rainfall
  <dbl>   <dbl>
1  1951    20.7
2  1952    16.7
3  1953    13.5
4  1954    14.1
5  1955    25.4
6  1956    12.0
7  1957    28.7
8  1958    11.0
9  1959    12.6
10 1960    12.8
# i 37 more rows
```

Note for later that the `Year` and the `Rainfall` have Capital Letters. You can call the data frame whatever you like, but I think something descriptive is better than eg. `mydata`.

Extra: this works because there is exactly one space between the year and the rainfall amount. But the year is always four digits, so the columns line up, and there is a space all the way down between the year and the rainfall. That means that this will also work:

```
my_url <- "http://ritsokiguess.site/datafiles/rainfall.txt"
rain <- read_table(my_url)
```

```
-- Column specification -----
cols(
  Year = col_double(),
  Rainfall = col_double()
)
```

```
rain
```

```
# A tibble: 47 x 2
  Year Rainfall
  <dbl>   <dbl>
1  1951    20.7
2  1952    16.7
3  1953    13.5
4  1954    14.1
```

```

5 1955      25.4
6 1956      12.0
7 1957      28.7
8 1958      11.0
9 1959      12.6
10 1960     12.8
# i 37 more rows

```

This is therefore also good.

It also looks as if it could be tab-separated values, since the rainfall column always starts in the same place, but if you try it, you'll find that it doesn't work:

```

my_url <- "http://ritsokiguess.site/datafiles/rainfall.txt"
rain_nogood <- read_tsv(my_url)

```

```

Rows: 47 Columns: 1

```

```

-- Column specification -----
Delimiter: "\t"
chr (1): Year Rainfall

```

```

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.

```

```

rain_nogood

```

```

# A tibble: 47 x 1
  `Year Rainfall`
  <chr>
1 1951 20.66
2 1952 16.72
3 1953 13.51
4 1954 14.1
5 1955 25.37
6 1956 12.05
7 1957 28.74
8 1958 10.98
9 1959 12.55
10 1960 12.75
# i 37 more rows

```

This looks as if it worked, but it didn't, because there is only *one* column, of years and rainfalls smooshed together as text, and if you try to do anything else with them later it won't work.

Hence those values that might have been tabs actually were not. There's no way to be sure about this; you have to try something and see what works. An indication, though: if you have more than one space, and the things in the later columns are *left*-justified, that could be tab-separated; if the things in the later columns are *right*-justified, so that they finish in the same place but don't start in the same place, that is probably aligned columns.

■

(b) Summarize the data frame.

Solution

I almost gave the game away: this is `summary`.

```
summary(rain)
```

	Year	Rainfall
Min.	:1951	Min. : 6.14
1st Qu.	:1962	1st Qu.:12.30
Median	:1974	Median :16.72
Mean	:1974	Mean :18.69
3rd Qu.	:1986	3rd Qu.:25.21
Max.	:1997	Max. :37.42

The summary of the years may not be very helpful, but the summary of the annual rainfall values might be. It's not clear yet why I asked you to do this, but it will become clearer later.

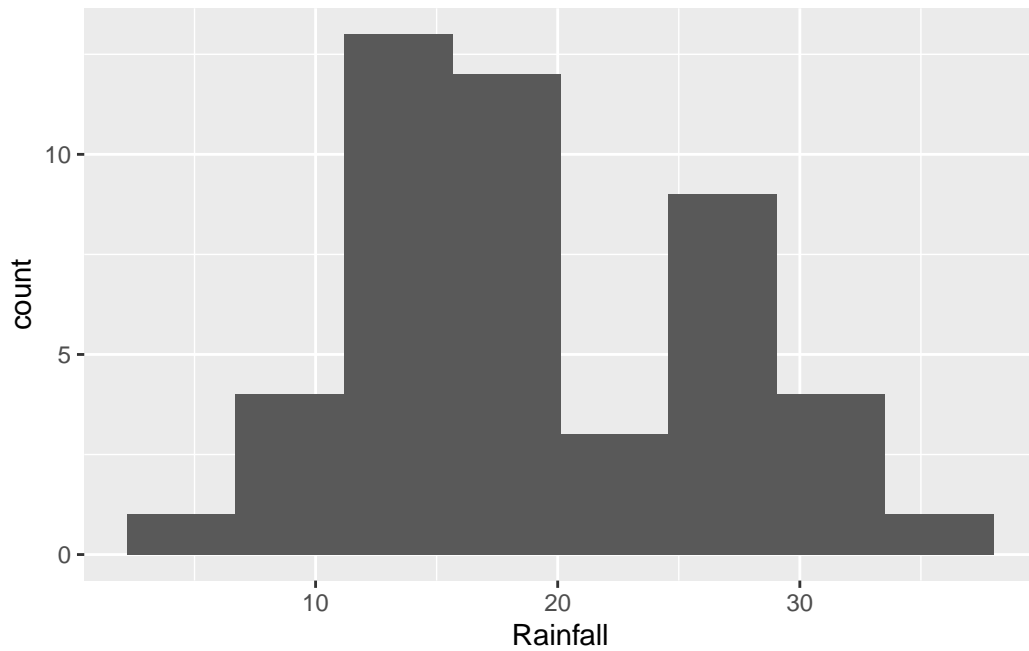
■

(c) Make a suitable plot of the rainfall values. (We are not, for the time being, concerned about the years.)

Solution

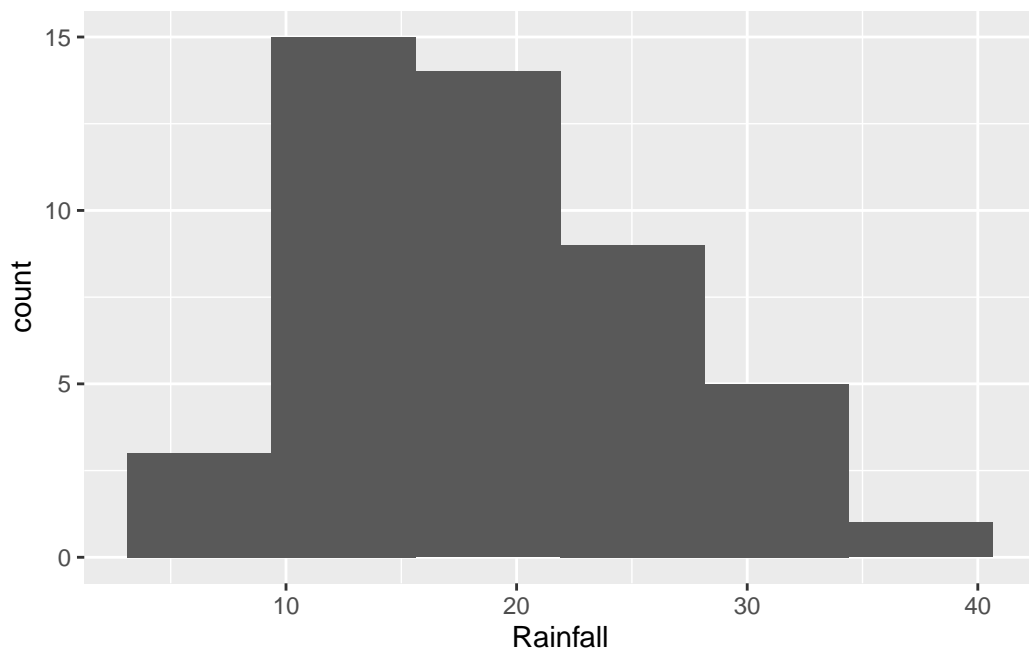
This is one quantitative variable, so a histogram is your first thought. This means picking a number of bins. Not too many, since you want a picture of the shape:

```
ggplot(rain, aes(x=Rainfall)) + geom_histogram(bins=8)
```



If you picked fewer bins, you'll get a different picture:

```
ggplot(rain, aes(x=Rainfall)) + geom_histogram(bins=6)
```





The choice of the number of bins depends on what you think the story about shape is that you want to tell (see next part). You will probably need to try some different numbers of bins to see which one you like best. You can say something about what you tried, for example “I also tried 8 bins, but I like the histogram with 6 bins better.”

■

(d) How would you describe the shape of the distribution of rainfall values?

Solution

This will depend on the histogram you drew in the previous part. If it looks like the first one, the best answer is “bimodal”: that is, it has two peaks with a gap between them. If it looks like the second one, you have an easier time; this is ordinary right-skewness.

■

(e) In the quote at the beginning of the question, where do you think the assertion that the 1997 rainfall was “at 170 to 180 percent of its normal level” came from? Explain briefly.

Solution

First we need the 1997 rainfall. Go back and find it in the data. I am borrowing an idea from later in the course (because I am lazy):

```
rain %>% filter(Year==1997)
```

```
# A tibble: 1 x 2
  Year Rainfall
  <dbl>   <dbl>
1  1997    29.7
```

29.7 inches.

Now, what would be a “normal level” of rainfall? Some kind of average, like a mean or a median, maybe. But we have those, from our summary that we made earlier, repeated here for (my) convenience:

```
summary(rain)
```

	Year		Rainfall
Min.	:1951	Min.	: 6.14
1st Qu.	:1962	1st Qu.	:12.30
Median	:1974	Median	:16.72

Mean	:1974	Mean	:18.69
3rd Qu.	:1986	3rd Qu.	:25.21
Max.	:1997	Max.	:37.42

The mean is 18.69 and the median is 16.72 inches.

So divide the 1997 rainfall by each of the summaries, and see what happens, using your calculator, or using R as a calculator:

```
29.7/18.69
```

```
[1] 1.589085
```

```
29.7/16.72
```

```
[1] 1.776316
```

The 1997 rainfall was about 178 percent of the normal level if the normal level was the *median*.



- (f) Do you think the official's calculation was reasonable? Explain briefly. (Note that this is not the same as asking whether the official's calculation was *correct*. This is an important distinction for you to make.)

Solution

There are several approaches to take. Argue for yours.

If you came to the conclusion that the distribution was right-skewed, you can say that the sensible “normal level” is the median, and therefore the official did the right thing. Using the mean would have been the wrong thing.

If you thought the distribution was bimodal, you can go a couple of ways: (i) it makes no sense to use any measure of location for “normal” (in fact, the mean rainfall is almost in that low-frequency bar, and so is not really a “normal level” at all). Or, (ii) it looks as if the years split into two kinds: low-rainfall years with around 15 inches, and high-rainfall years with more than 25 inches. Evidently 1997 was a high-rainfall year, but 29.7 inches was not especially high for a high-rainfall year, so the official's statement was an exaggeration. (I think (ii) is more insightful than (i), so ought to get more points.)

You could even also take a more conspiratorial approach and say that the official was trying to make 1997 look like a freak year, and picked the measure of location that made 1997 look more unusual.

“Normal level” here has nothing to do with a normal *distribution*; for this to make sense, the official would have needed to say something like “normal shape”. This is why language skills are also important for a statistician to have.



- (g) Do you think that the official was right to use the word “unprecedented” to describe the 1997 rainfall? Justify your answer briefly.

Solution

“Unprecedented” means “never seen before” or “never having happened or existed in the past”.<sup>9</sup> That came out of my head; [this link](#) has a very similar “never before known or experienced”).

If you look back at your histogram, there are several years that had over about 30 inches of rain: five or six, depending on your histogram. One of them was 1997, but there were others too, so 1997 was in no way “unprecedented”.

Another approach that you have seen is to `View` your dataframe:

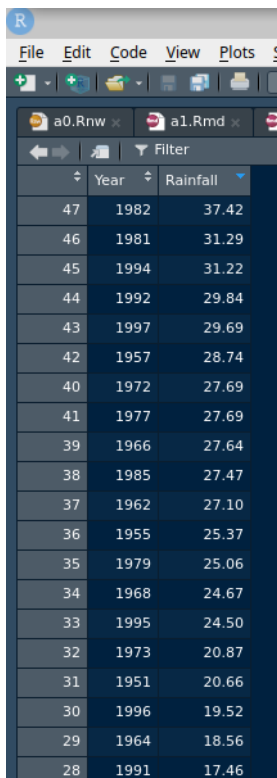
```
View(rain)
```

That will come out as a separate tab in your R Studio and you can look at it (yourself; it won’t appear in the Preview). You can look at the 1997 rainfall (29.69 inches) and count how many were bigger than that, 4 of them. Or, save yourself some effort<sup>10</sup> and sort the rainfall values in descending order (with the biggest one first), by clicking on the little arrows next to Rainfall (twice). Mine looks like this:

---

<sup>9</sup>Searching for “define” followed by a word is a good way to find out exactly what that word means, if you are not sure, but you should at least say where you got the definition from if you had to look it up.

<sup>10</sup>When you have a computer at your disposal, it’s worth taking a few minutes to figure out how to use it to make your life easier.



	Year	Rainfall
47	1982	37.42
46	1981	31.29
45	1994	31.22
44	1992	29.84
43	1997	29.69
42	1957	28.74
40	1972	27.69
41	1977	27.69
39	1966	27.64
38	1985	27.47
37	1962	27.10
36	1955	25.37
35	1979	25.06
34	1968	24.67
33	1995	24.50
32	1973	20.87
31	1951	20.66
30	1996	19.52
29	1964	18.56
28	1991	17.46

Later, we learn how to sort in code, which goes like this (to sort highest first):

```
rain %>% arrange(desc(Rainfall))
```

```
# A tibble: 47 x 2
  Year Rainfall
  <dbl>   <dbl>
1  1982    37.4
2  1981    31.3
3  1994    31.2
4  1992    29.8
5  1997    29.7
6  1957    28.7
7  1972    27.7
8  1977    27.7
9  1966    27.6
10 1985    27.5
# i 37 more rows
```

A more sophisticated way that we learn later:

```
rain %>% summarize(max=max(Rainfall))

# A tibble: 1 x 1
  max
<dbl>
1  37.4
```

This is greater than the rainfall for 1997, ruling out “unprecedented”.

1997 was only the *fifth* highest rainfall, and two of the higher ones were also in the 1990s. Definitely not “unprecedented”. The official needs to get a new dictionary!



### 3.14 Learning algebra

At a high school in New Jersey, teachers were interested in what might help students to learn algebra. One idea was laptops as a learning aid, to see whether having access to one helped with algebra scores. (This was some time ago.) The 20 students in one class were given laptops to use in school and at home, while the 27 students in another class were not given laptops. For all of these students, the final exam score in algebra was recorded. The data are in <http://ritsokiguess.site/datafiles/algebra.txt>, with two columns, one indicating whether the student received a laptop or not, and the other giving their score on the algebra final exam.

- (a) Read in and display (some of) the data. Do you have (i) the correct number of observations, and (ii) the correct *type* of columns? Explain briefly.

Solution

Take a look at the data file first: the data values are *aligned in columns* with variable numbers of spaces between, so `read_table` is the thing. Read directly from the URL, rather than trying to copy the data from the website:

```
my_url <- "http://ritsokiguess.site/datafiles/algebra.txt"
algebra <- read_table(my_url)

-- Column specification -----
cols(
  laptop = col_character(),
  score = col_double()
)
```

## algebra

```
# A tibble: 47 x 2
  laptop score
  <chr>   <dbl>
1 yes     98
2 yes     84
3 yes     97
4 yes     93
5 yes     88
6 yes     57
7 yes    100
8 yes     84
9 yes    100
10 yes    81
# i 37 more rows
```

There were  $20 + 27 = 47$  students altogether in the two classes, and we do indeed have 47 rows, one per student. So we have the right number of rows. This is two independent samples; each student was in only one of the two classes, either the class whose students got laptops or not. The values in the `laptop` column are text (see the `chr` at the top), and the values in the `score` column are numbers (`dbl` at the top). Alternatively, you can look at the R Console output in which you see that `laptop` is `col_character()` (text) and `score` is `col_double()` (numerical, strictly a decimal number).

Extra 1: `read.table` also works but it is *wrong* in this course (because it is not what I taught you in class).

Extra 2: with more than one space between the values, `read_delim` will not work. Or, perhaps more confusing, it will appear to work and then fail later, which means that you need to pay attention:

```
d <- read_delim(my_url, " ")
```

Warning: One or more parsing issues, call ``problems()`` on your data frame for details, e.g.:

```
dat <- vroom(...)
problems(dat)
```

Rows: 47 Columns: 2

-- Column specification -----

```
Delimiter: " "  
chr (2): laptop, score
```

```
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.
```

```
d
```

```
# A tibble: 47 x 2  
  laptop score  
  <chr>   <chr>  
1 yes    "    98"  
2 yes    "    84"  
3 yes    "    97"  
4 yes    "    93"  
5 yes    "    88"  
6 yes    "    57"  
7 yes    "   100"  
8 yes    "    84"  
9 yes    "   100"  
10 yes   "    81"  
# i 37 more rows
```

This *looks* all right, but look carefully: the `laptop` column is correctly text, but the `score` column, which should be numbers (`dbl`), is actually text as well. An easier way to see this is to look at the output from the console, which is the descriptions of the columns: they are *both* `col_character` or text, while `score` should be numbers. You might be able to see exactly what went wrong: with more than one space separating the values, the remaining spaces went into `score`, which then becomes a piece of text with some spaces at the front and then numbers.

This will actually work for a while, as you go through the question, but will come back to bite you the moment you need `score` to be numerical (eg. when you try to draw a boxplot), because it is actually not numerical at all.

Extra 3: this is the standard R way to lay out this kind of data, with all the outcome values in one column and a second (categorical) column saying which group each observation was in. In other places you might see two separate columns of scores, one for the students with laptops and one for the students without, as below (you won't understand the code below now, but you will by the end of the course):

```
algebra %>%  
  mutate(row = c(1:20, 1:27)) %>%
```

```
pivot_wider(names_from = laptop, values_from = score)
```

```
# A tibble: 27 x 3
   row  yes  no
  <int> <dbl> <dbl>
1     1    98   63
2     2    84   83
3     3    97   97
4     4    93   93
5     5    88   52
6     6    57   74
7     7   100   83
8     8    84   63
9     9   100   88
10    10    81   86
# i 17 more rows
```

A column of `yes` and a column of `no`. The classes were of different sizes, so the `yes` column, with only 20 observations, has some `NA` (“missing”) observations at the end (scroll down to see them) to enable the dataframe to keep a rectangular shape.

We will learn later what to call these layouts of data: “longer” and “wider” (respectively), and how to convert between them. R usually likes “longer” data, as in the data file, but you will often see data sets displayed wider because it takes up less space.



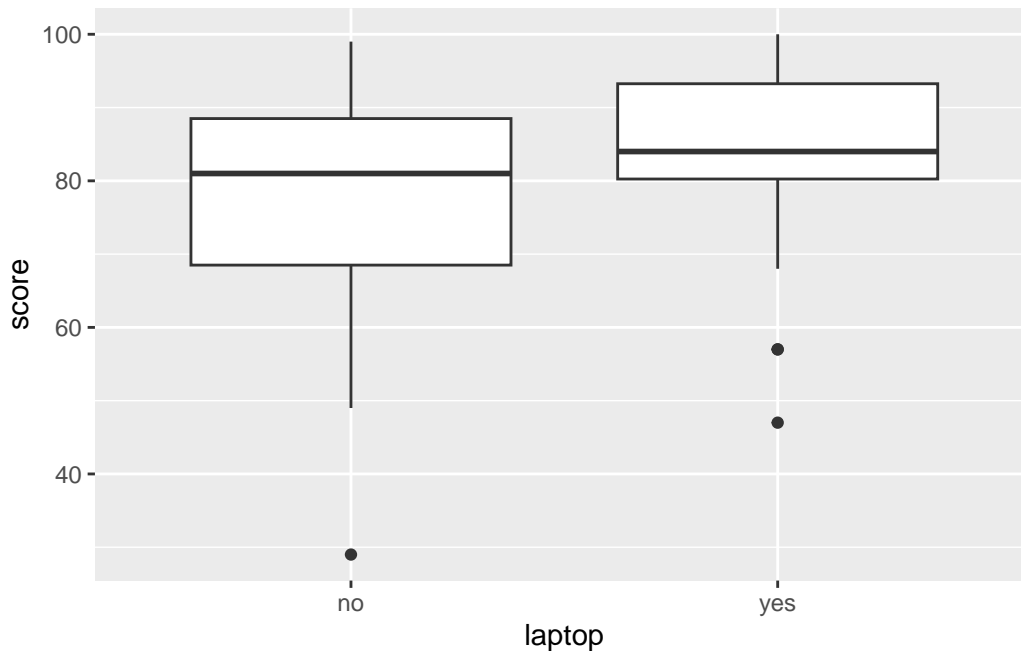
(b) Make a suitable graph of these data.

Solution

The teachers were hoping to see how the laptop-yes and the laptop-no groups compared in terms of algebra scores, so side-by-side boxplots would be helpful. More simply, we have one quantitative and one categorical variable, which is a boxplot according to the table in the notes:

```
ggplot(algebra, aes(x = laptop, y = score)) + geom_boxplot()
```

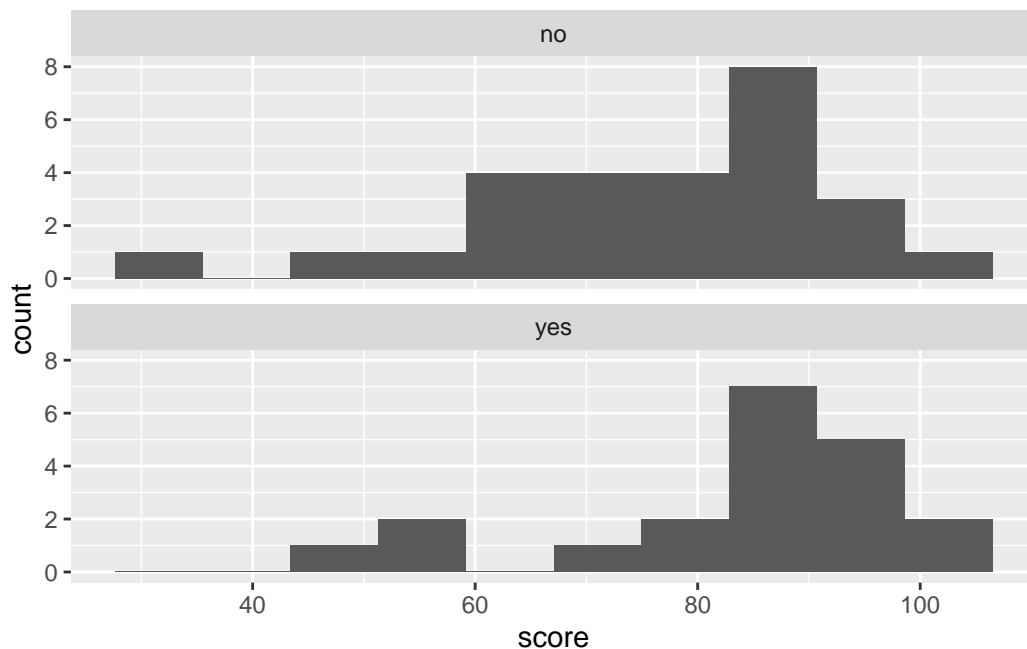




Extra: as you will note below, the median score for the students with laptops is a little higher for the students who had laptops. This is easy to see on a boxplot because that is what a boxplot does. (That was what Tukey, who we will meet later, *designed* the boxplot to do.)

Another plot you might have drawn is a histogram for each group, side by side, or, as they come out here, above and below. This works using facets:

```
ggplot(algebra, aes(x = score)) +  
  geom_histogram(bins = 10) +  
  facet_wrap(~laptop, ncol = 1)
```



Looking at those, can you *really* say that the median is slightly higher for the **yes** group? I really don't think you can. Certainly it is clear from the histograms that the spread for the **yes** group is less, but comparing the medians is much more easily done from the boxplot. The teachers were interested in whether the laptops were associated with higher scores on average, so the kind of comparison that the boxplot affords is clearly preferred here.

If you are interested in the code: you imagine you're going to make a histogram of scores regardless of group, and then at the end you facet by your grouping variable. I added the `ncol = 1` to make the plots come out in one column (that is, one above the other). If you don't do this, they come out left and right, which makes the distributions even harder to compare.

■

(c) Comment briefly on your graph, thinking about what the teachers would like to know.

Solution

There are three things to say something about, the first two of which would probably interest the teachers:

- comparison of centre: the *median* score for the group that had laptops is (slightly) higher than for the group that did not.
- comparison of spread: the scores for the group that had laptops are less spread out (have smaller *interquartile range*) than for the group that did not.
- assessment of shape: both groups have low outliers, or are skewed to the left in shape.

Some comments from me:

- boxplots say *nothing* about mean and standard deviation, so don't mention those here. You should say something about the measures of centre (median) and spread (IQR) that they *do* use.
- I think of skewness as a property of a whole distribution, but outlierness as a property of individual observations. So, when you're looking at this one, think about where the evidence about shape is coming from: is it coming from those one or two low values that are different from the rest (which would be outliers), or is it coming from the whole distribution (would you get the same story if those maybe-outliers are taken away)? My take is that if you take the outliers away, both distributions are close to symmetric, and therefore what you see here is outliers rather than skewness. If you see something different, make the case for it.

One reason to suspect skewness or something like it is that test scores have an upper limit (100) that some of the scores got close to, and no effective lower limit (the lower limit is 0 but no-one got very close to that). In this sort of situation, you'd expect the scores to be skewed away from the limit: that is, to the left. Or to have low outliers rather than high ones.

■

- (d) Work out the median and inter-quartile range for the students who did and who did not have laptops, and compare with the boxplot. (In R, the inter-quartile range is `IQR` in uppercase.)

Solution

This is easy to make way harder than it needs to be: `group_by` and `summarize` will do it. Put the two summaries in one `summarize`:

```
algebra %>%  
  group_by(laptop) %>%  
  summarize(med = median(score), iqr = IQR(score))
```

```
# A tibble: 2 x 3  
  laptop    med    iqr  
  <chr>   <dbl> <dbl>  
1 no         81     20  
2 yes        84     13
```

Then relate these to the information on the boxplot: the centre line of the box is the median. For the `no` group this is just above 80, so 81 makes sense; for the `yes` group this is not quite halfway between 80 and 90, so 84 makes sense.

The inter-quartile range is the height of the box for each group. Estimate the top and bottom of the two boxes from the boxplot scale, and subtract. For the **no** group this is something like  $88 - 68$  which *is* 20, and for the **yes** group it is something like  $93 - 80$  which is indeed 13.

Extra: I didn't ask you here about whether the difference was likely meaningful. The focus here was on getting the graph and summaries. If I had done so, you would then need to consider things like whether a three-point difference in medians could have been chance, and whether we really had random allocation of students to groups.

To take the second point first: these are students who chose to take two different classes, rather than being randomly allocated to classes as would be the case in a true experiment. What we have is really in between an experiment and an observational study; yes, there was a treatment (laptop or not) that was (we hope) randomly allocated to one class and not the other, but the classes could have been different for any number of other reasons that had nothing to do with having laptops or not, such as time of day, teacher, approach to material, previous ability at algebra, etc.

So even if we are willing to believe that the students were as-if randomized to laptop or not, the question remains as to whether that three-point difference in medians is reproducible or indicative of a real difference or not. This is the kind of thing we would try a two-sample *t*-test with. In this case, we might doubt whether it will come out significant (because of the small difference in medians and presumably means, compared to the amount of variability present), and, even then, there is the question of whether we *should* be doing a *t*-test at all, given the outliers.

■