Problems and Solutions in Applied Statistics

Ken Butler

2018-10-09

Chapter 1

Introduction

This book will hold a collection of problems, and their solutions, in applied statistics with R. These come from my courses STAC32 and STAD29 at the University of Toronto Scarborough.

The problems were originally written in Sweave (that is, LaTeX with R code chunks), using the `exam` document class. I wrote a Perl program to strip out the LaTeX and turn each problem into R Markdown for this book. You will undoubtedly see bits of LaTeX still embedded in the text. I am trying to update my program to catch them, but I am sure to miss some. If you see anything, file an issue on the Github page for now. I want to fix problems programmatically at first, but when the majority of the problems have been caught, I will certainly take pull requests. I will acknowledge all the people who catch things.

Current progress: 5 chapters (up to one-sample inference), but the problems in chapters 4 and 5 don't have titles yet. (I want to fix up my Perl to catch some more latexery, and when I am happy with that, the problems will get titles.)

Chapter 2

Getting used to R and R Studio

We begin with this:

```
library(tidyverse)
```

```
## -- Attaching packages ---------------------------------------------------------------------------------
```

```
## v ggplot2 3.0.0     v purrr   0.2.5
## v tibble  1.4.2     v dplyr   0.7.6
## v tidyr   0.8.1     v stringr 1.3.1
## v readr   1.1.1     v forcats 0.3.0
```

```
## -- Conflicts ------------------------------------------------------------------------------------------
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

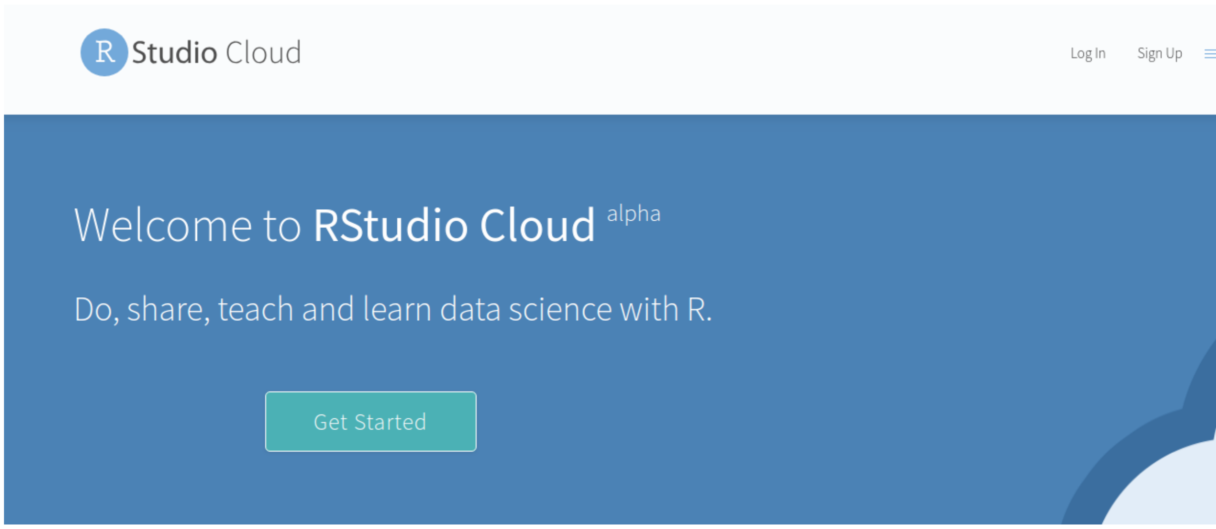
and so to the problems:

Getting started on R Studio Cloud

Follow these steps to get an R Studio Cloud account.

(a) Point your web browser at rstudio.cloud. (If you already have R and R Studio installed on your computer, you can use that instead, throughout the course; just do part (d) of this question. Any references to R Studio Cloud in this assignment also apply to R Studio on your computer.)

Solution

You should see this: 

Welcome to RStudio Cloud alpha

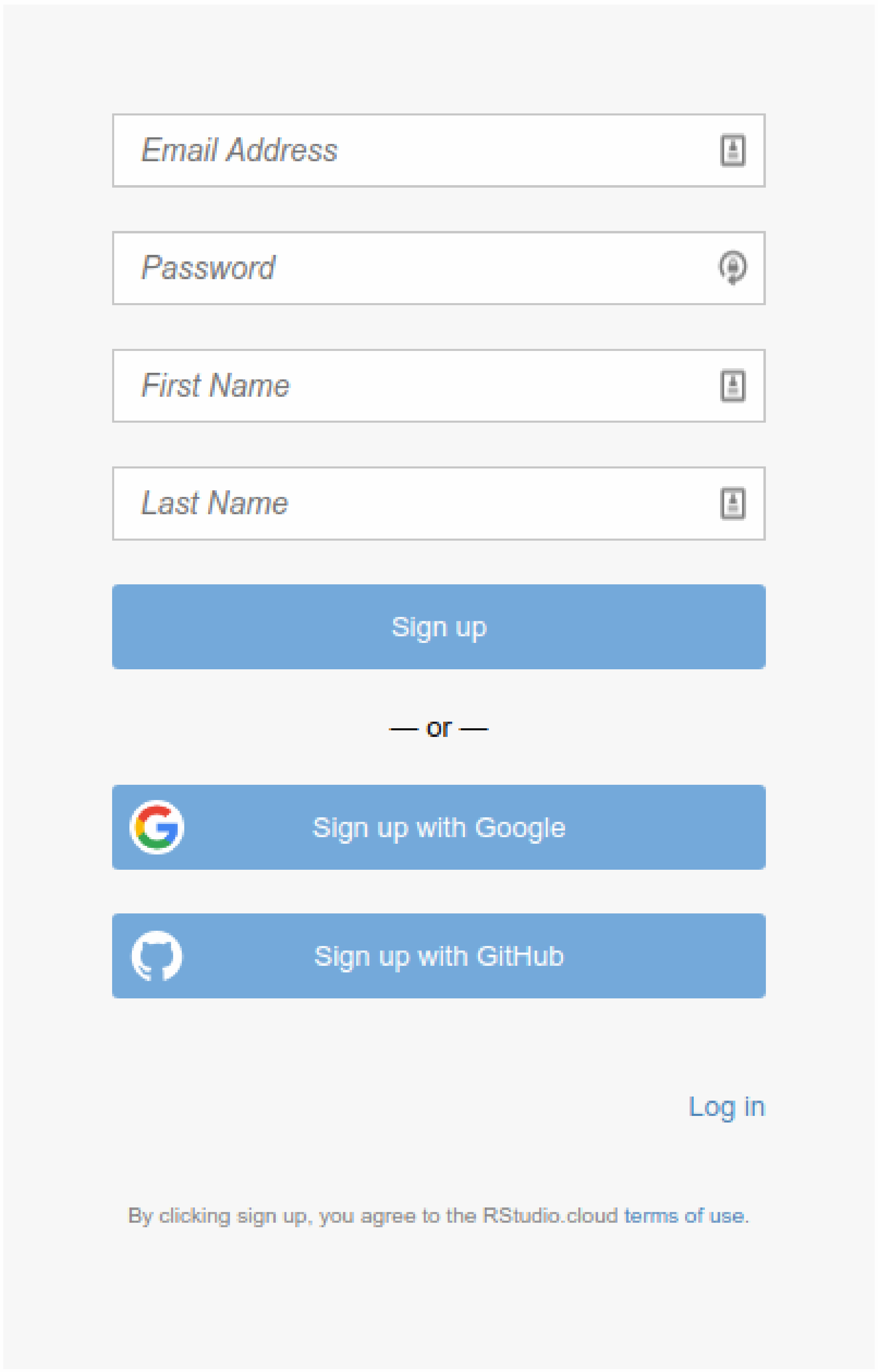
Do, share, teach and learn data science with R.

Click on Get Started. You might instead see the screen in the next part.

(b) Choose an account to use.

Solution

Here's what you should see now:

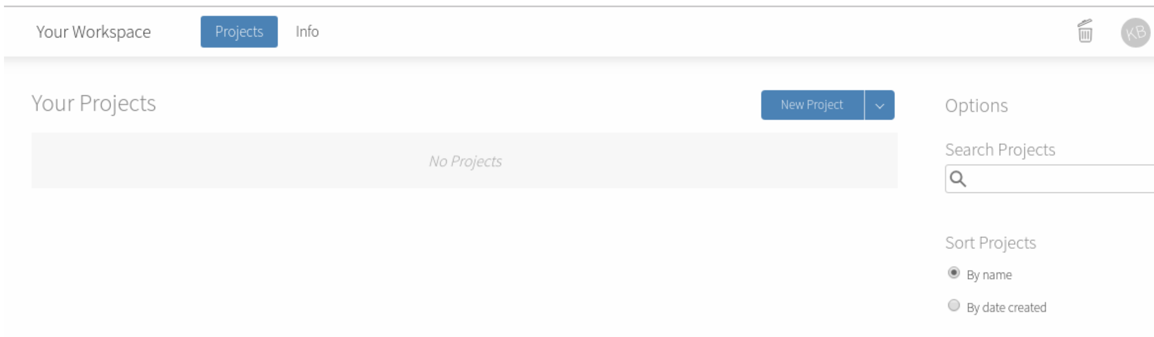


By clicking sign up, you agree to the RStudio.cloud terms of use.

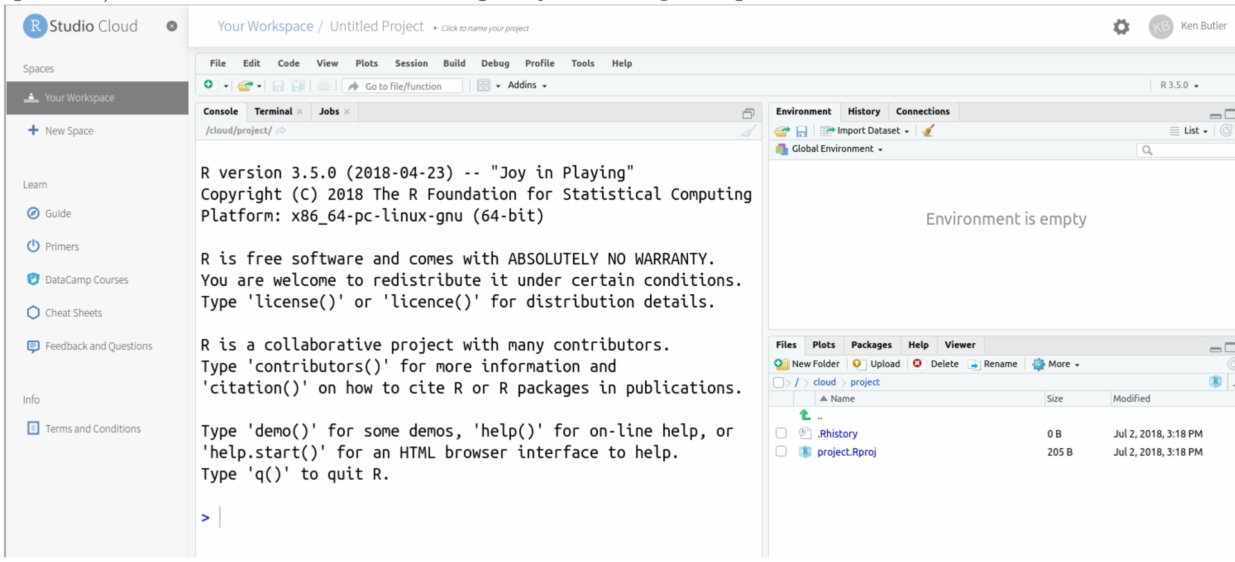
If you're happy with using your Google account, click that button. You will probably have to enter your Google password. (If you are doing this on your own computer, you might not have to do that.) If you have a GitHub account and you want to use *that*, same principle. You can also use an email address as your login to R Studio Cloud. (You can use any e-mail address; I'm not checking.) Enter it in the top box, and enter a password to use with R Studio Cloud in the second. (This does not have to be, and indeed probably should not be, the same as your email password.) Below that, enter your first and last name. This will appear at the top right of the screen when you are logged in. Then click Sign Up. After that, you will have to make a unique account name (which *you* actually never use, but verb+rstudio.cloud+ uses to name your files). After that, you are automatically logged in.

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

Solution



This is what you see now:
Click on the blue New Project button to create a new Project. (A project is a self-contained piece of work, like for example an assignment.) You will see the words “Loading Project” and spinning circles for a few moments.

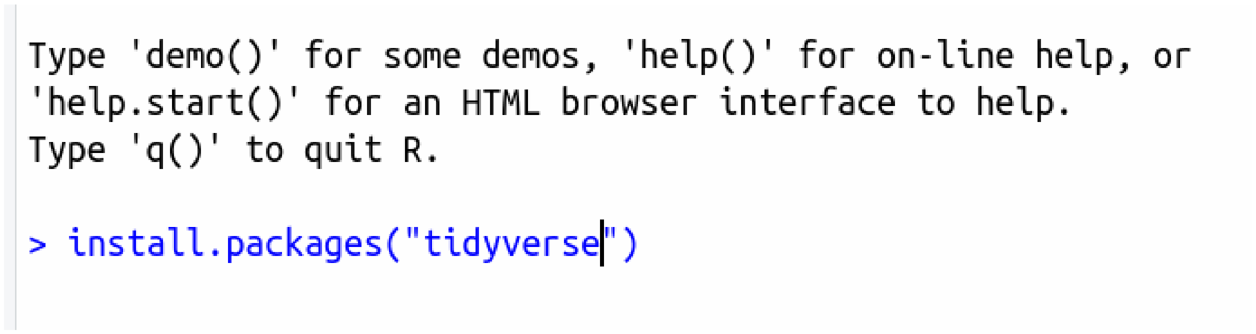


```
R version 3.5.0 (2018-04-23) -- "Joy in Playing"
Copyright (C) 2018 The R Foundation for Statistical Computing
Platform: x86_64-pc-linux-gnu (64-bit)

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.

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.

>
```

```
.Rhistory      0 B     Jul 2, 2018, 3:18 PM
project.Rproj  205 B   Jul 2, 2018, 3:18 PM
```

Then you see this:
To give your project a name, click at the top where it says Untitled Project and type a name like Assignment 0 into the box.

(d) Before we get to work, look for the blue > at the bottom left. Click next to it to get a flashing cursor, and then type what you see here (in blue):

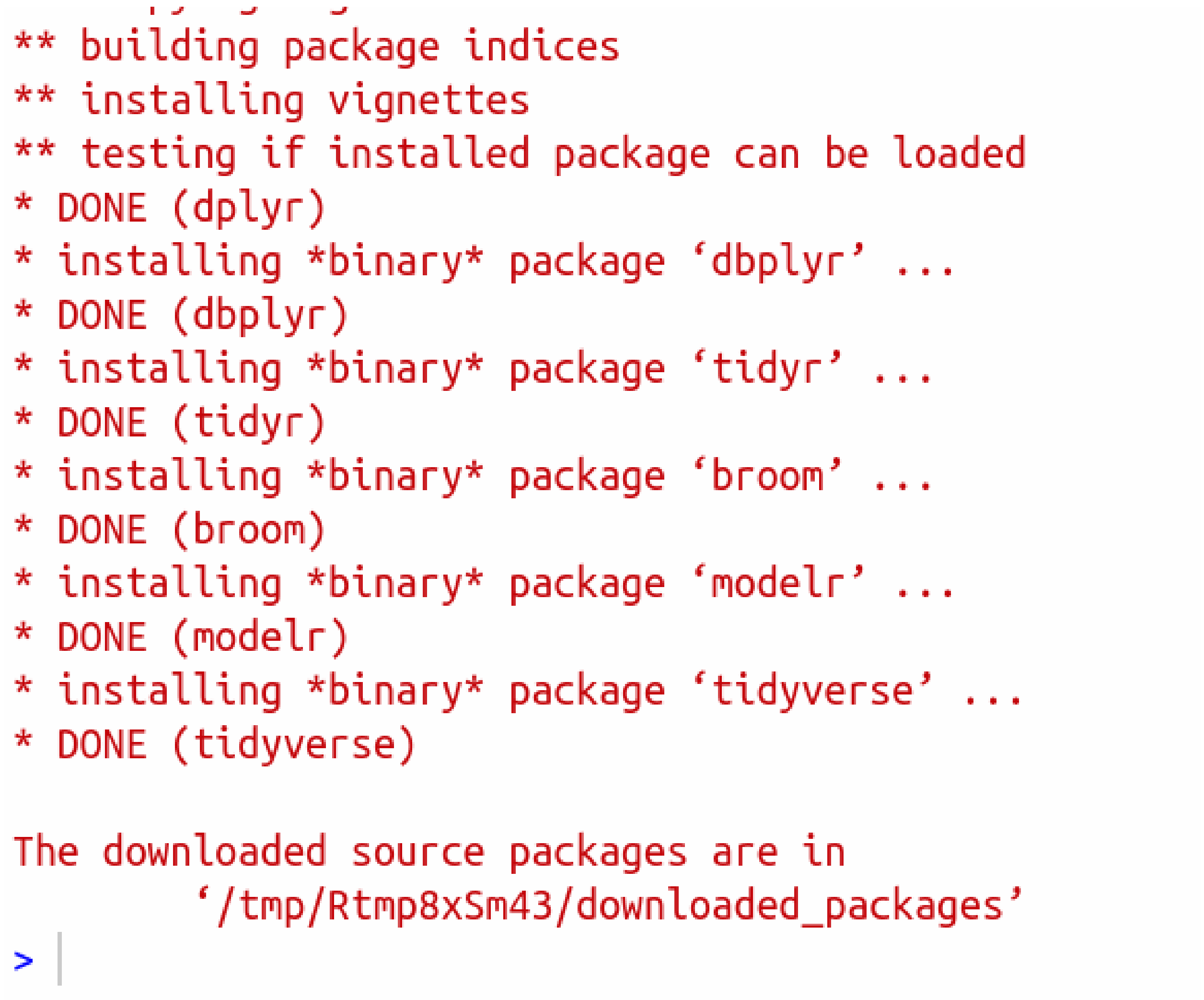


```
Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

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

Then press Enter.

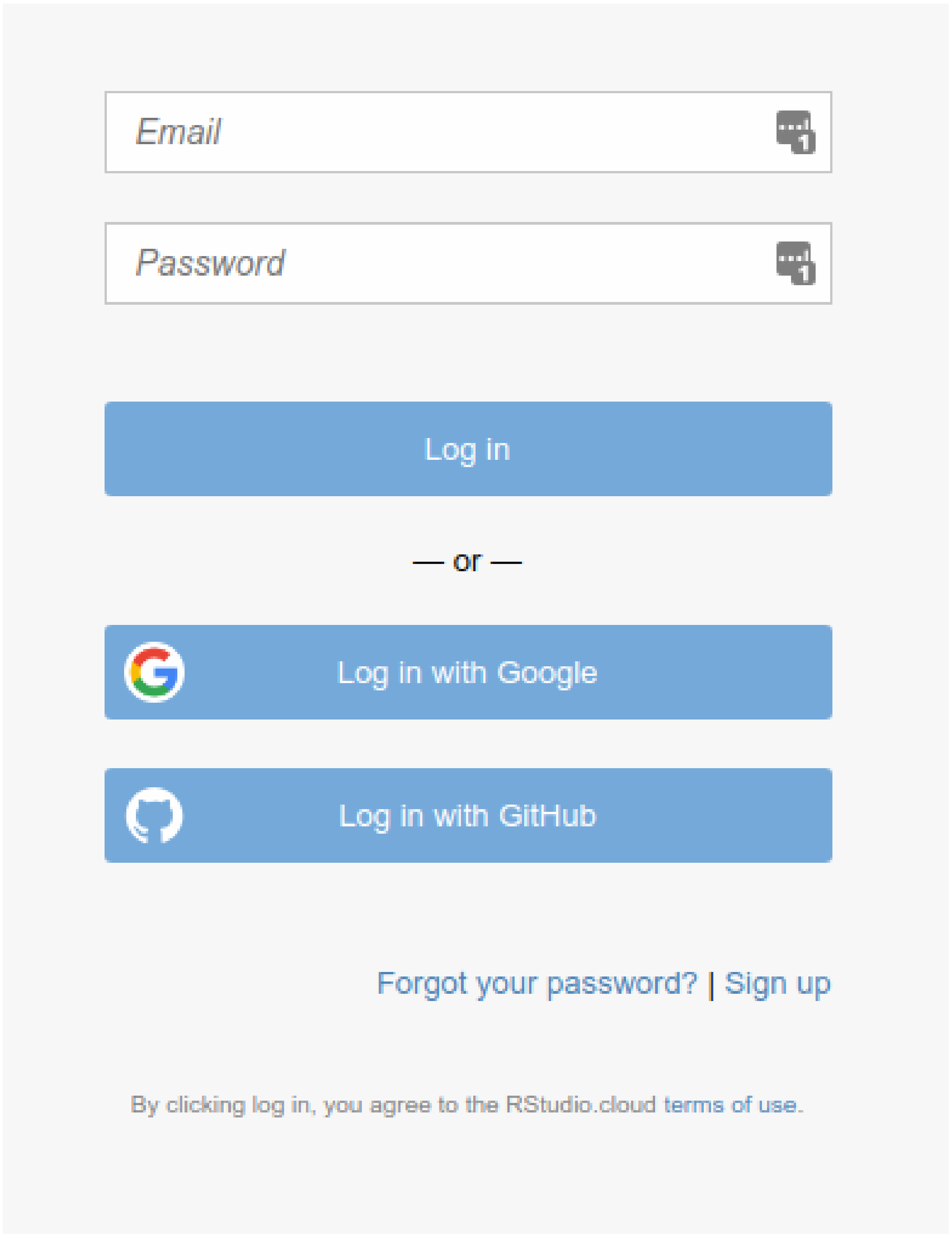
Solution

This lets it install a bunch of things. It may take some time. If you are watching it, look out for lines beginning with `g++`, which are C++ code that needs to be compiled. This is the end of what I had. Look out for the word DONE near the bottom:



```
** building package indices
** installing vignettes
** testing if installed package can be loaded
* DONE (dplyr)
* installing *binary* package ‘dbplyr’ ...
* DONE (dbplyr)
* installing *binary* package ‘tidyr’ ...
* DONE (tidyr)
* installing *binary* package ‘broom’ ...
* DONE (broom)
* installing *binary* package ‘modelr’ ...
* DONE (modelr)
* installing *binary* package ‘tidyverse’ ...
* DONE (tidyverse)

The downloaded source packages are in
        ‘/tmp/Rtmp8xSm43/downloaded_packages’
>
```

(e) Not for now, but for later: if you are on a lab computer, you should probably log out when you are done. To do that, find your name at the top right. Click on it, and two things should pop out to the right: Profile and Log Out. Select Log Out. You should be returned to one of the screens you began with, possibly the Welcome to R Studio Cloud one. To log back in, now or next time, look for Log In at the top right. Click it, to get this:


Email
Password
Log in
— or —
Log in with Google
Log in with GitHub
Forgot your password? | Sign up
By clicking log in, you agree to the RStudio.cloud terms of use.

and then you can log in with your email and password, or Google or Github IDs, whichever you used. Now we can get down to some actual work.

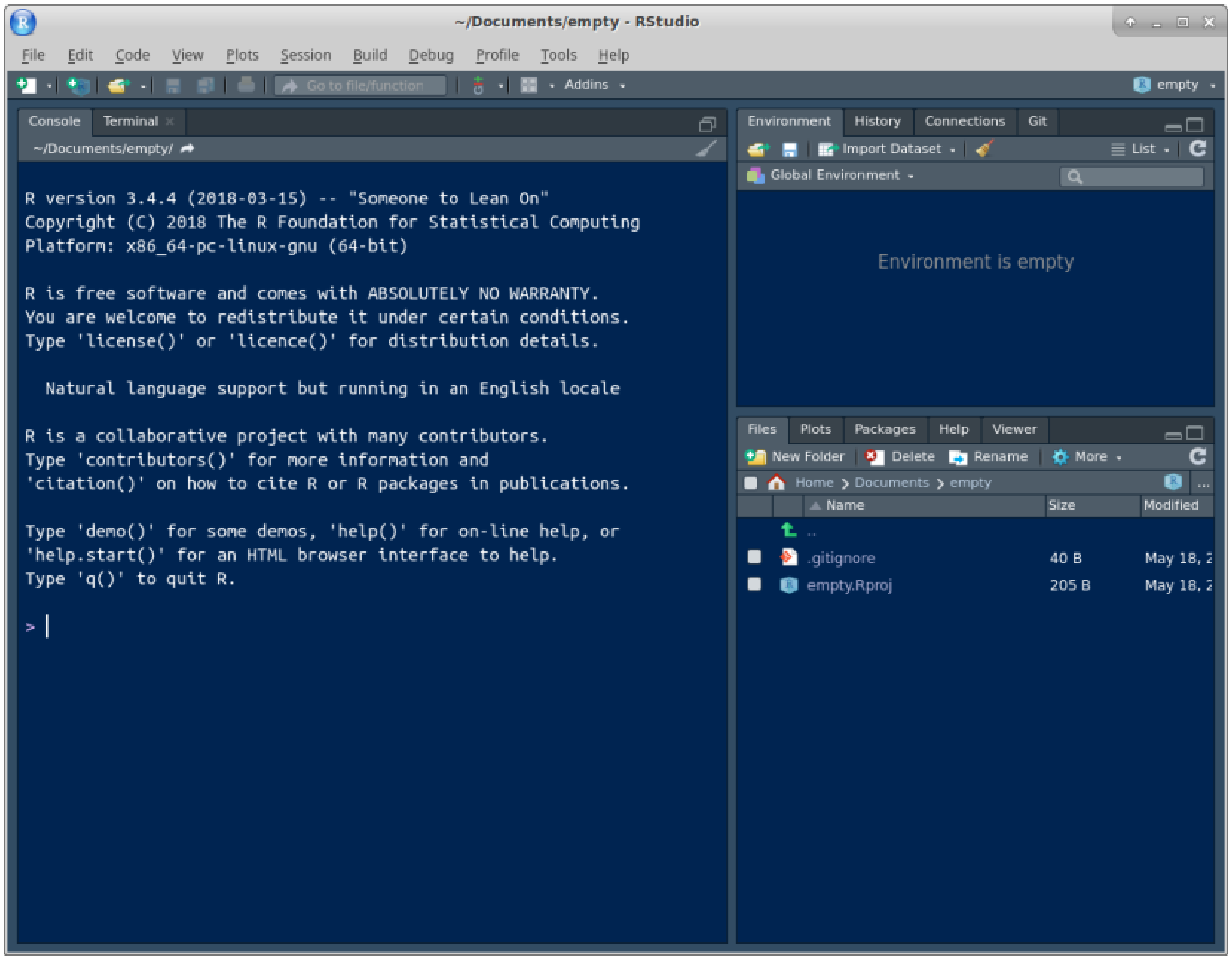
Getting used to R Studio

This question is to get you started using R.

(a) Start R Studio Cloud, 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.)

Solution

You ought to see something like this. I have a dark blue background here, which you probably do not.


~/Documents/empty - RStudio
File Edit Code View Plots Session Build Debug Profile Tools Help
Go to file/function
Addins
empty
Console
Terminal
~/Documents/empty/
Environment History Connections Git
Import Dataset
List
Global Environment
Environment is empty
Files Plots Packages Help Viewer
New Folder Delete Rename More
Home > Documents > empty
Name Size Modified
.gitignore 40 B May 18,
empty.Rproj 205 B May 18,

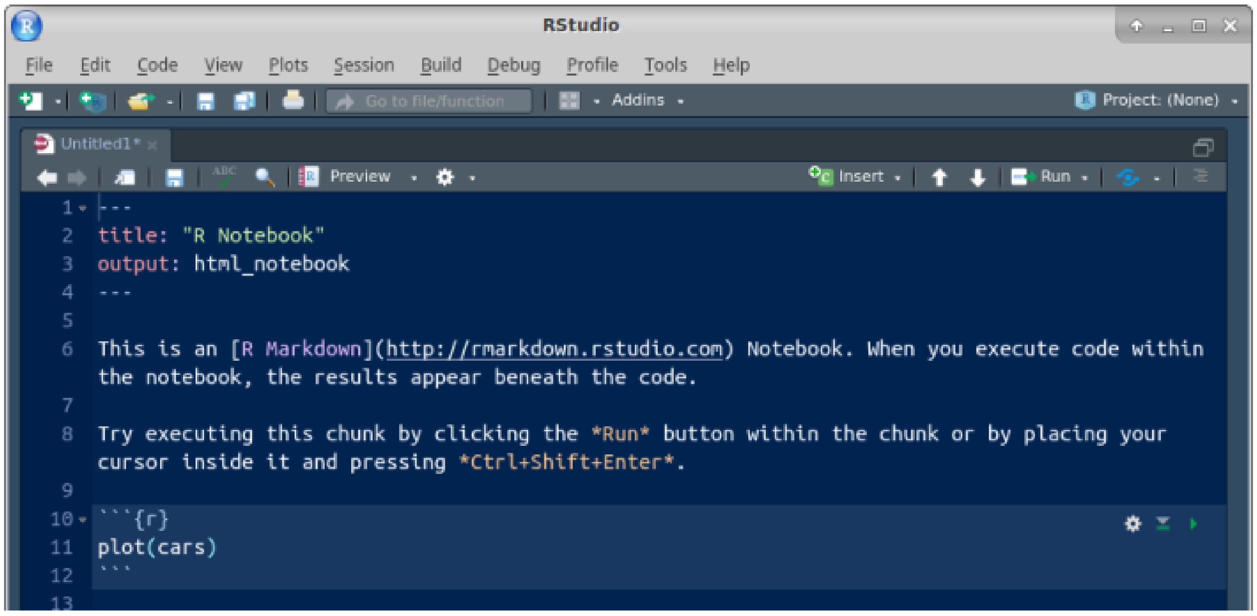
```
R version 3.4.4 (2018-03-15) -- "Someone to Lean On"
Copyright (C) 2018 The R Foundation for Statistical Computing
Platform: x86_64-pc-linux-gnu (64-bit)

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.

> 
```

It won't look exactly like that (for example, the background will probably be white) but 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 an R Notebook by selecting File, New File and R Notebook.

Solution

The first time, you'll be invited to "install some packages" to make the Notebook thing work. Let it do that by clicking Yes. After that, you'll have this:



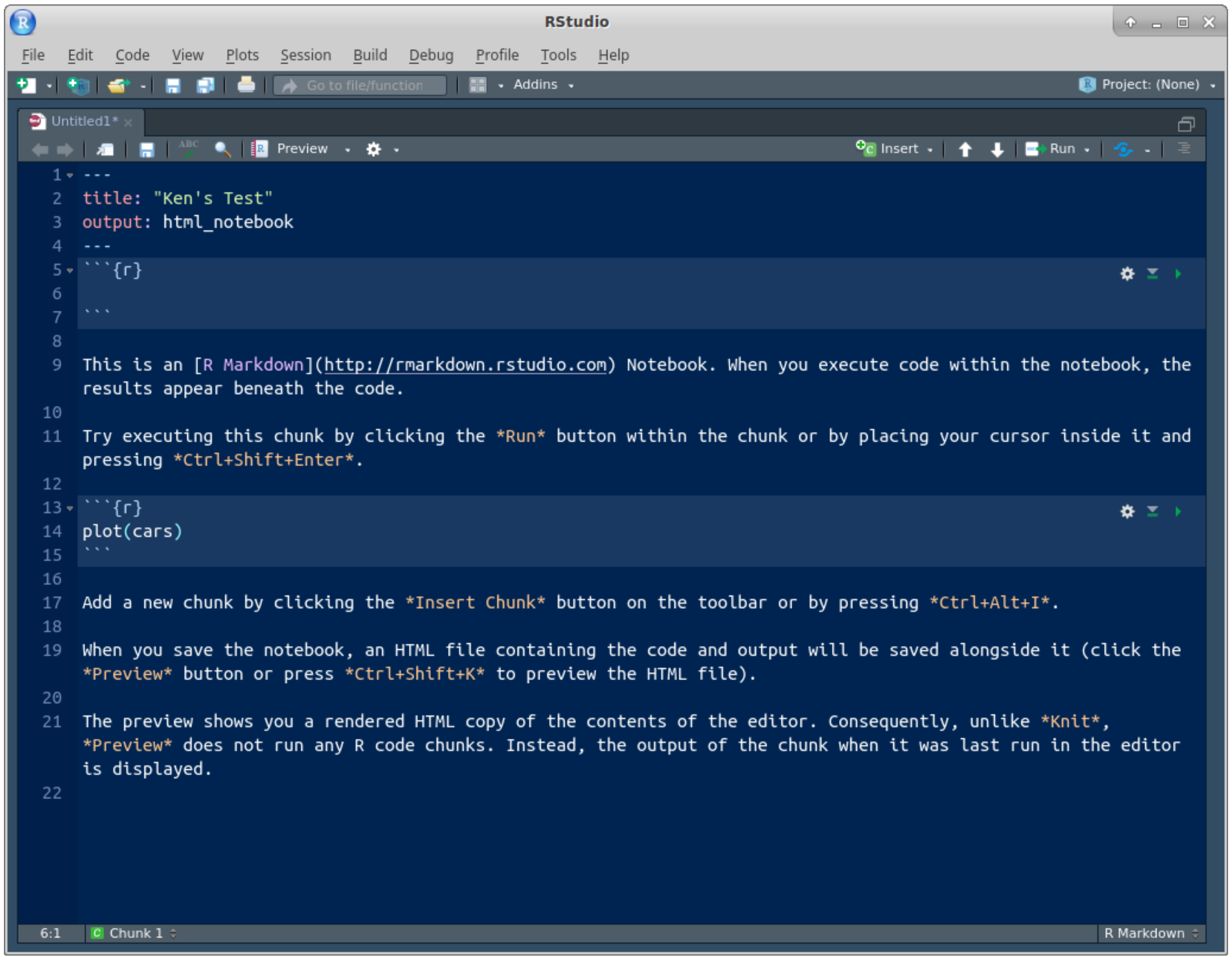
```
 1 ---
 2 title: "R Notebook"
 3 output: html_notebook
 4 ---
 5 
 6 This is an [R Markdown](http://rmarkdown.rstudio.com) Notebook. When you execute code within
   the notebook, the results appear beneath the code.
 7 
 8 Try executing this chunk by clicking the *Run* button within the chunk or by placing your
   cursor inside it and pressing *Ctrl+Shift+Enter*.
 9 
10 ```{r}
11 plot(cars)
12 ```
13
```

Find the Insert and Run buttons along the top of the R Notebook window. We'll be using them shortly. (The template notebook may or may not be maximized; it doesn't matter either way. You might see all four panes or as few as one. If you want to control that, select View at the top, then Panes, then either Show All Panes or Zoom Source, as you prefer. In the menus, you'll also see keyboard shortcuts for these, which you might find worth learning.)

(c) Change the title to something of your choosing. Then go down to line 5, click on the Insert button and select R. You should see a "code chunk" appear at line 5, which we are going to use in a moment.

Solution

Something like this:



~~~
---
title: "Ken's Test"
output: html_notebook
---
```{r}

~~~

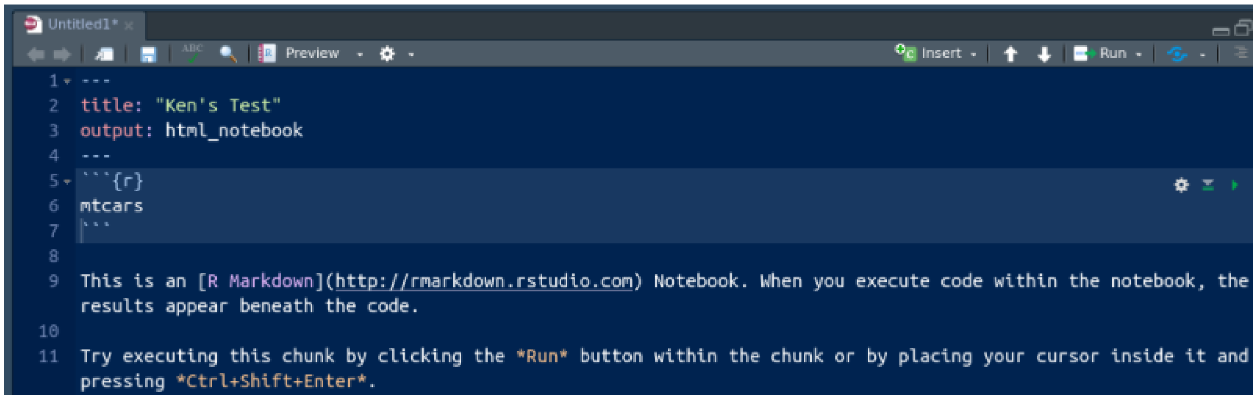
This is an [R Markdown](http://rmarkdown.rstudio.com) Notebook. When you execute code within the notebook, the results appear beneath the code.

Try executing this chunk by clicking the *Run* button within the chunk or by placing your cursor inside it and pressing *Ctrl+Shift+Enter*.

```{r}
plot(cars)
```

Add a new chunk by clicking the *Insert Chunk* button on the toolbar or by pressing *Ctrl+Alt+I*.

When you save the notebook, an HTML file containing the code and output will be saved alongside it (click the *Preview* button or press *Ctrl+Shift+K* to preview the HTML file).

The preview shows you a rendered HTML copy of the contents of the editor. Consequently, unlike *Knit*, *Preview* does not run any R code chunks. Instead, the output of the chunk when it was last run in the editor is displayed.
```

(d) Type the line of code shown below into the chunk in the R Notebook:

```
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.


Untitled1*
Preview
Insert
Run

```
---
title: "Ken's Test"
output: html_notebook
---
```{r}
mtcars
```

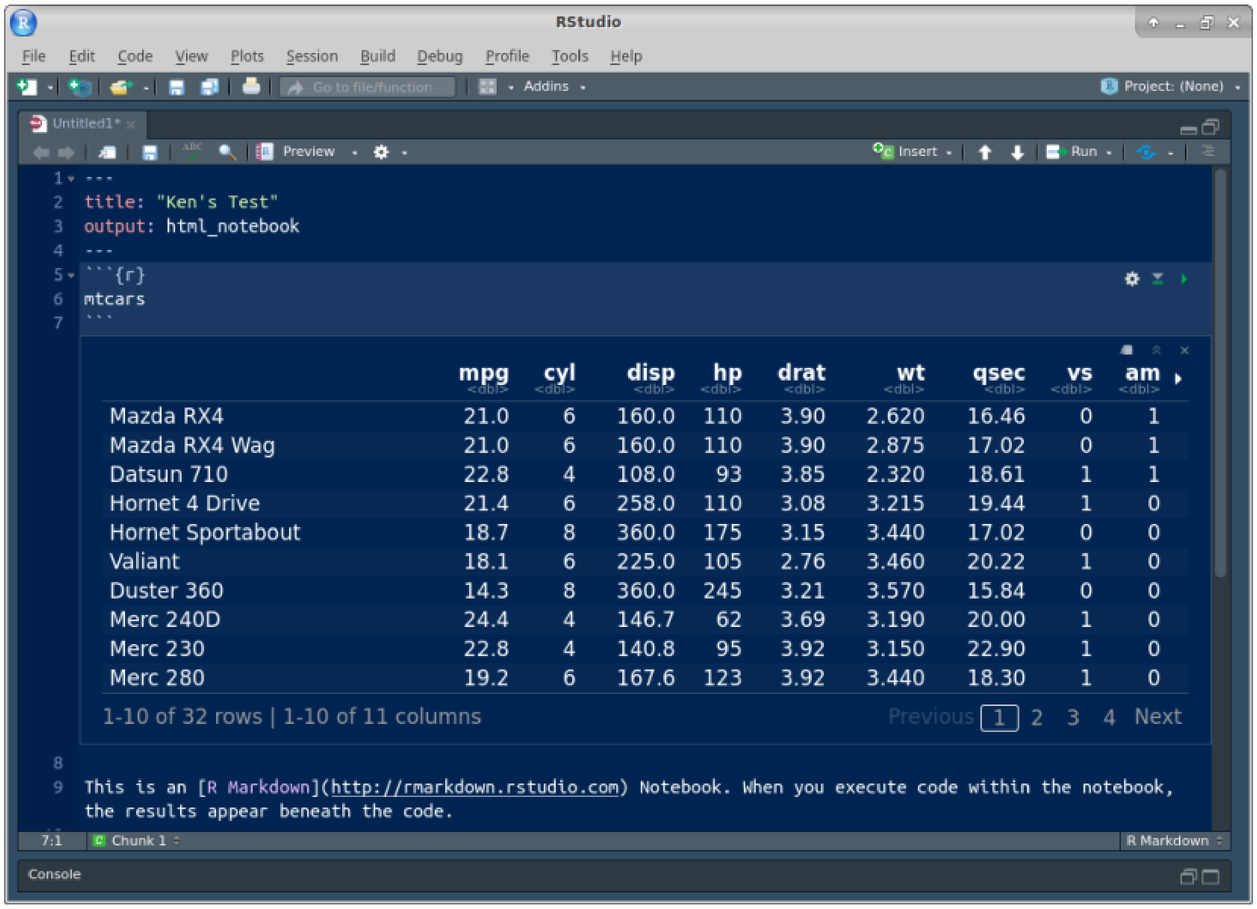
This is an [R Markdown](http://rmarkdown.rstudio.com) Notebook. When you execute code within the notebook, the results appear beneath the code.

Try executing this chunk by clicking the *Run* button within the chunk or by placing your cursor inside it and pressing *Ctrl+Shift+Enter*.
```

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 gear symbol, 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 will be the same).



~~~
---
title: "Ken's Test"
output: html_notebook
---
```{r}
mtcars
```
~~~

	mpg	cyl	disp	hp	drat	wt	qsec	vs	am
	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>
Mazda RX4	21.0	6	160.0	110	3.90	2.620	16.46	0	1
Mazda RX4 Wag	21.0	6	160.0	110	3.90	2.875	17.02	0	1
Datsun 710	22.8	4	108.0	93	3.85	2.320	18.61	1	1
Hornet 4 Drive	21.4	6	258.0	110	3.08	3.215	19.44	1	0
Hornet Sportabout	18.7	8	360.0	175	3.15	3.440	17.02	0	0
Valiant	18.1	6	225.0	105	2.76	3.460	20.22	1	0
Duster 360	14.3	8	360.0	245	3.21	3.570	15.84	0	0
Merc 240D	24.4	4	146.7	62	3.69	3.190	20.00	1	0
Merc 230	22.8	4	140.8	95	3.92	3.150	22.90	1	0
Merc 280	19.2	6	167.6	123	3.92	3.440	18.30	1	0

1-10 of 32 rows | 1-10 of 11 columns Previous 1 2 3 4 Next

```
This is an [R Markdown](http://rmarkdown.rstudio.com) Notebook. When you execute code within the notebook,
the results appear beneath the code.
```

This is a rectangular array of rows and columns, with individuals in rows and variables in columns, known as a "data frame". When you display a data frame in an R Notebook, 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:

```
mtcars
```

```
## # A tibble: 32 x 11
##       mpg   cyl  disp    hp  drat    wt  qsec    vs    am  gear  carb
##   * <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
##   1  21       6   160   110   3.9  2.62  16.5     0     1     4     4
##   2  21       6   160   110   3.9  2.88  17.0     0     1     4     4
```

```
##   3  22.8     4  108     93  3.85  2.32  18.6     1     1     4     1
##   4  21.4     6  258    110  3.08  3.22  19.4     1     0     3     1
##   5  18.7     8  360    175  3.15  3.44  17.0     0     0     3     2
##   6  18.1     6  225    105  2.76  3.46  20.2     1     0     3     1
##   7  14.3     8  360    245  3.21  3.57  15.8     0     0     3     4
##   8  24.4     4  147.    62  3.69  3.19  20       1     0     4     2
##   9  22.8     4  141.    95  3.92  3.15  22.9     1     0     4     2
## 10  19.2     6  168.   123  3.92  3.44  18.3     1     0     4     4
## # ... with 22 more rows
```

The top bit is the code, the bottom bit with the `##` the output. In this kind of display, you only see the first ten rows (by default).

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.

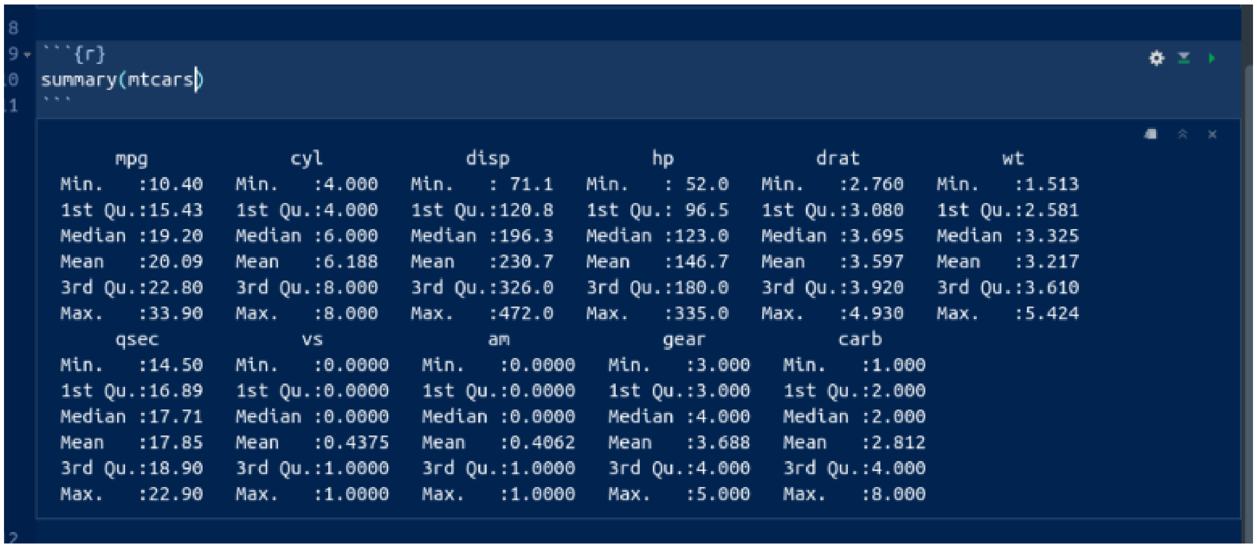
On the code chunk, the other symbols are the settings for this chunk (you have the choice to display or not display the code or the output or to not actually run the code). The second one, the down arrow, runs all the chunks prior to this one (but not this one).

The output has its own little buttons. 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.

Solution

This is what you should see:



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

```
      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.

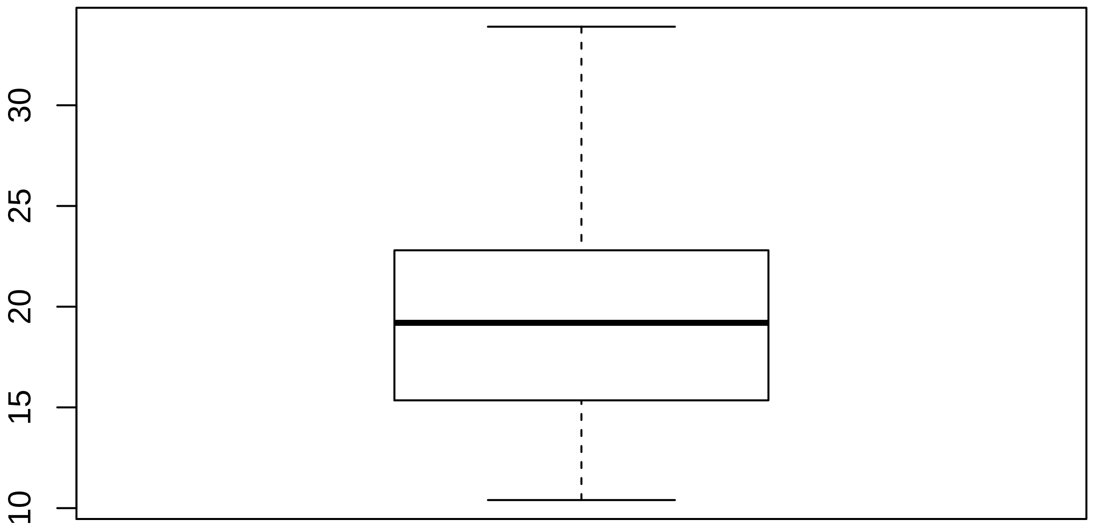
(In case you are wondering, the way without screenshots is obtained by *my* writing a notebook with code chunks and running them, so this output genuinely *is* obtained by running the code you see.)

(g) Let's make a boxplot of the gas mileage data. This is a "poor man's boxplot"; we'll see a nicer-looking way later. To do it this way, make another new code chunk, enter the code `boxplot(mtcars$mpg)` into it, and run the chunk.

Solution

This is what you should see:

```r
boxplot(mtcars$mpg)
```



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

(h) Some aesthetics to finish with: delete the template notebook (all the stuff you didn't type below your code chunks and output). Then 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.

Solution

My complete R Notebook is at http://www.utsc.utoronto.ca/~butler/c32/a0-notebook-1.Rmd. Take a look at it. I added one extra thing: my variable names have "backticks" around them. You'll see the effect of this in a moment. Backtick is on the key to the left of 1 and below Esc on your keyboard, along with a "squiggle" symbol that we'll be using later in the course.

(i) Save your notebook (the usual way with File and Save). This saves it *on the R Studio Cloud servers* (and not on your computer). This means that when you come back to R Studio Cloud later, even from another device, this notebook will still be available to you. Now click Preview. This produces a pretty HTML version of your notebook.

Solution

Note that the HTML document only contains output from the chunks you've run in the notebook, so it's up to you to run them there first.
My HTML document is at http://www.utsc.utoronto.ca/~butler/c32/a0-notebook-1.nb.html. Here's where you see the effect of the backticks: all the variable names are in `typewriter font` so that you can see they are variable names and not something else. If you want to try this notebook out yourself, you have a couple of options: (i) make a new R Notebook on R Studio Cloud and copy-paste the contents of my file (it's just text), or (ii) download my R Notebook onto your computer, and then upload it to R Studio Cloud. Look in the Files pane bottom right, and next to New Folder you should see Upload. Upload the file from wherever it got saved to when you downloaded it. Extra: if you're feeling ambitious, click the arrow to the right of Preview and select Knit to Word. The button changes to Knit with a ball of wool beside it. Now, when you "knit" the notebook, you get a Word document directly — look for it in the Files pane. If you want to, you can hand this kind of thing in (on later assignments), but you'll have to do a little work first: first, find it in your Files list, then click the checkbox to the left of it, then click More (with the gear, on the same line as New Folder and Upload), then select Export (and click Download). This will put a copy in your downloads folder on your computer, and you can open it from there. If you're feeling extra-ambitious, you can try Knit to PDF. This produces something that looks as if it was written in LaTeX, but actually wasn't. To make this work, if you have a `library(tidyverse)` line somewhere, as you probably will, find the code chunk it's in, and make it look like this:

"‘

```
library(tidyverse)
```

"‘

Then it will work. Extra extra: if you like the keyboard better than the mouse, R Studio has a lot of keyboard shortcuts. Two that are useful now: control-alt-i inserts a code chunk where the cursor is, and control-shift-enter runs the code chunk that the cursor is in, if it is in one. (Mac users, "command" instead of "control" in both cases.) I use these two a lot.

(j) Optional extra: practice handing in your previewed R notebook, 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

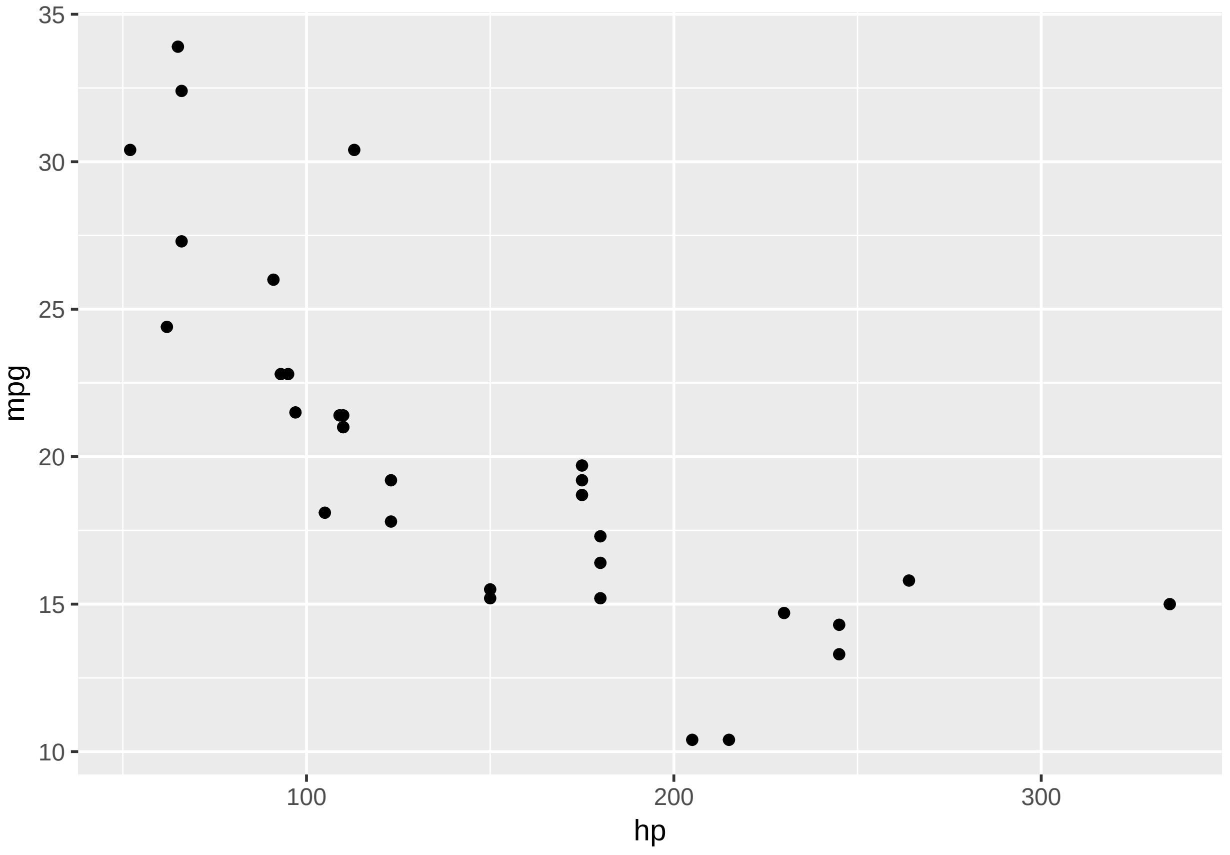
There are two steps: download the HTML file onto your computer, and then handing it in on Quercus. To download: find the HTML file that you want to download in the Files pane bottom right. There should be two files starting with the same thing, eg. `test1.Rmd`, which is the notebook you wrote, and `test1.nb.html`, which is the previewed version of it, and is the one you want to download. (The `test1` part is the name *you* chose when you saved it.) 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`, log in and navigate to this course. Click Assignments. Click (the title of) Assignment 0. There is a big blue Submit 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. If you want to try this again, you can Re-submit Assignment as many times as you like. (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 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()
```



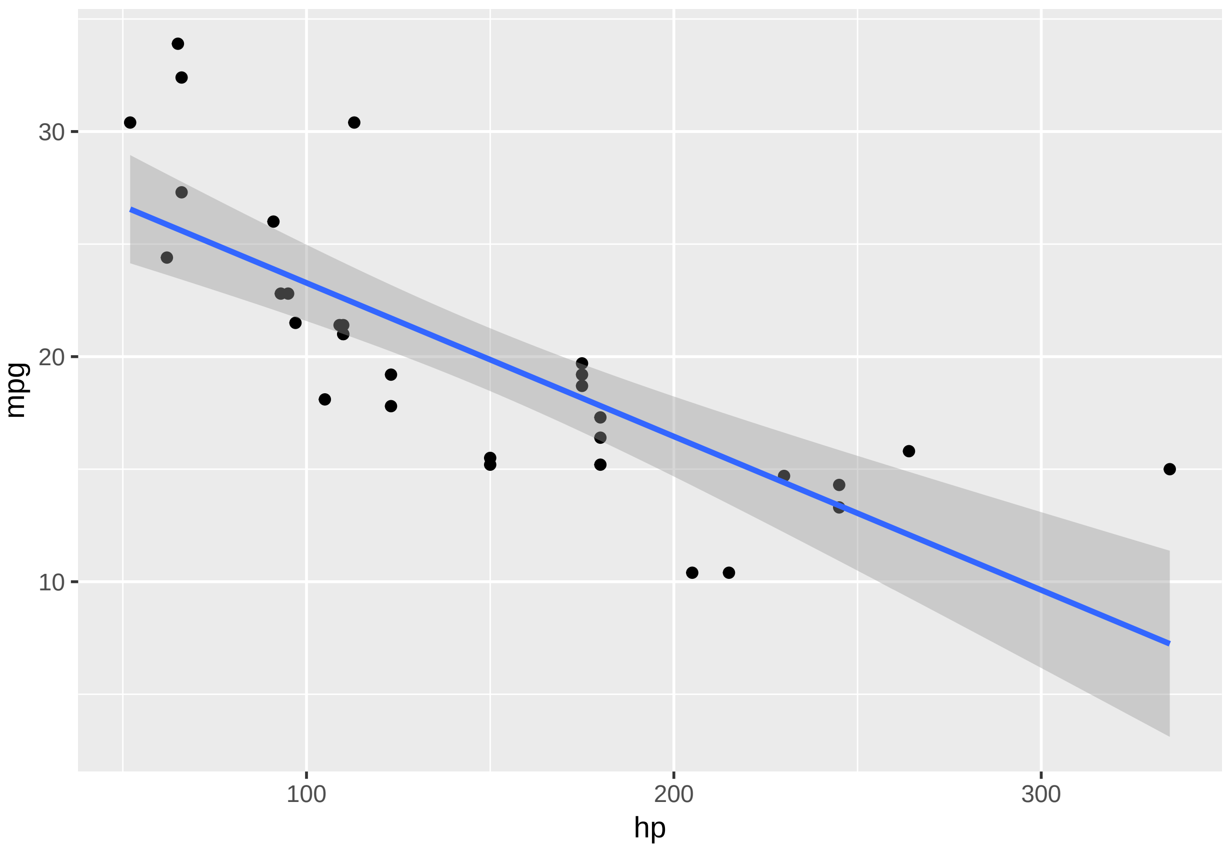
$

%$ %$

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")
```



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

Data file operations

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 https://www.utsc.utoronto.ca/~butler/c32/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 R Notebook. Leave the first four lines, 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:

```
library(tidyverse)
my_url="https://www.utsc.utoronto.ca/~butler/c32/jumping.txt"
rats=read_delim(my_url," ")
```

```
## Parsed with column specification:
## cols(
##   group = col_character(),
##   density = col_integer()
## )
```

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.

(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>     <int>
##  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
## # ... with 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.
```

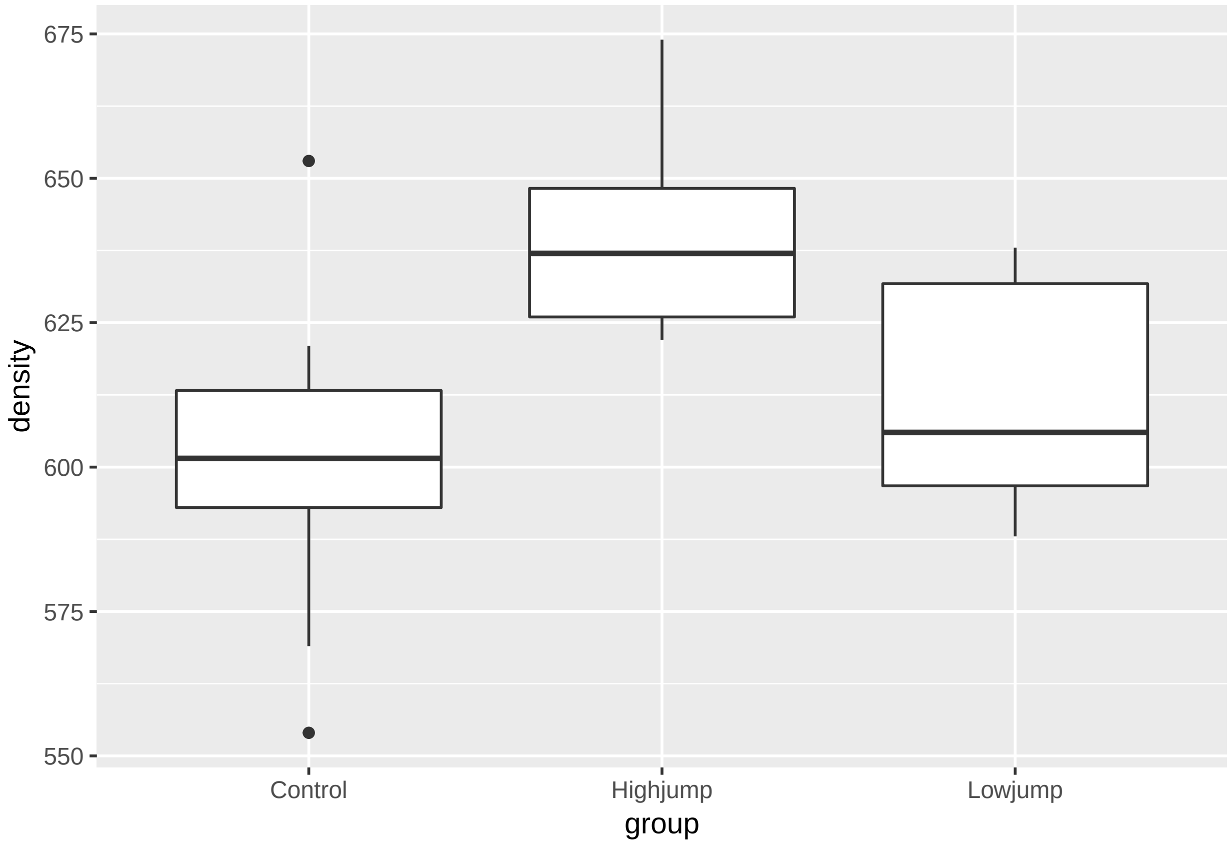
The mean bone density is clearly highest for the high jumping group, and not much different between the low-jumping and control groups.

(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.

Chapter 3

Reading in data and drawing some graphs

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 http://www.utsc.utoronto.ca/~butler/c32/ojuice.txt. 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)
```

```
## -- Attaching packages ---------------------------------------------------------------------------

## v ggplot2 3.0.0     v purrr   0.2.5
## v tibble  1.4.2     v dplyr   0.7.6
## v tidyr   0.8.1     v stringr 1.3.1
## v readr   1.1.1     v forcats 0.3.0

## -- Conflicts ------------------------------------------------------------------------------------
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

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:endnote{I say "better" because otherwise the `read_delim` 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://www.utsc.utoronto.ca/~butler/c32/ojuice.txt"
juice=read_delim(url," ")
```

```
## Parsed with column specification:
## cols(
##   run = col_integer(),
##   sweetness = col_double(),
##   pectin = col_integer()
## )
```

`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
##    <int>     <dbl>  <int>
##  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
## # ... with 14 more rows
```

I appear to have all the data. If you want further convincing, click Next a couple of times (on yours) 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. 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)
```

```
## Parsed with column specification:
## cols(
##   x = col_integer(),
##   y = col_integer()
## )
```

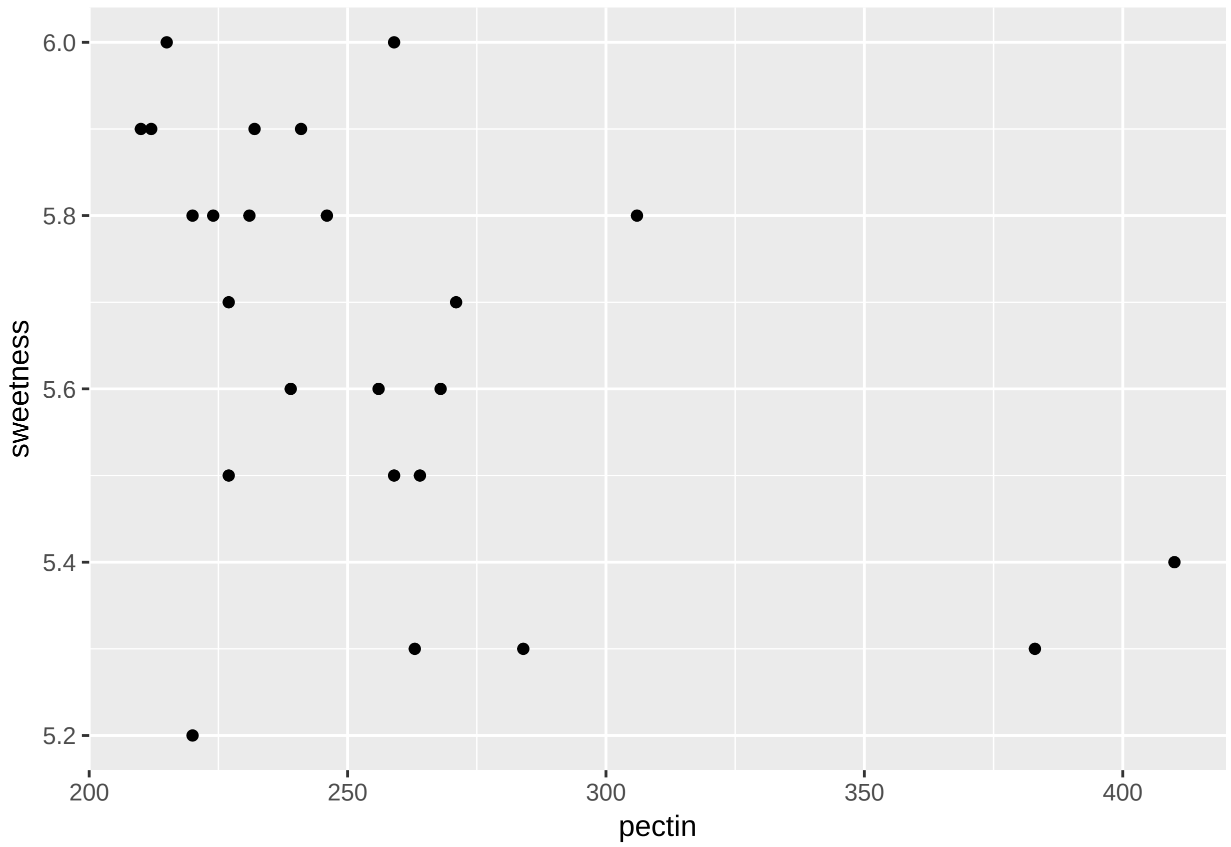
```
## # A tibble: 3 x 2
##       x     y
##   <int> <int>
## 1     1     2
## 2     3     4
## 3     5     6
```

(d) The juice manufacturer was interested in whether there was a relationship between sweetness and pectin. To assess this, draw a scatterplot. Does it look as if there is any kind of a relationship? (I think `sweetness` is the outcome variable and `pectin` is explanatory, so draw your scatterplot appropriately.)

Solution

This requires a `ggplot` plot. You can go back and look at the lecture notes to figure out how to make a scatterplot: the "what to plot" is the x-axis and y-axis variables, with the response on the y-axis (starting with a data frame to get the variables from), and the "how to plot" is `geom_point` to plot the points:

```
ggplot(juice,aes(x=pectin,y=sweetness))+geom_point()
```

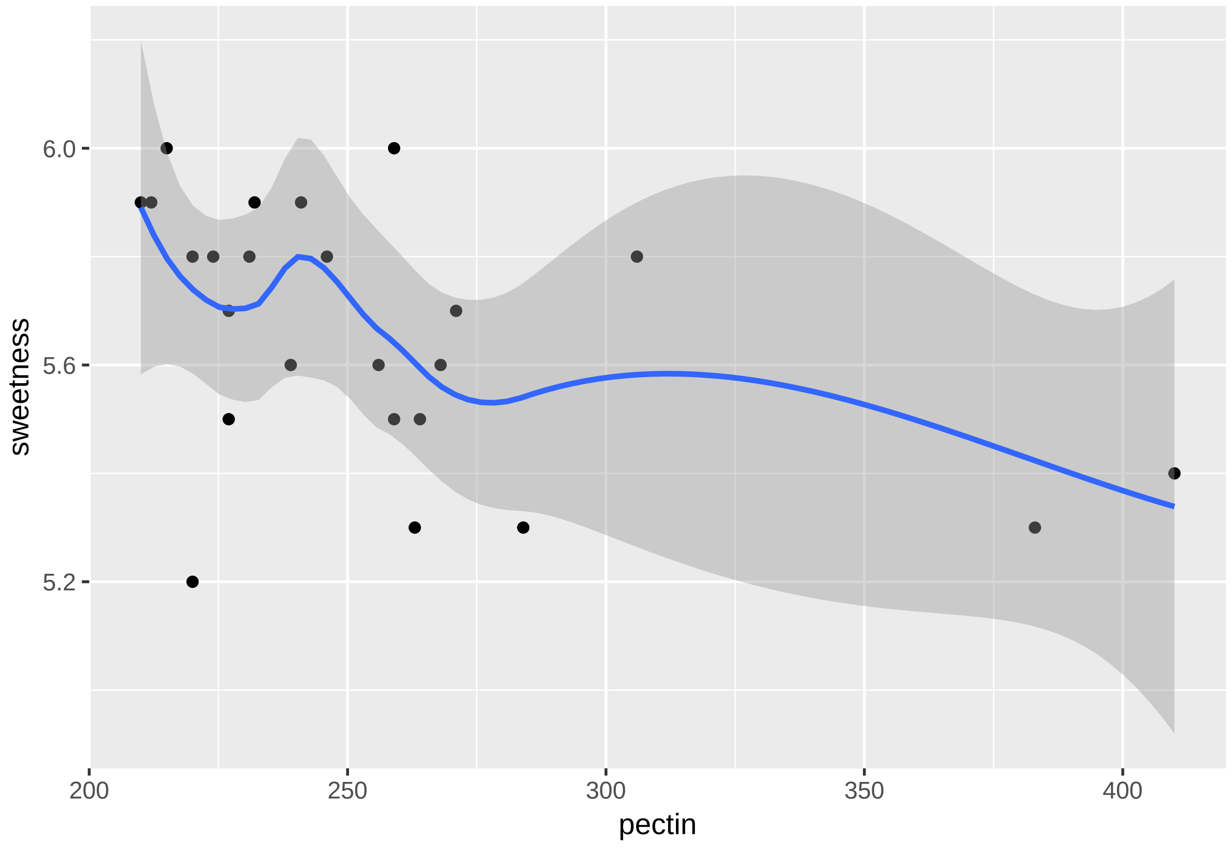


It looks to me as if there is a negative relationship: as pectin goes up, sweetness tends to go *down*. The trend appears to go top left to bottom right.

Having said that, I'm wondering how much of the apparent trend is caused by those two observations bottom right with pectin over 350. If you take those away, the trend seems to me to be a lot less convincing. As an extra, you could add a smooth trend to the plot:

```
ggplot(juice,aes(x=pectin,y=sweetness))+geom_point()+geom_smooth()
```

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



The smooth trend is kind of downhill, but not very convincing.

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 http://www.utsc.utoronto.ca/~butler/c32/soap.txt.

(a) Read the data into R. Display the data. There should be 27 rows. Are there?

Solution

Read directly from the URL, most easily:

```
url="http://www.utsc.utoronto.ca/~butler/c32/soap.txt"
soap=read_delim(url," ")
```

```
## Parsed with column specification:
## cols(
##   case = col_integer(),
##   scrap = col_integer(),
##   speed = col_integer(),
##   line = col_character()
## )
```

```
soap
```

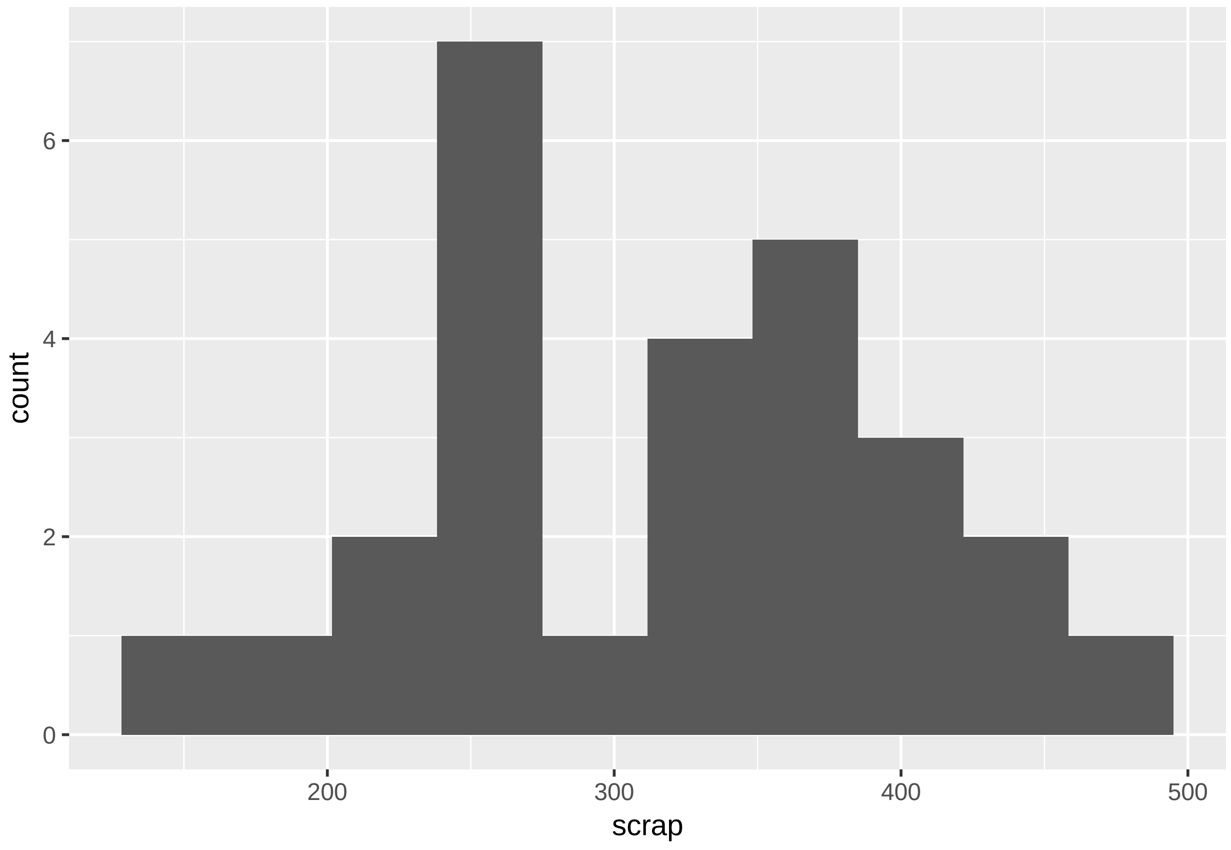
```
## # A tibble: 27 x 4
##     case scrap speed line
##    <int> <int> <int> <chr>
##  1     1   218   100 a
##  2     2   248   125 a
##  3     3   360   220 a
##  4     4   351   205 a
##  5     5   470   300 a
##  6     6   394   255 a
##  7     7   332   225 a
##  8     8   321   175 a
##  9  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

27 rows. `line`, which is either `a` or `b`, was correctly deduced to be text.

(b) Obtain a histogram of the `scrap` values, using 10 bins for your histogram.

Solution

```
ggplot(soap,aes(x=scrap))+geom_histogram(bins=10)
```

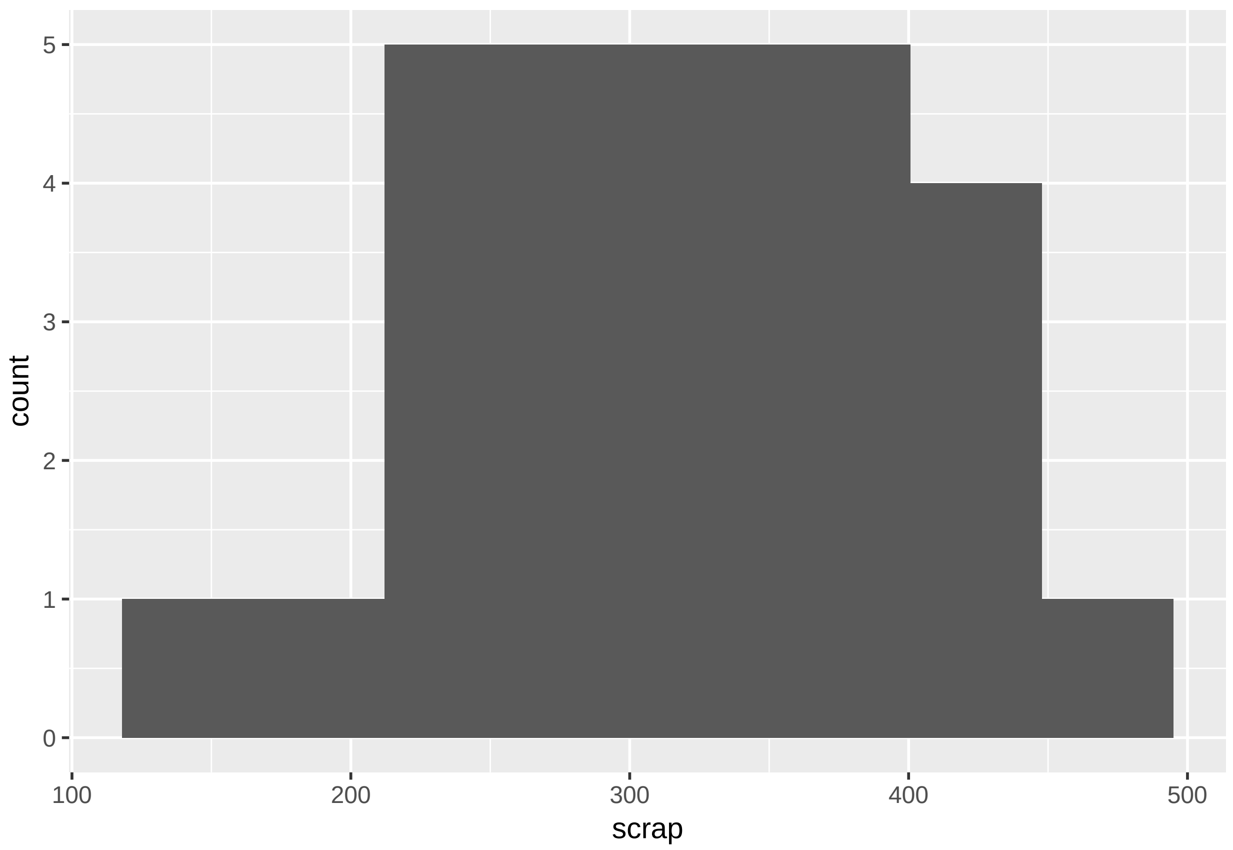


(c) Comment briefly on the shape of the histogram. Is it approximately symmetric, skewed to the left, skewed to the right or something else? (By "comment briefly" I mean "say in a few words why you gave the answer you did.")

Solution

I would call this "bimodal". There are two peaks to the histogram, one around 250 and one around 370, with a very small frequency in between (the bar around 300). Apart from the bimodality, there is no particular evidence for a long tail on either end, so I don't think you could otherwise call it anything other than symmetric. Having said that (this is going beyond the question), the way a histogram looks can depend on the bins you choose to draw it with. This is 8 bins rather than 10:

```
ggplot(soap,aes(x=scrap))+geom_histogram(bins=8)
```

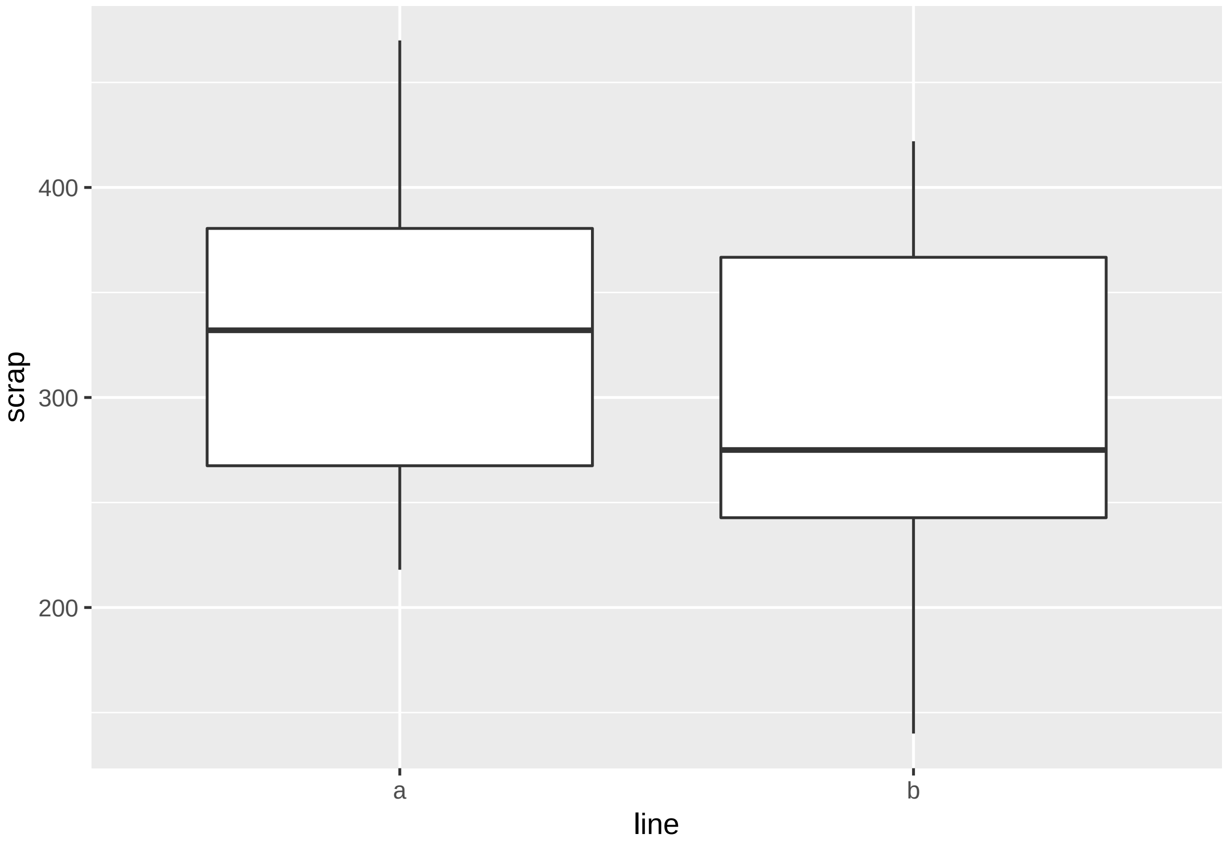


The middle low-frequency bin has gone, and this one just looks symmetric, with a kind of "flat top".

(d) Make side-by-side boxplots of scrap values for each production line.

Solution

```
ggplot(soap,aes(x=line,y=scrap))+geom_boxplot()
```



One categorical, one quantitative variable, so boxplots make sense.

(e) Do you think your boxplot says that there are differences in the amount of scrap produced by the two production lines, or not? Explain briefly.

Solution

I would say that there *is* a difference between the two production lines, with line A producing an average (median) of about 330 and line B producing a median of about 275. But you could also make the case that, although the medians are rather different, there is a lot of variability and hence a lot of overlap between the two boxplots, and therefore that there is not a "substantial" difference. I would say that either of those answers are good emph{if you back them up with proper reasons}. This is going to be a common theme in this course: I am going to ask you to make a decision and support it, where the reasons you provide are often more important than the decision you make. You might be wondering whether the medians, or means, since there is no serious skewness here and definitely no outliers, are "significantly different". This is inference, which we will come to later, but a preview looks like this:

```
t.test(scrap~line,data=soap)
```

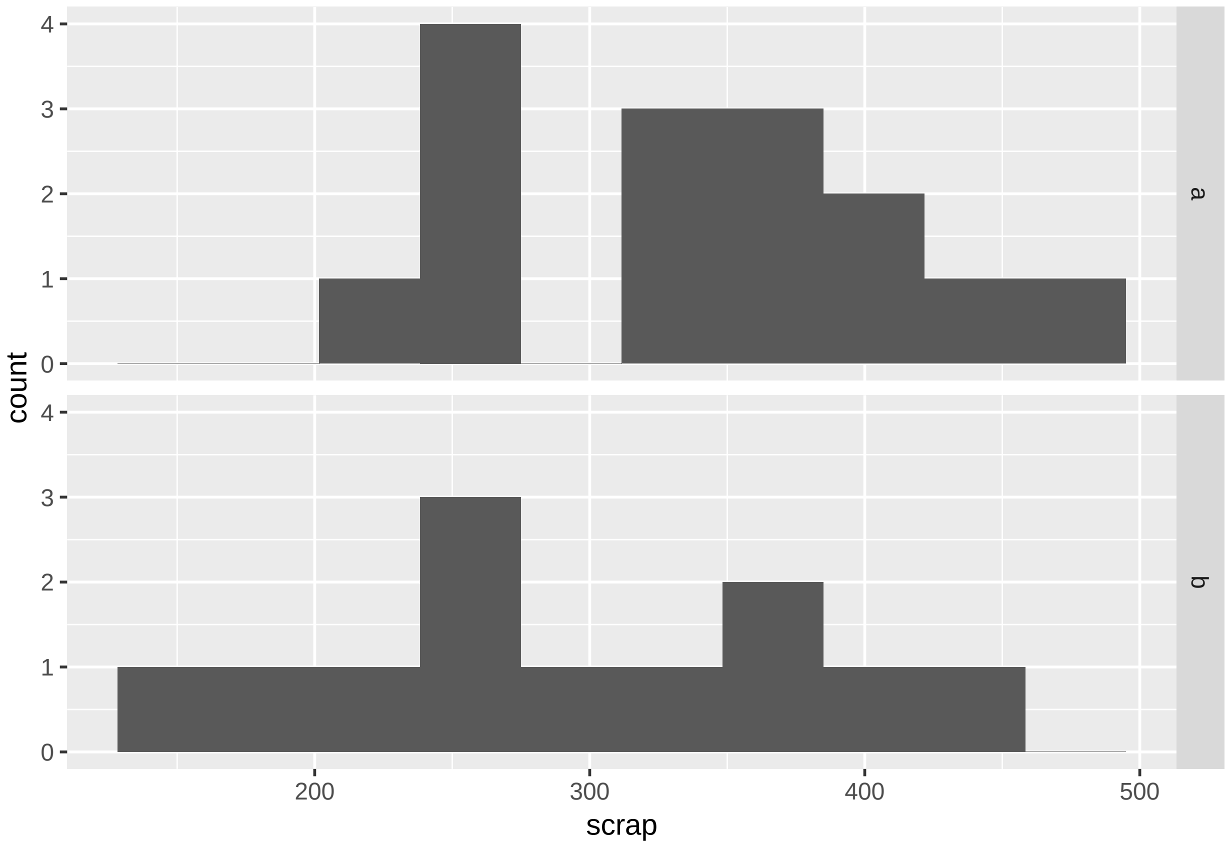
```
##
##  Welch Two Sample t-test
##
## data:  scrap by line
## t = 1.2493, df = 21.087, p-value = 0.2253
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -26.97888 108.21222
## sample estimates:
## mean in group a mean in group b
##        333.5333        292.9167
```

They are not: the P-value of 0.22 is not anywhere near as small as 0.05, so we can't reject the null hypothesis that the two lines have equal mean amount of scrap.

Rusty on this stuff? Don't worry; we're going to come back to it later in the course.

I was also wondering about something else: that bimodal histogram. Could that be explained by the scrap values being two different production lines being mixed together? One way to understand that is to have two separate histograms, one for each line, side by side, which is what facetting does. There is an extra wrinkle here that I explain afterwards:

```
ggplot(soap,aes(x=scrap))+geom_histogram(bins=10)+facet_grid(line~.)
```



I could have used `facet_wrap`, but that would have put the histograms side by side, and I wanted them one above the other (for ease of comparison, since they'll be on the same scale). `facet_grid` is like `facet_wrap`, but offers you more control over where the facets go: you can arrange them above and below by a variable, or left and right by a variable. Whatever is facetting the plots up and down (on the y axis) goes before the squiggle, and whatever facets them left and right goes after. If there is nothing separating the facets in one direction, here horizontally, the variable is replaced by a dot.

In some ways, `facet_grid` is also *less* flexible, because the facets have to be arranged up/down or left/right by a variable. That worked here, but if you think back to the Australian athletes, where there were ten different sports, it was `facet_wrap` that did the right thing, arranging the sports along rows *and* columns to produce a pleasing display.

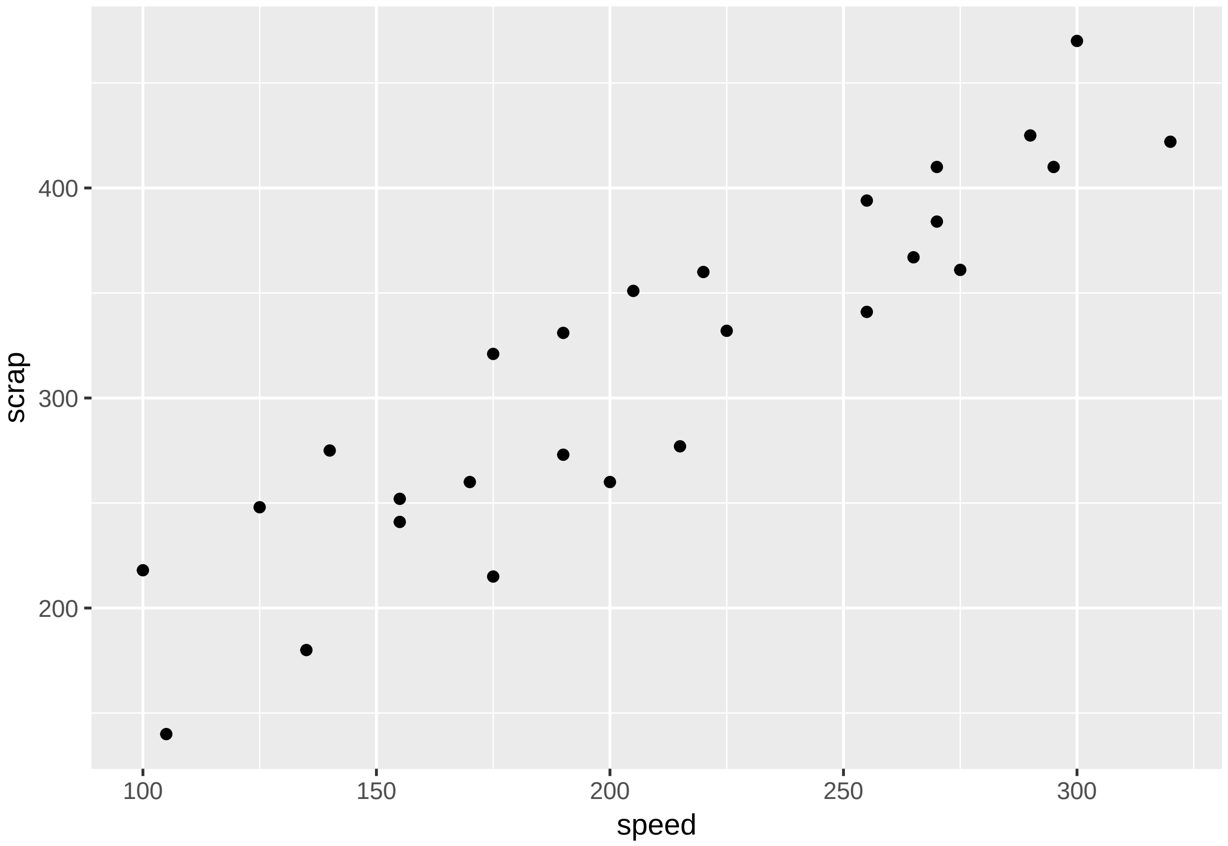
All right, that bimodality. I was expecting that the scrap values from one line would be centred about one value and the scrap values from the other line would be centred about a different value, with a gap in between. But that's not what happened at all: the line B values are all over the place, while it's the line A values that are actually bimodal all by themselves. I'm not sure whether that really means anything, since the data sets are pretty small, but it's kind of interesting.

(f) We started out with the suspicion that if the line was run faster, there would be more scrap. We haven't assessed this yet. Draw a scatter plot with `scrap` on the y axis and `speed` on the x axis.

Solution

Same mechanism as before:

```
ggplot(soap,aes(x=speed,y=scrap))+geom_point()
```

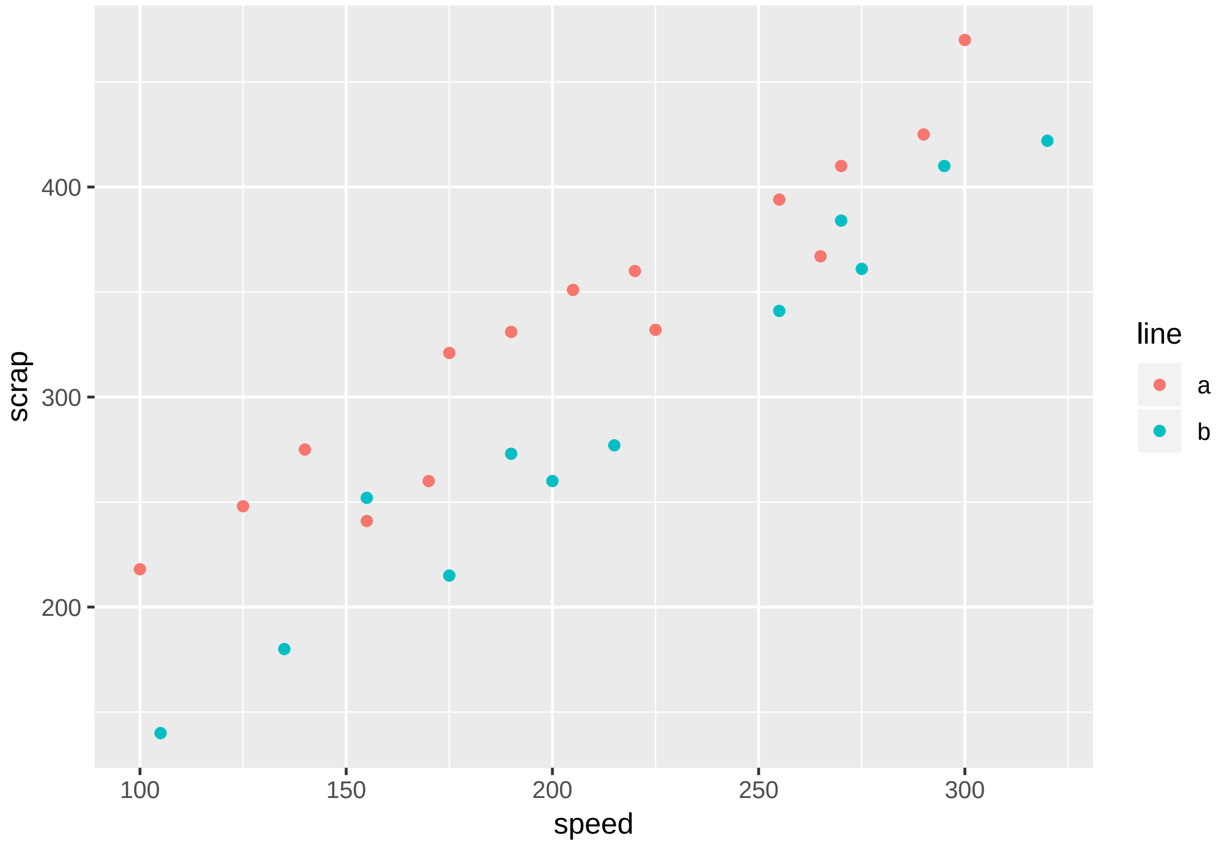


(g) What do you think is the most important conclusion from your plot of the previous part? Describe your conclusion in the context of the data.

Solution

There seems to be a pretty evident upward trend, apparently linear, which means that if the speed of the production line is higher, the amount of scrap produced is also higher. My last sentence was meant to remind you that ‘`there is an upward trend'' is *not a complete answer*: we are concerned with what that upward trend tells us about the data. This, in other words, confirms the suspicion expressed in the question, which was therefore a rather large clue: more speed tends to go with more scrap. That was as far as I wanted you to go: there seems to be an association with speed, and there might be an association with`line`that turned out not to be statistically significant. What we haven't done is to assess the relationship between speed and scrap for *each* production line. To do that, we want to plot the scrap-speed points distinguished for each production line.`ggplot`makes that easy: you add a`colour`endnote{If you are concerned about the spelling: the guy who wrote`ggplot`is from New Zealand, where they spell "colour" the same way we do. However, if you want to use`color=`‘, that works too.} to say what you want to distinguish by colour. This is two quantitative variables and one categorical variable, if you want to think of it that way:

```
ggplot(soap,aes(x=speed,y=scrap,colour=line))+geom_point()
```

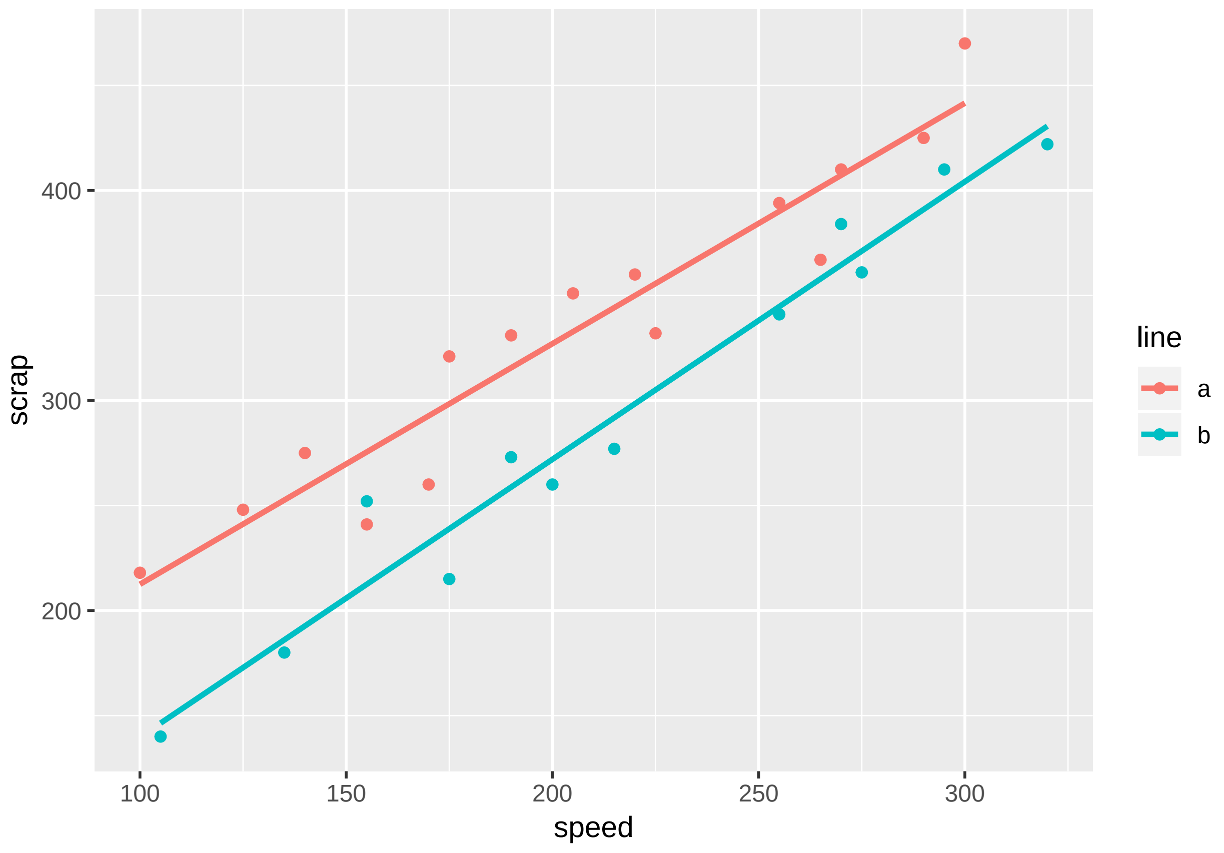


Notice that we get a legend, automatically.

What is interesting about this one is the red dots are mostly at the top (for any given speed), and the blue dots are mostly at the bottom. That seems to mean that *when we account for speed*, there is a difference between lines.

I want to show you one more embellishment, which is to put the regression lines on the plot for each group separately. This is where `ggplot` is so nice, since I just have to add one thing:

```
ggplot(soap,aes(x=speed,y=scrap,colour=line))+
geom_point()+geom_smooth(method="lm",se=F)
```



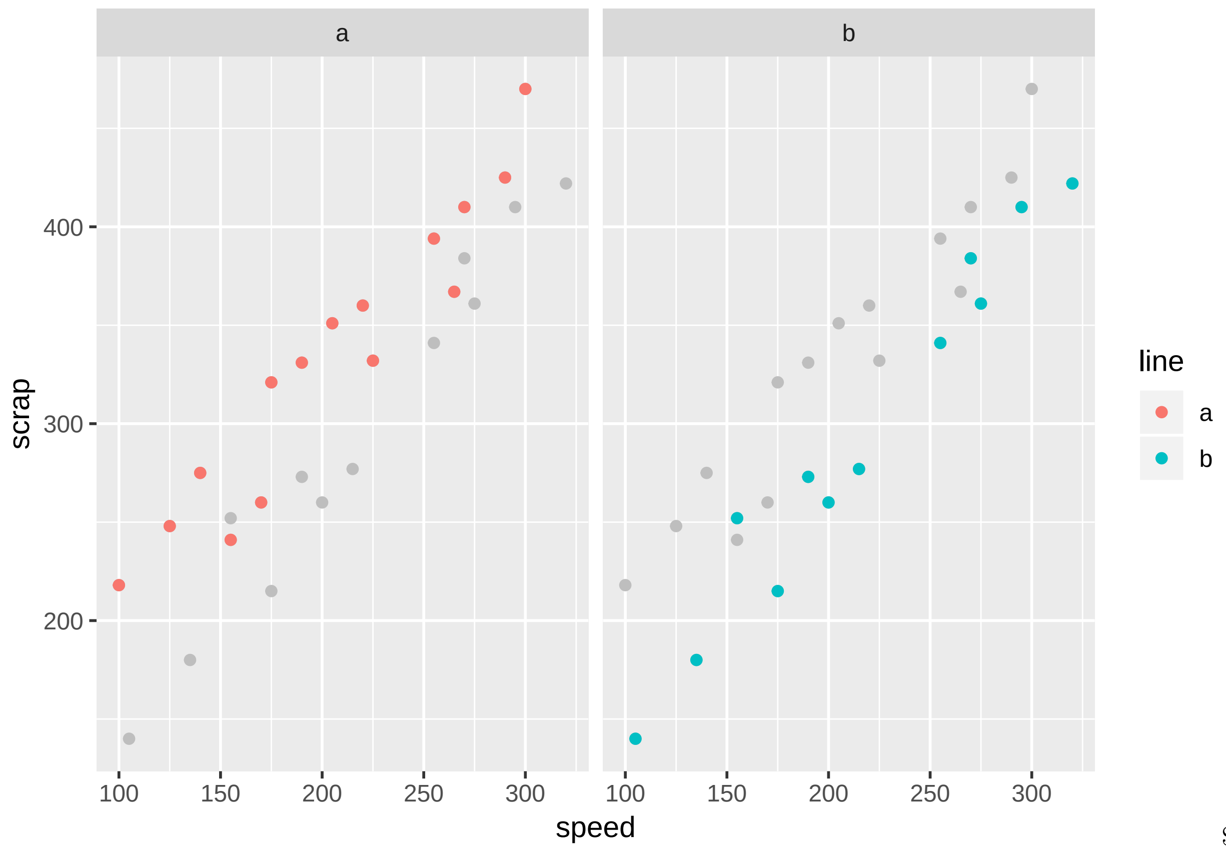
The points and lines have come out in different colours, without our having to think too hard.

Both lines show an upward trend, with about the same slope, which means that regardless of line, increasing the speed goes with increasing the scrap by the same amount. The fact that the red line is above the blue one, however, suggests that production line A produces more scrap at the same speed than production line B.

From a management point of view, there is an interesting dynamic at work: if you run the production line faster, you'll produce more bars of soap, but you'll produce more scrap as well. The crucial thing for the people in the supervisor's office is how much raw material is used per bar of soap, and if you make the soap bars faster, you might use more raw material, which will eat into your profits (from one angle), but you will also have more bars of soap to sell.

Here's another way to see the same thing. I'm *definitely* not expecting you to follow the code, but you can admire the result!

```
soap2=soap %>% select(-line)
ggplot(soap,aes(x=speed,y=scrap))+
geom_point(data=soap2,colour="grey")+
geom_point(aes(colour=line))+facet_wrap(~line)
```



$

The idea is that we plot all the points in grey (to "put them in the background") and then in each plot we plot the points again, *coloured, for the group we are looking at*: line A in the left, line B on the right. This is another way of seeing that line A has more scrap than line B, given the speed at which the line was being run. (I discovered this technique only yesterday. I think the code is remarkably concise for what it does.)

The logic of the code is: begin{itemize} item create a new data frame that contains everything in `soap` except for `line` item make a scatter plot of all the points in this new data frame, coloured grey item plot the points again (from the original data frame), coloured by which production line they're from item produce a separate scatterplot for each production line. end{itemize}

The trick about creating the new data frame was to enable plotting of all points regardless of group on each subplot ("facet" in `ggplot` terminology), as well as the ones that come from that production line.

I don't expect you to be able to follow all the details of the code below, either, but I would like you to try and get the logic. What we do is a regression predicting `scrap` from *two* things: `speed` and production `line`. The results we get are these:

```
scrap.1=lm(scrap~speed+line,data=soap)
summary(scrap.1)
```

```
##
## Call:
## lm(formula = scrap ~ speed + line, data = soap)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -39.557 -14.161  -0.121  17.518  33.953
##
## Coefficients:
```

```
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)  80.41099   14.54379   5.529 1.10e-05 ***
## speed         1.23074    0.06555  18.775 7.48e-16 ***
## lineb       -53.12920    8.21003  -6.471 1.08e-06 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 21.13 on 24 degrees of freedom
## Multiple R-squared:  0.9402, Adjusted R-squared:  0.9352
## F-statistic: 188.6 on 2 and 24 DF,  p-value: 2.104e-15
```

The P-values for `speed` and `line` are the second and third things in the last column, $7 times 10^{-16}$ and $1 times 10^{-6}$ respectively. These are both very strongly significant, in contrast to the two-sample *t*-test where `line` was not significant.

So does production line make a difference or not?

The plot says that it does, and the meaning of model `scrap.1` just above is that emph{`speed` affects scrap when you account for `line`}, and emph{`line` affects scrap when you account for speed}. (In the two-sample *t*-test above we didn't account for speed at all, since the various speeds were all mixed up.)

There is a moral to this story, which I would like you to get even if you don't get any of the statistics: emph{if a variable makes a difference}, it should be in your model and on your graph,endnote{Meaning that the graph should contain all three variables, `speed`, `scrap` and `line`.} because it enables you to get better (more precise) conclusions about your other variables. Here, there really is a difference between the production lines, but the *t*-test was too much of a blunt instrument to unearth it (because `speed` made a difference as well).

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 http://www.utsc.utoronto.ca/~butler/c32/global.csv 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. How many rows and columns does it have?

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://www.utsc.utoronto.ca/~butler/c32/global.csv"
shipments=read_csv(url)
```

```
## Parsed with column specification:
## cols(
##   warehouse = col_character(),
##   size = col_integer(),
##   cost = col_double()
## )
```

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>     <int> <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
```

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

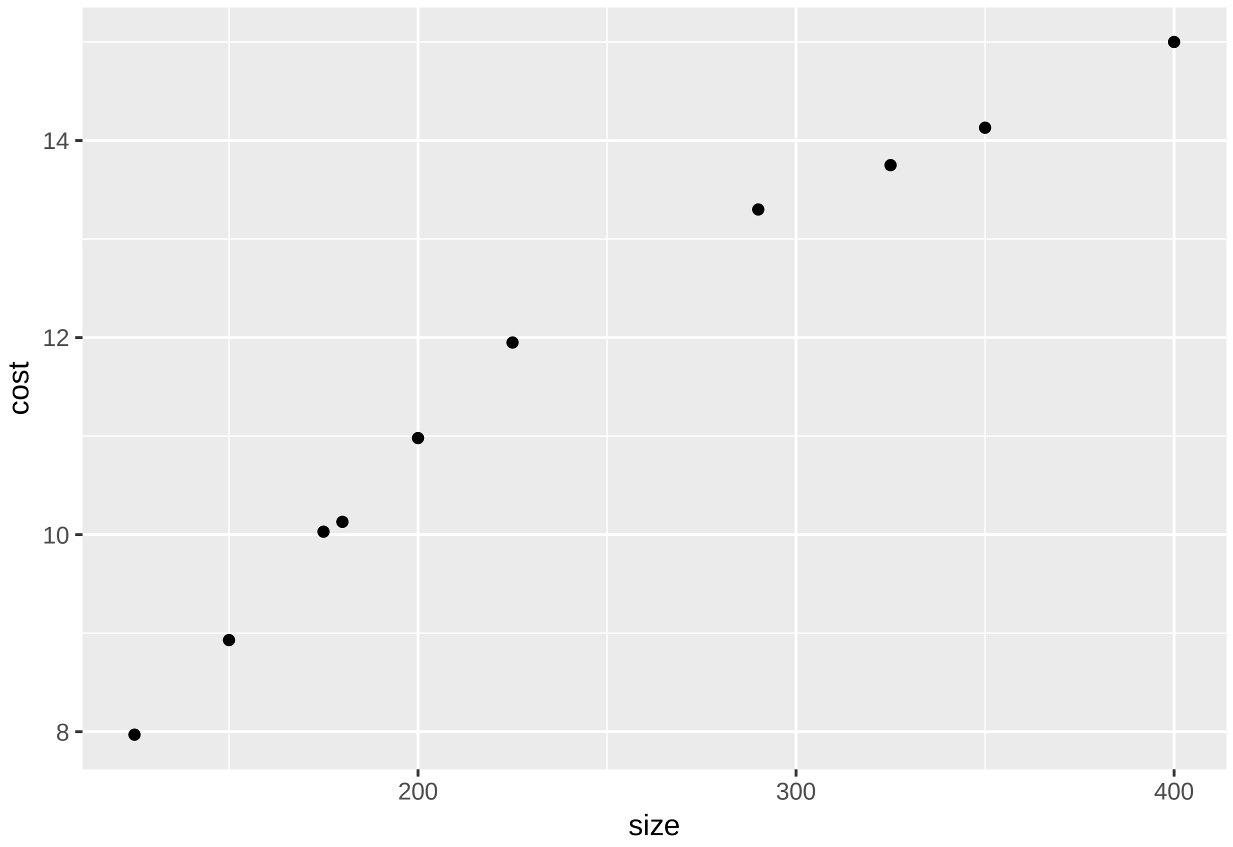
That is, there were 10 shipments recorded, and for each of them, 3 variables were noted: the size and cost of the shipment, and the warehouse it was handled at.

(b) Make a scatterplot of the cost of handling each shipment as it depends on the shipment's size.

Solution

The wording of the question says that cost is the response and so belongs on the *y*-axis. To make the plot, `ggplot` with an `x=` and a `y=` in the `aes` (the `what to plot'' part), and a `geom_point()` after (the`how to plot it"):

```
ggplot(shipments,aes(x=size,y=cost))+geom_point()
```



As a matter of coding, there are usually *two* brackets to close after the `aes`, the one that begins the `ggplot` and the one that begins the `aes`.

(c) What kind of relationship do you see on the scatterplot? Do you think a straight line would describe it appropriately? Explain briefly.

Solution

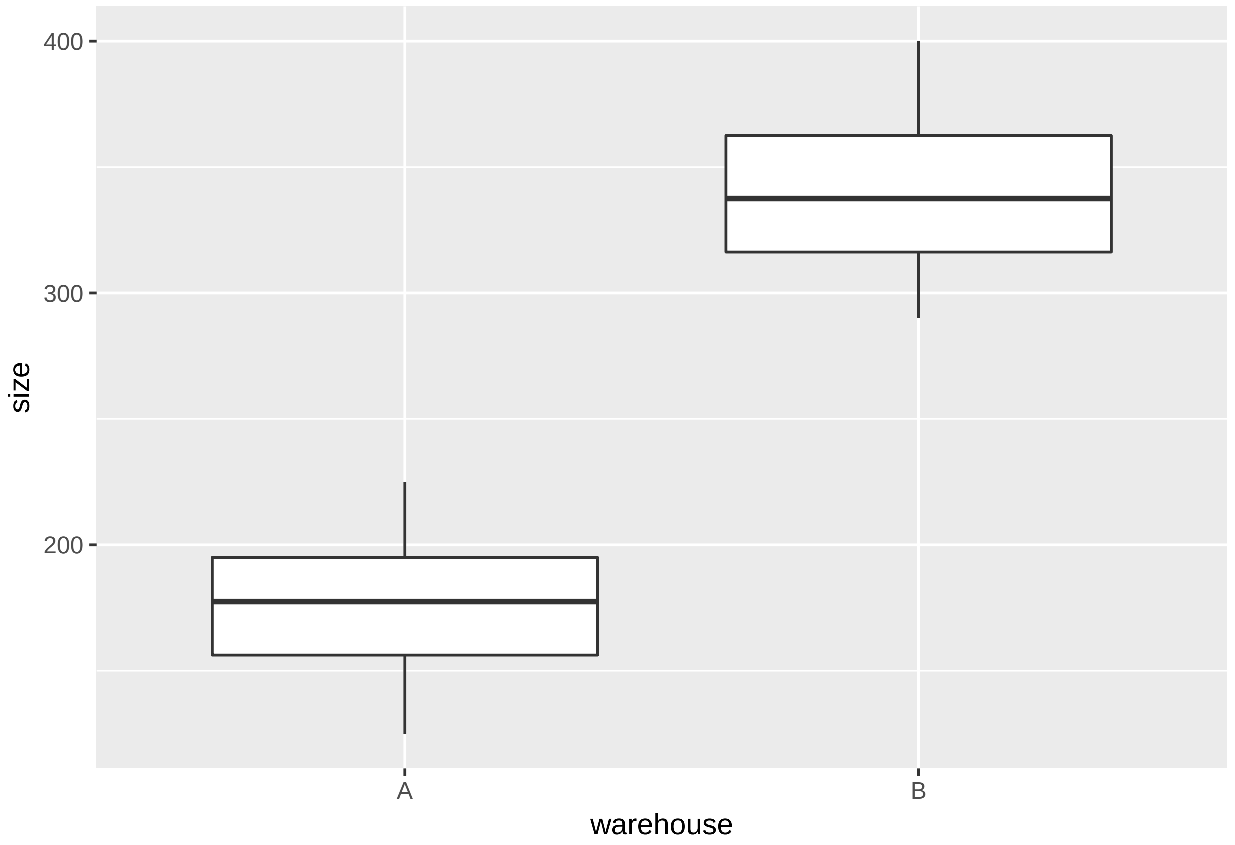
I see an upward trend: a shipment with larger `size` costs more to handle. If you look carefully at the scatterplot, you see that the cost of handling a small shipment goes up fairly steeply with its size, but the cost of handling a large shipment, while it still increases with `size`, does not increase so fast. Thus having one straight line to describe the whole relationship would not work so well. The relationship is actually two different straight lines joined end-to-end, which we will explore later, but if you think the relationship is curved, I'll accept that. The point is to get at the idea that the rate of increase is not constant.

(d) When a shipment comes in, the cost of handling it is not known. A decision is made about which warehouse to send it to, and then, after it is handled, the cost is recorded. What do you think determines which warehouse an incoming shipment goes to? Provide a graph to support your answer.

Solution

The veiled hint in the question is that the decision must depend on `size`, since it cannot depend on `cost`. So we have one quantitative variable `size` and one categorical variable `warehouse`, which suggests drawing boxplots:

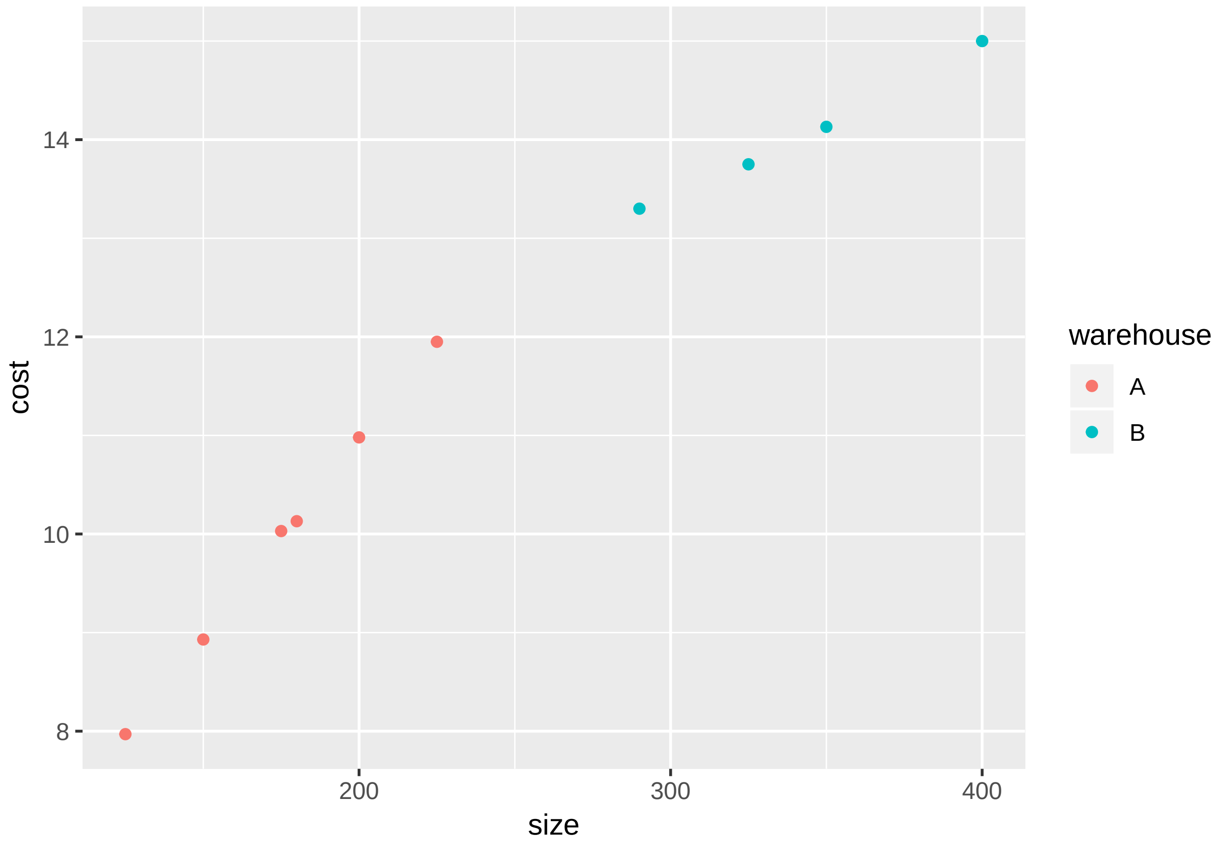
```
ggplot(shipments,aes(x=warehouse,y=size))+geom_boxplot()
```



Well, there's the answer right there. When the shipment has small `size`, it goes to warehouse A, and when it's large, it goes to Warehouse B. We know this because *all* the shipments smaller than about 250 (thousand parts) went to A and *all* the shipments larger than that went to B. (If you want to provide a number to delineate "small" and "large", anything between the largest A, about 225, and the smallest B, about 290, will do.)

Another way to think about this is to add something to the scatterplot you drew before. The obvious thing is to make the two warehouses different colours:

```
ggplot(shipments,aes(x=size,y=cost,colour=warehouse))+
geom_point()
```



As a point of technique, you can split lines of code to make them fit on your screen. You can do this as long as emph{the code that ends the line must be incomplete}, so that R knows more is to come. Ending a line with a pipe symbol, or, as here, with one of the pluses in the middle of a `ggplot`, will work. If you put the plus on the start of the next line, you'll get a blank plot, because R thinks you're done plotting. Try it and see.

Anyway, this plot tells exactly the same story: the small shipments (in size or cost) go to Warehouse A and the large ones to Warehouse B. But we don't know cost when the decision is made about which warehouse to send a shipment to, so the decision must be based on `size`.

In the place where I got these data, it said "larger shipments are sent to Warehouse B, since this warehouse has specialized equipment that provides greater economies of scale for larger shipments". That is to say, very large shipments are more expensive to handle, but not as expensive as you might think.endnote{This is the same idea that it costs more to ride the GO bus from UTSC to York U than it does to ride from UTSC to Scarborough Town, but if you work out how much it costs per kilometre, the longer journey costs less per km. As of when I'm writing this, \$5.30 for the 7.2 km to Scarborough Town and \$6.75 for the 38 km to York. That's quite an economy of scale, isn't it?} That makes sense with our scatterplot, because the *slope* for larger shipments is less than for smaller shipments.

When we get to regression later, we'll see what happens if we fit a straight line to data like these, and how to tell whether we really ought to be able to do better with a different form of relationship. There is also a trick to fit what is called a "piecewise linear regression", which has one slope for small shipment sizes, a different (smaller) slope for large ones, and joins up in the middle. But that's well beyond our scope now.

Chapter 4

Data exploration

```r
library(tidyverse)
```

```
## -- Attaching packages ---------------------------------------------------------------------------

## v ggplot2 3.0.0     v purrr   0.2.5
## v tibble  1.4.2     v dplyr   0.7.6
## v tidyr   0.8.1     v stringr 1.3.1
## v readr   1.1.1     v forcats 0.3.0

## -- Conflicts ------------------------------------------------------------------------------------
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

Question title

The data in file http://www.utsc.utoronto.ca/~butler/c32/ncbirths.csv 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 (rather than downloading the file, unless you are working offline):

```r
myurl="http://www.utsc.utoronto.ca/~butler/c32/ncbirths.csv"
bw=read_csv(myurl)
```

```
## Parsed with column specification:
## cols(
##   `Father Age` = col_integer(),
##   `Mother Age` = col_integer(),
##   `Weeks Gestation` = col_integer(),
##   `Pre-natal Visits` = col_integer(),
##   `Marital Status` = col_integer(),
##   `Mother Weight Gained` = col_integer(),
##   `Low Birthweight?` = col_integer(),
##   `Weight (pounds)` = col_double(),
##   `Premie?` = col_integer(),
##   `Few Visits?` = col_integer()
## )
```

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 Gestatio~ `Pre-natal Visi~
##           <int>        <int>            <int>            <int>
##  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
## # ... with 490 more rows, and 6 more variables: `Marital Status` <int>,
## #   `Mother Weight Gained` <int>, `Low Birthweight?` <int>, `Weight
## #   (pounds)` <dbl>, `Premie?` <int>, `Few Visits?` <int>
```

or you can take a "glimpse" of it:

```
glimpse(bw)
```

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

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.

What you'll notice is that the variables have *spaces* in their names, which will require special handling later. These outputs show you what to do about those spaces in variable names: 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.)

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, so we have to surround these variables with backticks too.)

(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, prematureness and length of pregnancy? (You might have to make guesses based on the names of the variables.)

Solution

I do indeed have 500 observations on 10 variables ("several"). (If you don't have several variables, check to see that you didn't use `read_delim` or something by mistake.) After the '`500 observations of 10 variables'' line(s) in each case, you see all the variables by name, with what type of values they haveendnote{these are mostlyintor "integer".}, and the first few of the values.endnote{Other possible variable types arenumfor (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 alsolgl`' 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.}

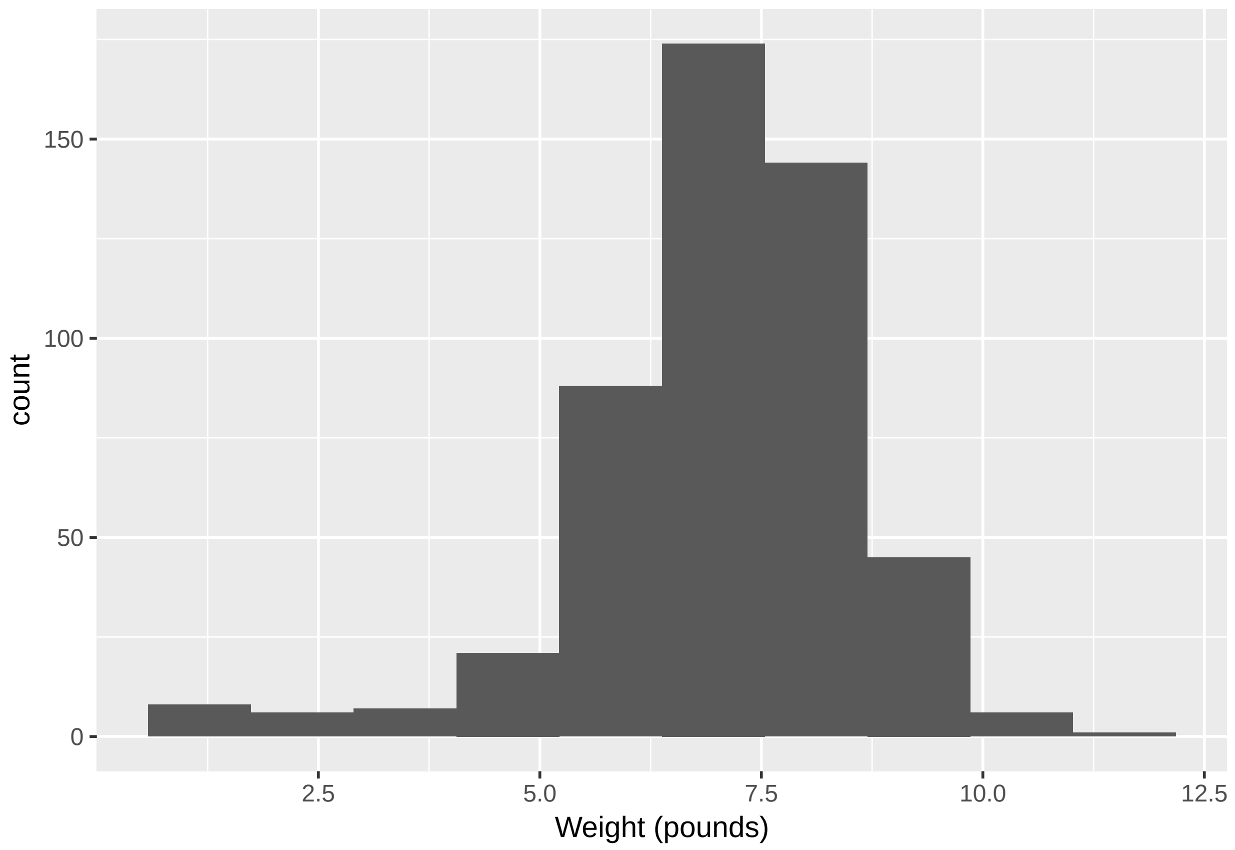
The variable verb=`Weight (pounds)`= is the birthweight (in pounds), verb=`Premie?`= is 1 for a premature baby and 0 for a full-term baby, and verb=`Weeks Gestation`= is the number of weeks the pregnancy lasted. I've put backticks around each of those to remind us that that's how we'll need to refer to them when we use them later.endnote{The backticks look different from each other for annoying technical reasons, but they're all backticks.}

(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.)

Solution

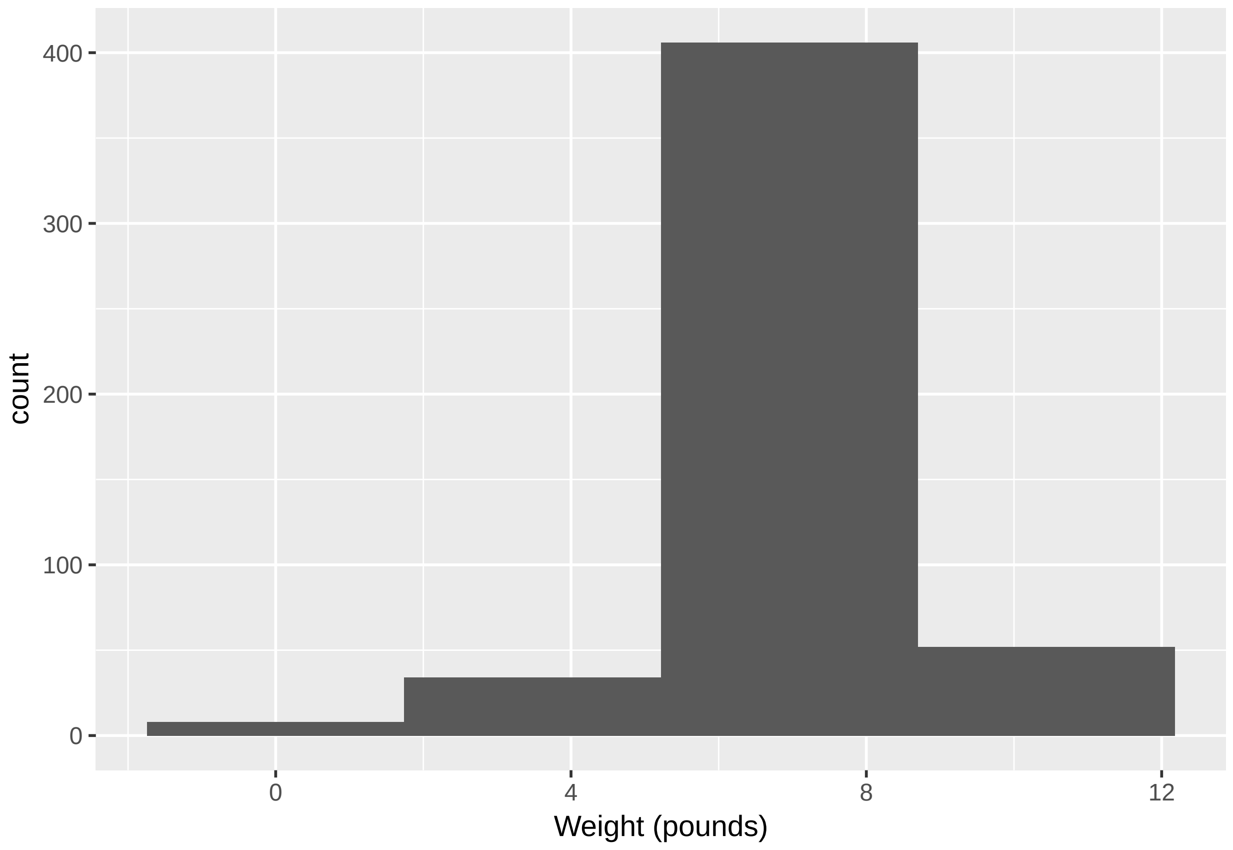
You'll have seen that I often start with 10 bins, or maybe not quite that many if I don't have much data, and this is a decent general principle. That would give

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



which is perfectly acceptable. 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:

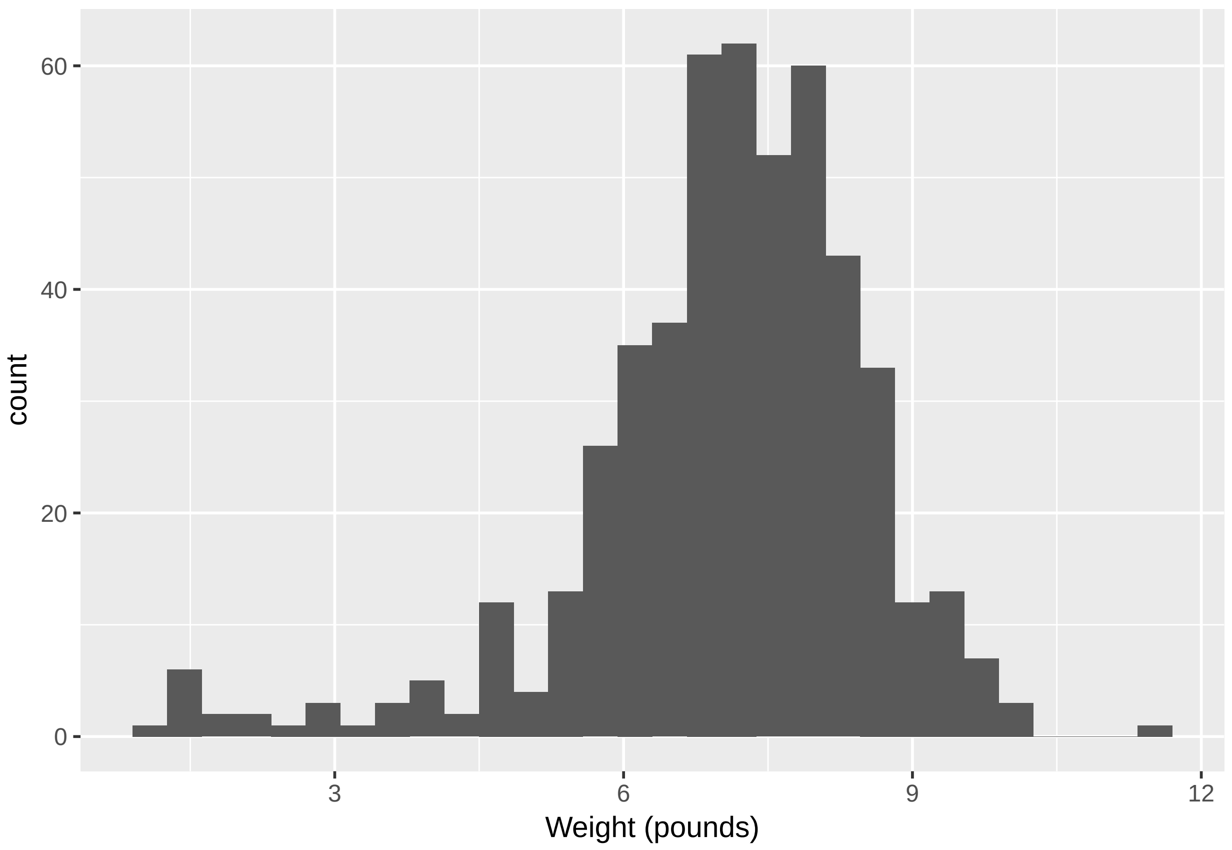
```
ggplot(bw,aes(x=`Weight (pounds)`))+geom_histogram(bins=4)
```



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

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: http://ritsokiguess.site/docs/2017/06/08/histograms-and-bins/. 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 href{https://robjhyndman.com/papers/sturges.pdf}{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:

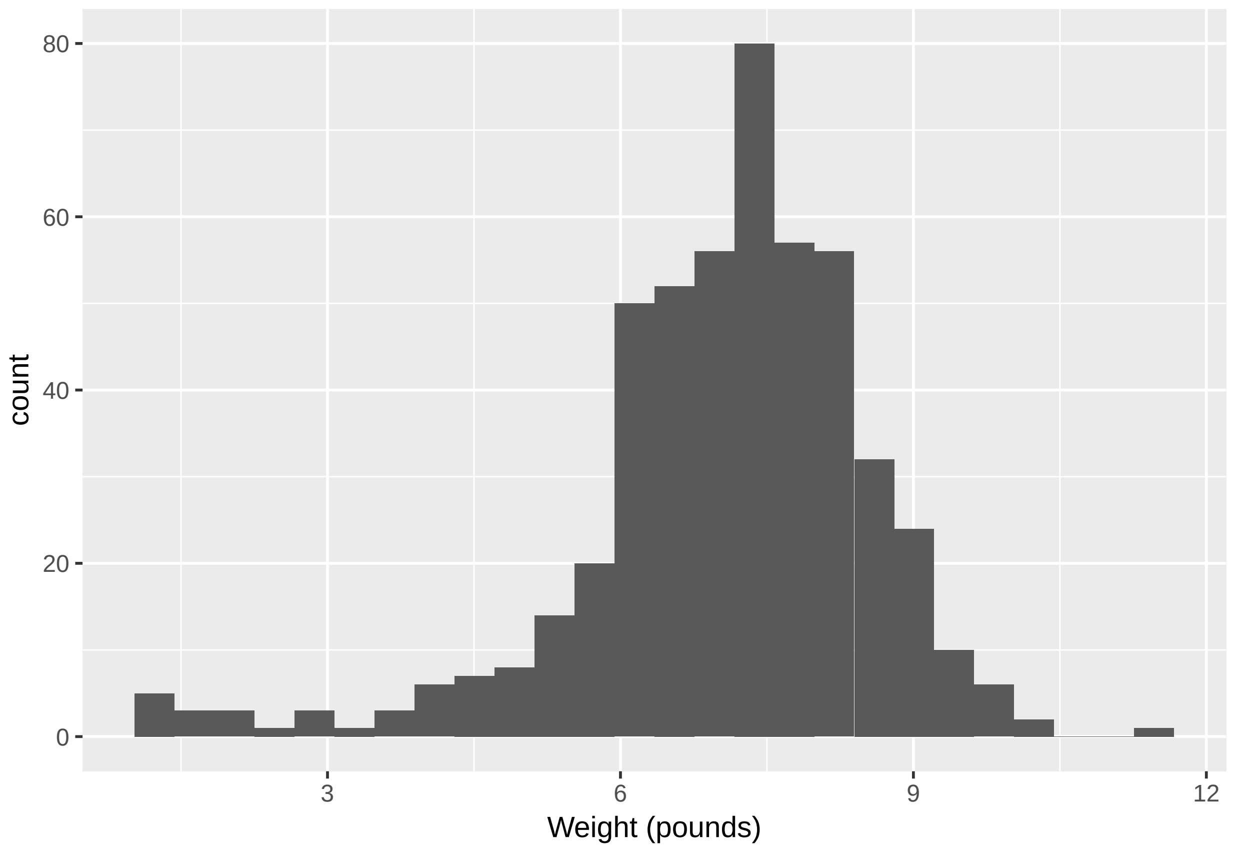
$$w = 2(IQR)n^{-1/3}$$

The nice thing about this is that it uses the interquartile range, so it won't be distorted by outliers. verb=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
```

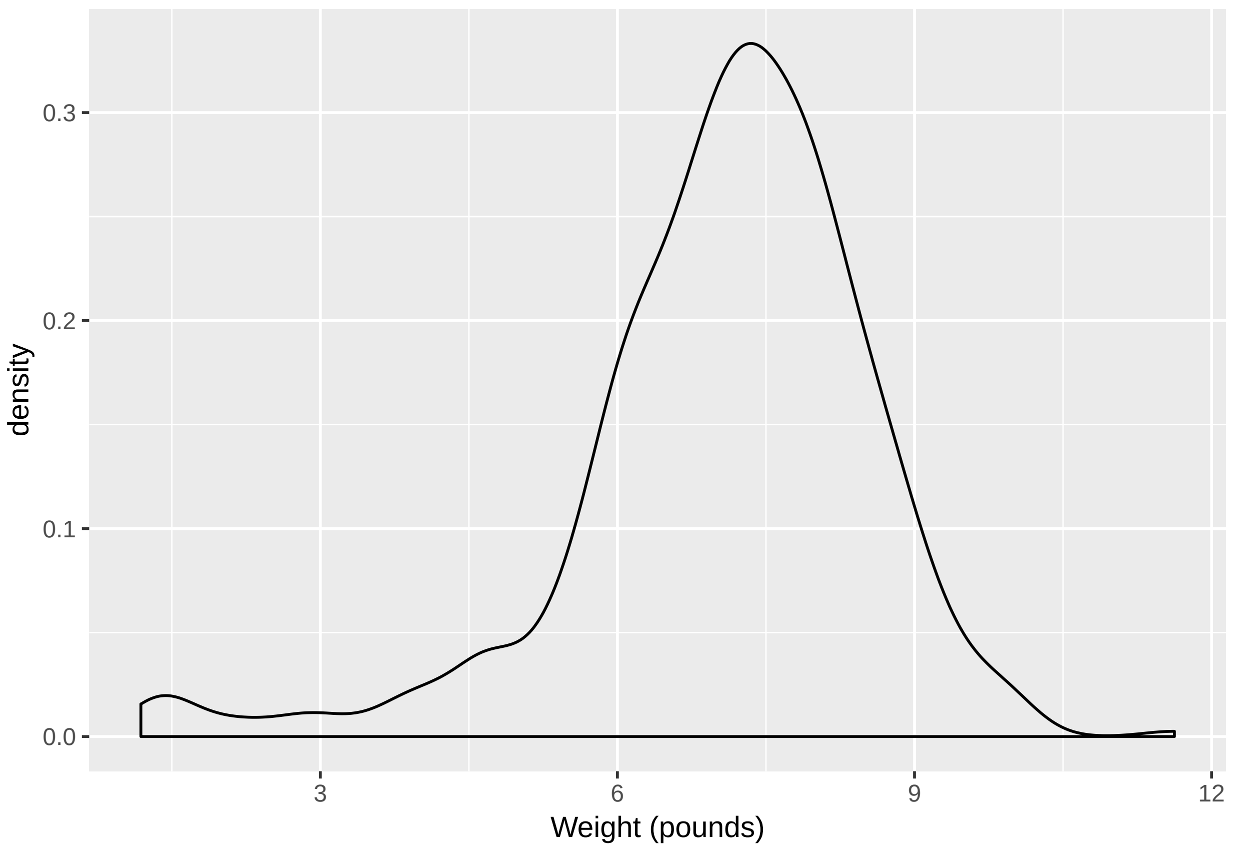
```
## [1] 26
```

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()
```



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

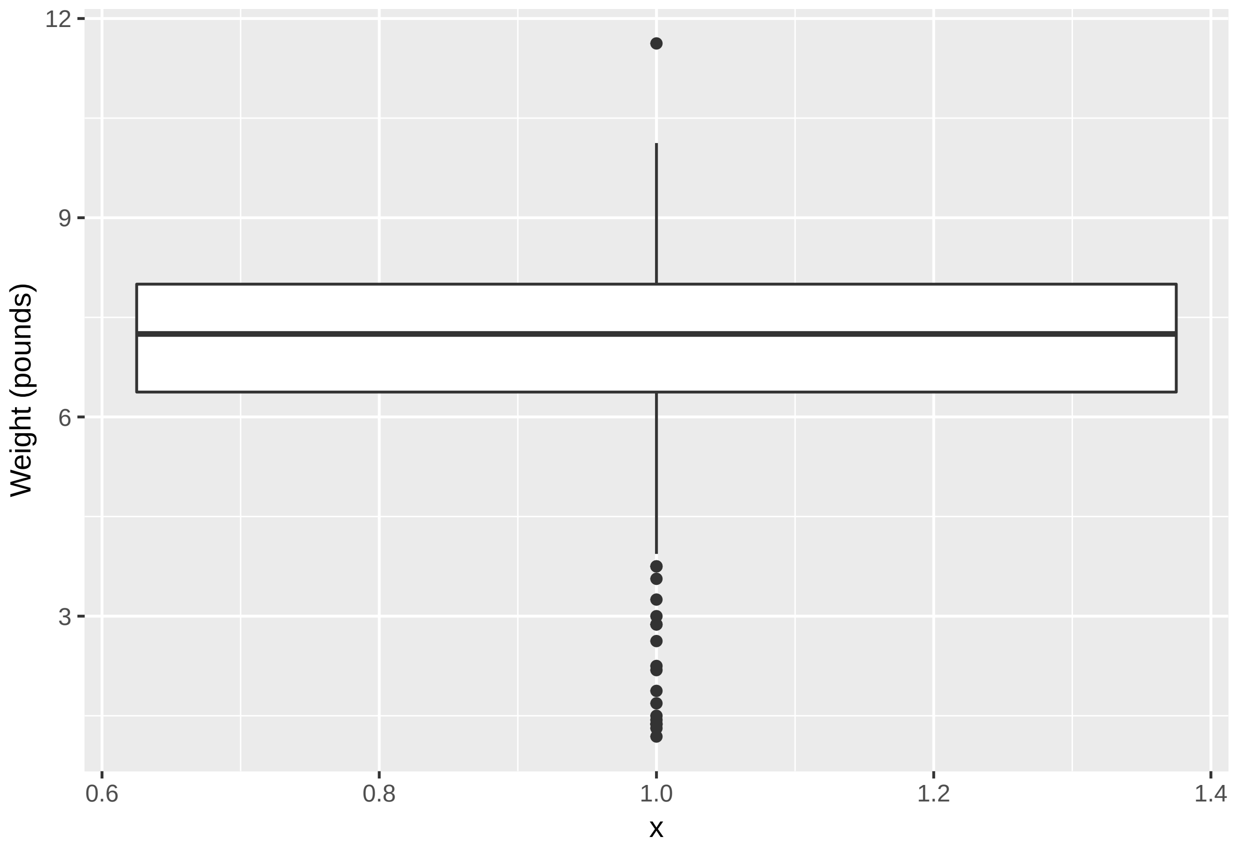
Alright, before we got distracted, we were assessing normality. What about that?

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!endnote{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.}

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.

Question title

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

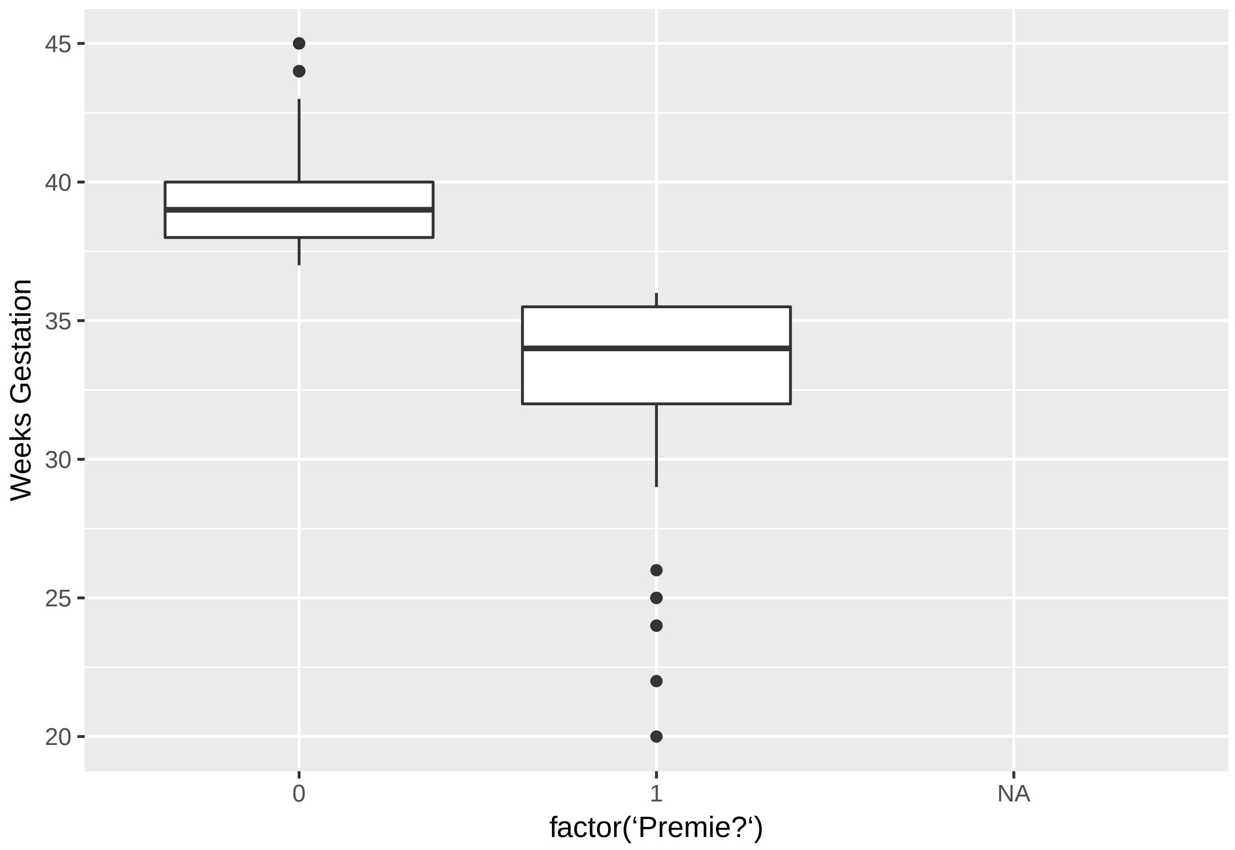
(a) How short does a pregnancy have to be, for the birth to be classified as "premature"? Deduce this from the data, by drawing a suitable graph or otherwise.

Solution

To figure it out from the data, we can see how `Weeks Gestation` depends on verb=`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()
```

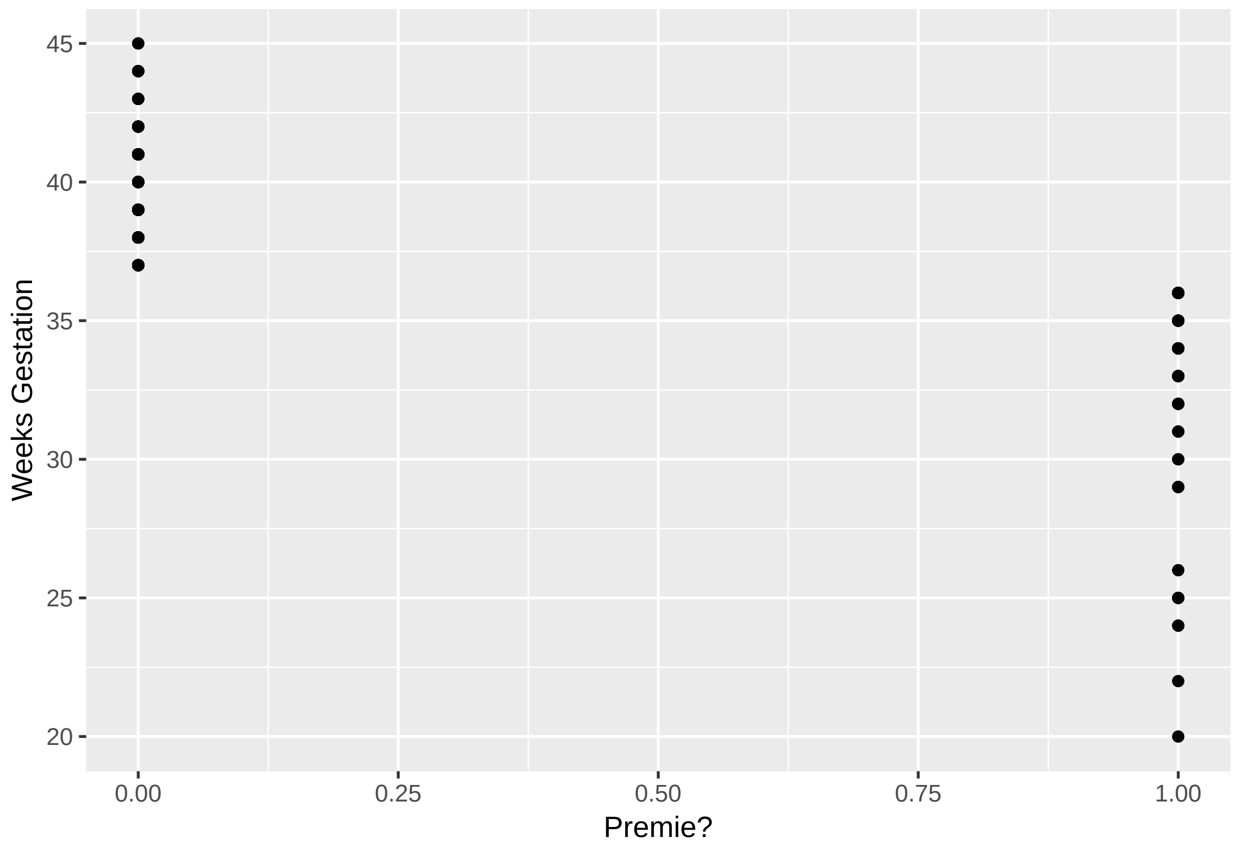
```
## Warning: Removed 1 rows containing non-finite values (stat_boxplot).
```



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 verb=`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 verb+`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.endnote{I explain the missing values below.} 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
##       <int> <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 verb=`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.

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 verb=`Premie?`=: evidently its weeks of gestation was missing as well, since the min and max of that were missing.endnote{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.}

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.

```
bw %>% filter(is.na(`Premie?`)) %>%
gather(name,value,everything())
```

```
## # 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.

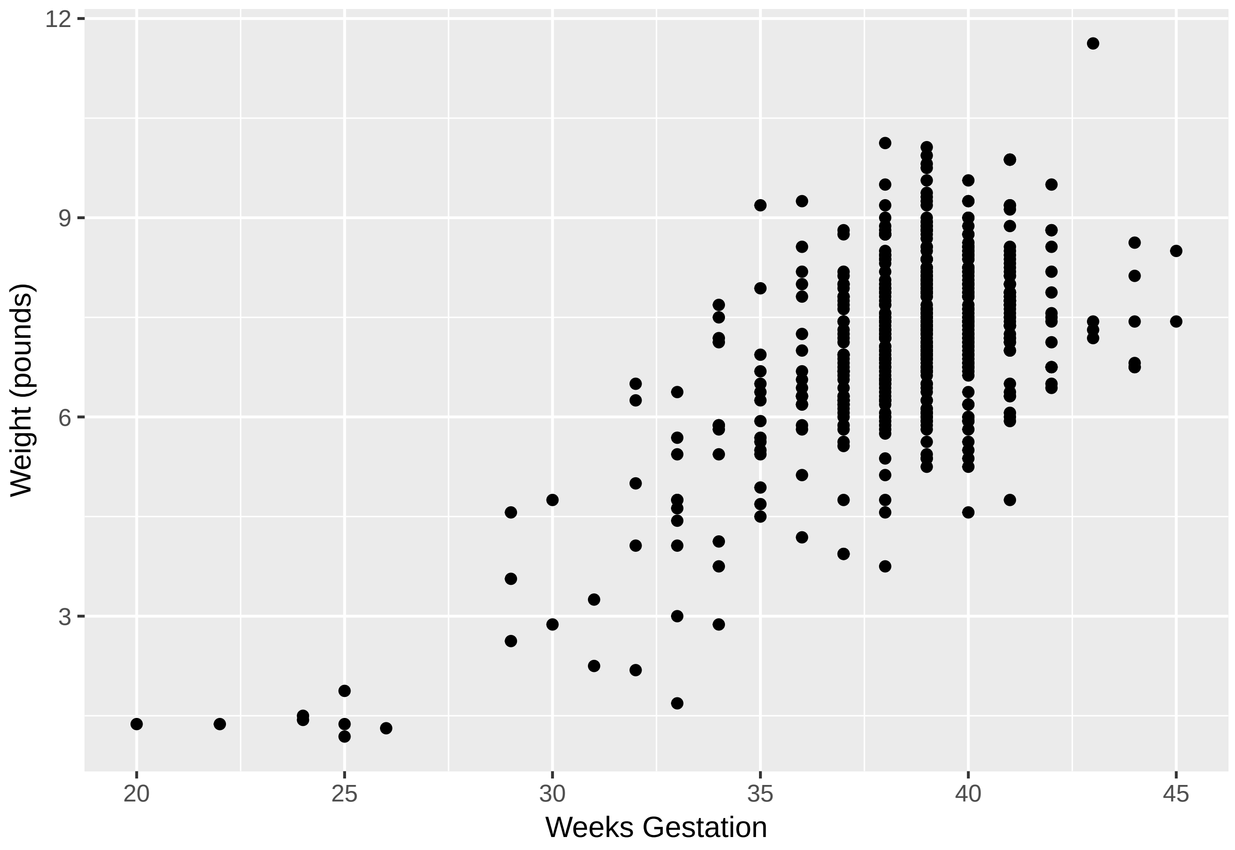
(b) Explore the relationship between birth weight and length of pregancy ("gestation") using a suitable graph. What do you see?

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).
```

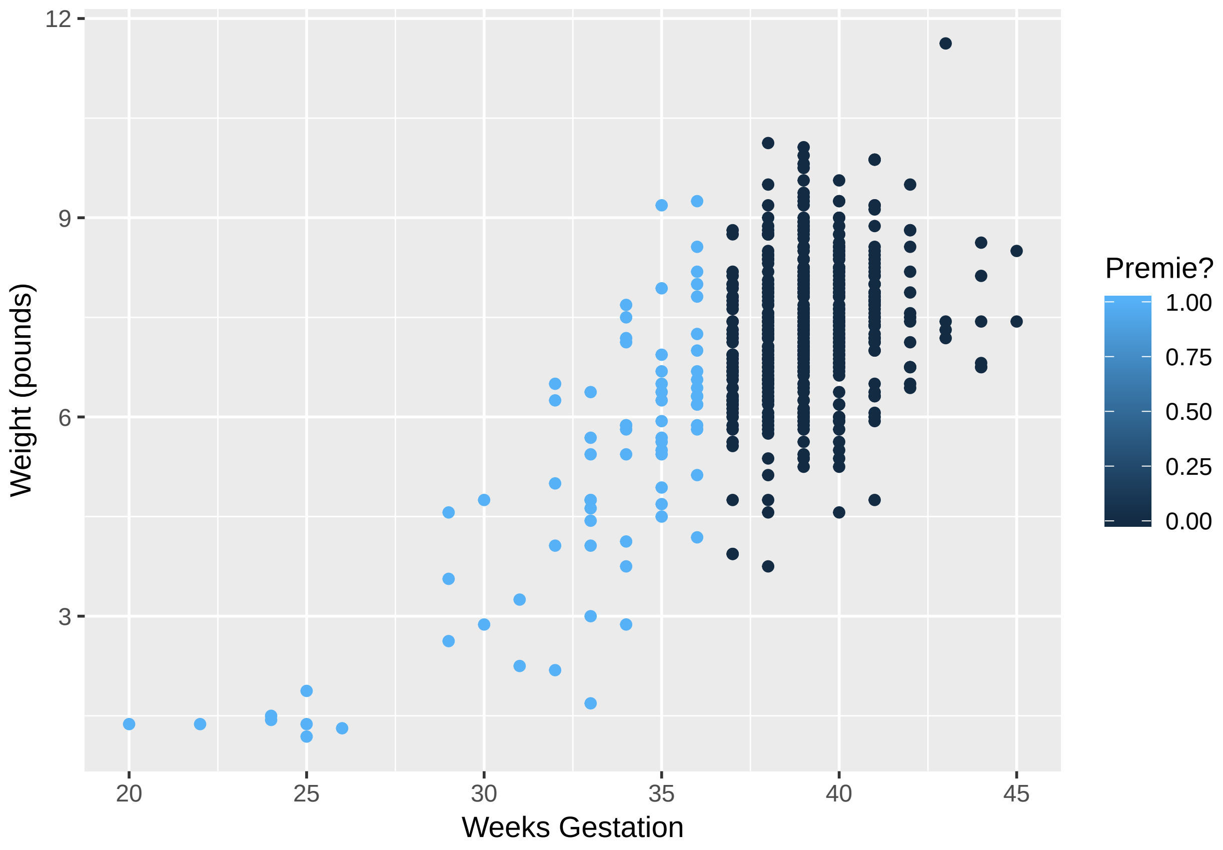


You see a rather clear upward trend. Those very underweight babies came from very short pregnancies, but the vast majority of pregnancies were of more or less normal length (40 weeks is normal) and resulted in babies of more or less normal birth weight.

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()
```

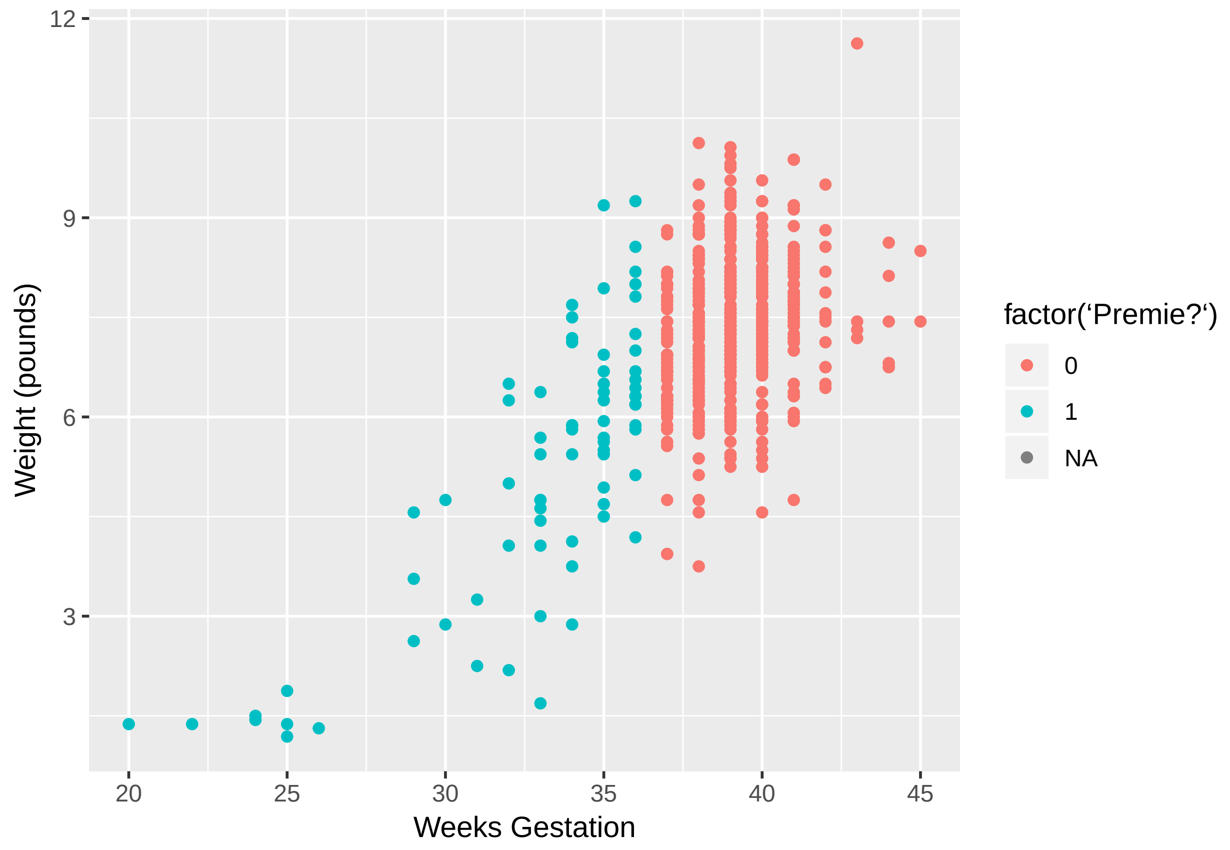
```
## Warning: Removed 1 rows containing missing values (geom_point).
```



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`, ldots”, 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.

Question title

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 http://www.utsc.utoronto.ca/

~butler/c32/nenana.txt.

(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://www.utsc.utoronto.ca/~butler/c32/nenana.txt"
nenana=read_tsv(myurl)
```

```
## Parsed with column specification:
## cols(
##   Year = col_integer(),
##   JulianDate = col_double(),
##   `Date&Time` = col_character()
## )
```

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," ")
```

```
## Parsed with column specification:
## cols(
##   `Year  JulianDate  Date&Time` = col_character()
## )

## Warning in rbind(names(probs), probs_f): number of columns of result is not
## a multiple of vector length (arg 1)

## Warning: 87 parsing failures.
## row # A tibble: 5 x 5 col     row col   expected  actual    file
## ... ................. ... ..........................................................................
## See problems(...) for more details.
```

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"
##  2 "1918\t131.3983\tMay"
##  3 "1919\t123.6066\tMay"
##  4 "1920\t132.4490\tMay"
##  5 "1921\t131.2795\tMay"
##  6 "1922\t132.5559\tMay"
##  7 "1923\t129.0837\tMay"
##  8 "1924\t132.6323\tMay"
##  9 "1925\t127.7726\tMay"
## 10 "1926\t116.6691\tApril"
## # ... with 77 more rows
```

Those verb=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> <chr> <chr>     <chr>    <chr>
##  1     1 <NA>  1 columns 5 colum~ 'http://www.utsc.utoronto.ca/~butler/c3~
##  2     2 <NA>  1 columns 5 colum~ 'http://www.utsc.utoronto.ca/~butler/c3~
##  3     3 <NA>  1 columns 5 colum~ 'http://www.utsc.utoronto.ca/~butler/c3~
##  4     4 <NA>  1 columns 5 colum~ 'http://www.utsc.utoronto.ca/~butler/c3~
##  5     5 <NA>  1 columns 5 colum~ 'http://www.utsc.utoronto.ca/~butler/c3~
##  6     6 <NA>  1 columns 5 colum~ 'http://www.utsc.utoronto.ca/~butler/c3~
##  7     7 <NA>  1 columns 5 colum~ 'http://www.utsc.utoronto.ca/~butler/c3~
##  8     8 <NA>  1 columns 5 colum~ 'http://www.utsc.utoronto.ca/~butler/c3~
##  9     9 <NA>  1 columns 5 colum~ 'http://www.utsc.utoronto.ca/~butler/c3~
## 10    10 <NA>  1 columns 5 colum~ 'http://www.utsc.utoronto.ca/~butler/c3~
## # ... with 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:

```
nenana
```

```
## # A tibble: 87 x 3
##     Year JulianDate `Date&Time`
##    <int>      <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
## # ... with 77 more rows
```

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

```
glimpse(nenana)
```

```
## Observations: 87
## Variables: 3
## $ Year        <int> 1917, 1918, 1919, 1920, 1921, 1922, 1923, 1924, 19...
## $ JulianDate  <dbl> 120.4795, 131.3983, 123.6066, 132.4490, 131.2795, ...
## $ `Date&Time` <chr> "April 30 at 11:30 AM", "May 11 at 9:33 AM", "May ...
```

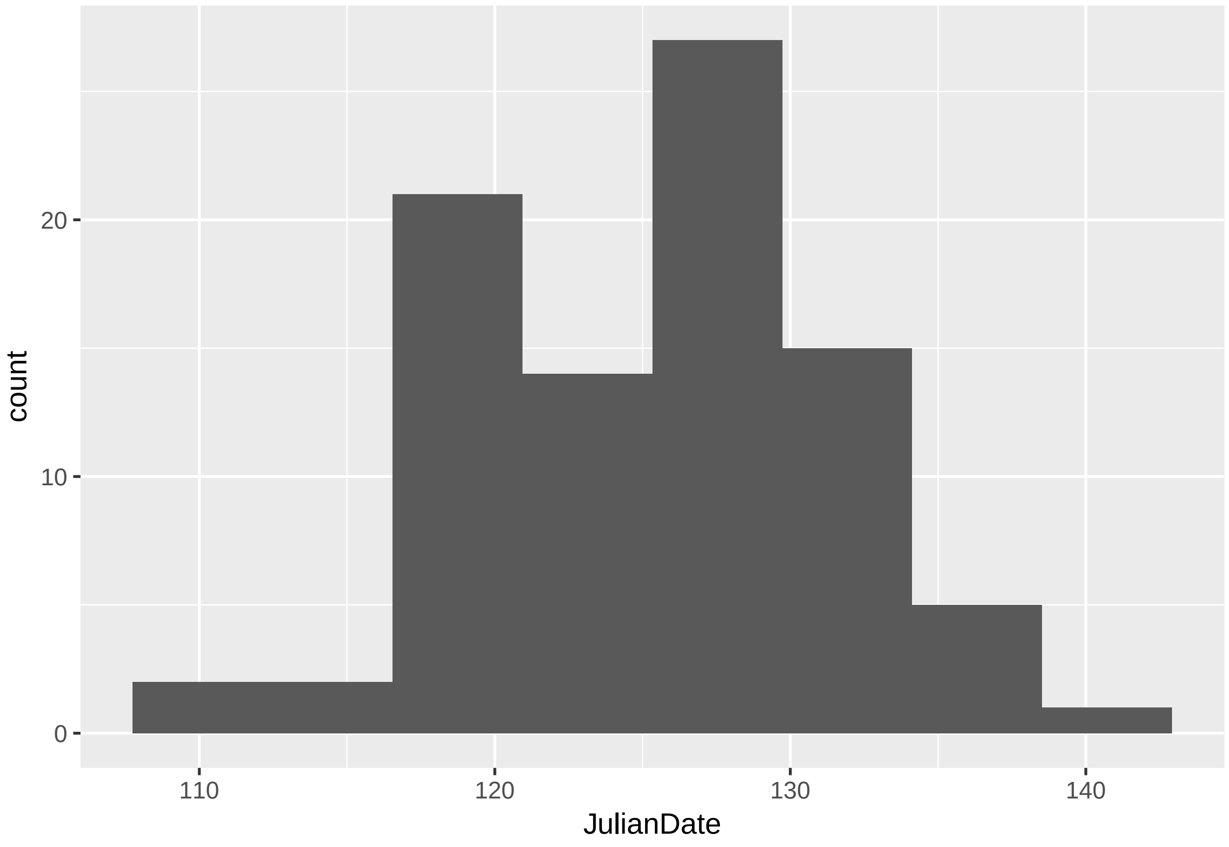
There are 87 years, and 3 columns (variables). The first column is year, and the last column is the date and time that the tripod fell into the river, written as a piece of text. I explain the second column in a moment.

(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:

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

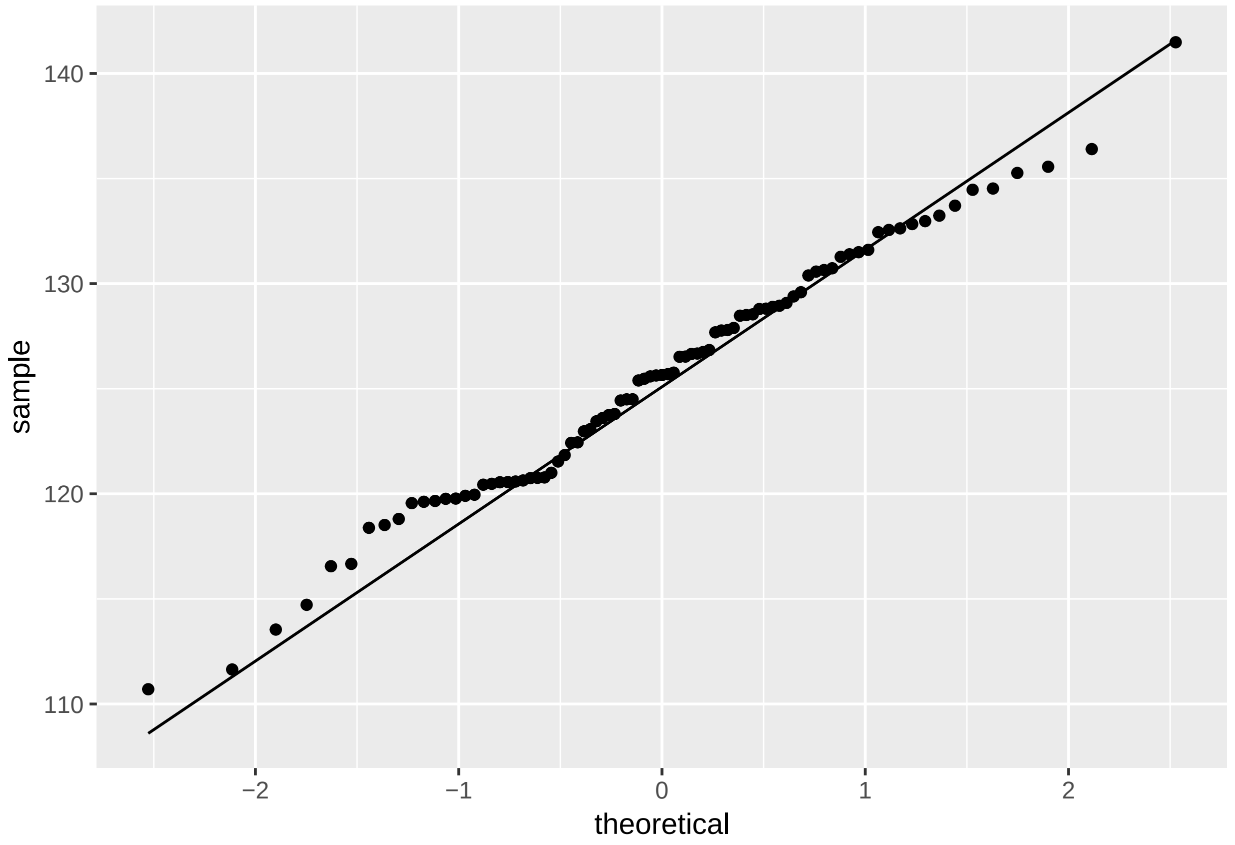


Note that you need to type `JulianDate` exactly as it appears, capital letters and all. R is case-sensitive.

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.endnote{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*.}

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

```
library(lubridate)
```

```
##
## Attaching package: 'lubridate'

## The following object is masked from 'package:base':
##
##     date
```

```
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
##    <int>      <dbl> <chr>             <chr>             <dttm>
##  1  1917       120. April 30 at 11:3~ 1917 April 30 a~ 1917-04-30 11:30:00
##  2  1918       131. May 11 at 9:33 AM 1918 May 11 at ~ 1918-05-11 09:33:00
##  3  1919       124. May 3 at 2:33 PM  1919 May 3 at 2~ 1919-05-03 14:33:00
##  4  1920       132. May 11 at 10:46 ~ 1920 May 11 at ~ 1920-05-11 10:46:00
##  5  1921       131. May 11 at 6:42 AM 1921 May 11 at ~ 1921-05-11 06:42:00
##  6  1922       133. May 12 at 1:20 PM 1922 May 12 at ~ 1922-05-12 13:20:00
##  7  1923       129. May 9 at 2:00 AM  1923 May 9 at 2~ 1923-05-09 02:00:00
##  8  1924       133. May 11 at 3:10 PM 1924 May 11 at ~ 1924-05-11 15:10:00
##  9  1925       128. May 7 at 6:32 PM  1925 May 7 at 6~ 1925-05-07 18:32:00
## 10  1926       117. April 26 at 4:03~ 1926 April 26 a~ 1926-04-26 16:03:00
## # ... with 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. verb-str_c- glues pieces of text together: in this case, the year, a space, and then the rest of the verb-Date&Time-. I stored this in verb-longdt-. The second verb-mutate- is the business end of it: verb-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 verb-lubridate- function is called verb-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> <dttm>
##  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
## # ... with 77 more rows
```

Hmm, some of those are off by one. What do the off-by-one ones have in common? Let's look at more of them. verb-round- rounds off to the nearest integer (since these are actually decimal numbers):

```
nenana2 %>%
filter(round(JulianDate) != round(jd)) %>%
select(JulianDate,jd,datetime)
```

```
## # A tibble: 61 x 3
##    JulianDate    jd datetime
##         <dbl> <dbl> <dttm>
##  1       124.   123 1919-05-03 14:33:00
##  2       133.   132 1922-05-12 13:20:00
##  3       133.   132 1924-05-11 15:10:00
##  4       128.   127 1925-05-07 18:32:00
##  5       117.   116 1926-04-26 16:03:00
##  6       128.   127 1928-05-06 16:25:00
##  7       126.   125 1929-05-05 15:41:00
##  8       129.   128 1930-05-08 19:03:00
##  9       129.   128 1933-05-08 19:30:00
## 10       121.   120 1934-04-30 14:07:00
## # ... with 51 more rows
```

The ones shown here are all *after noon*, and the Julian date in the data file appears as one more than the one calculated by verb-lubridate-. What has actually happened is a quirk of how tibbles are displayed: they show 3 significant digits, *rounded*. The Julian dates given by verb-yday- are the whole-number part, so the ones in the data value are that plus more than 0.5, which will round *up*. The first line of code below displays 6 significant digits rather than only three:

```r
options(pillar.sigfig=6)
nenana2 %>%
filter(round(JulianDate) != round(jd)) %>%
select(JulianDate,jd,datetime)
```

```
## # A tibble: 61 x 3
##    JulianDate    jd datetime
##         <dbl> <dbl> <dttm>
##  1    123.607   123 1919-05-03 14:33:00
##  2    132.556   132 1922-05-12 13:20:00
##  3    132.632   132 1924-05-11 15:10:00
##  4    127.773   127 1925-05-07 18:32:00
##  5    116.669   116 1926-04-26 16:03:00
##  6    127.684   127 1928-05-06 16:25:00
##  7    125.654   125 1929-05-05 15:41:00
##  8    128.794   128 1930-05-08 19:03:00
##  9    128.813   128 1933-05-08 19:30:00
## 10    120.588   120 1934-04-30 14:07:00
## # ... with 51 more rows
```

Displaying more decimals shows that I was right: verb-jd- is (to this accuracy) a whole number, but verb-JulianDate- is a decimal with fractional part greater than 0.50.

Now I have to turn the extra signficant digits off:

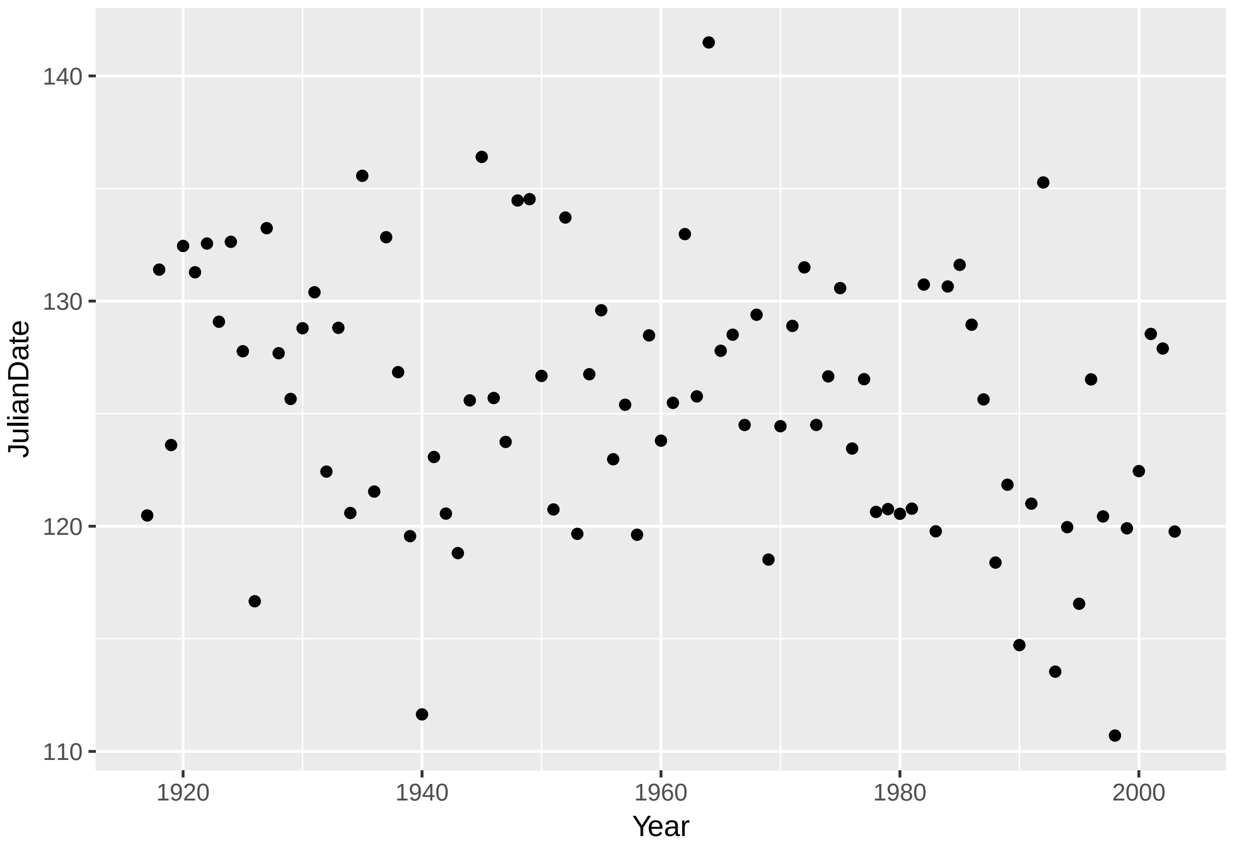
```r
options(pillar.sigfig=3)
```

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

Solution

verb+geom_point+:

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

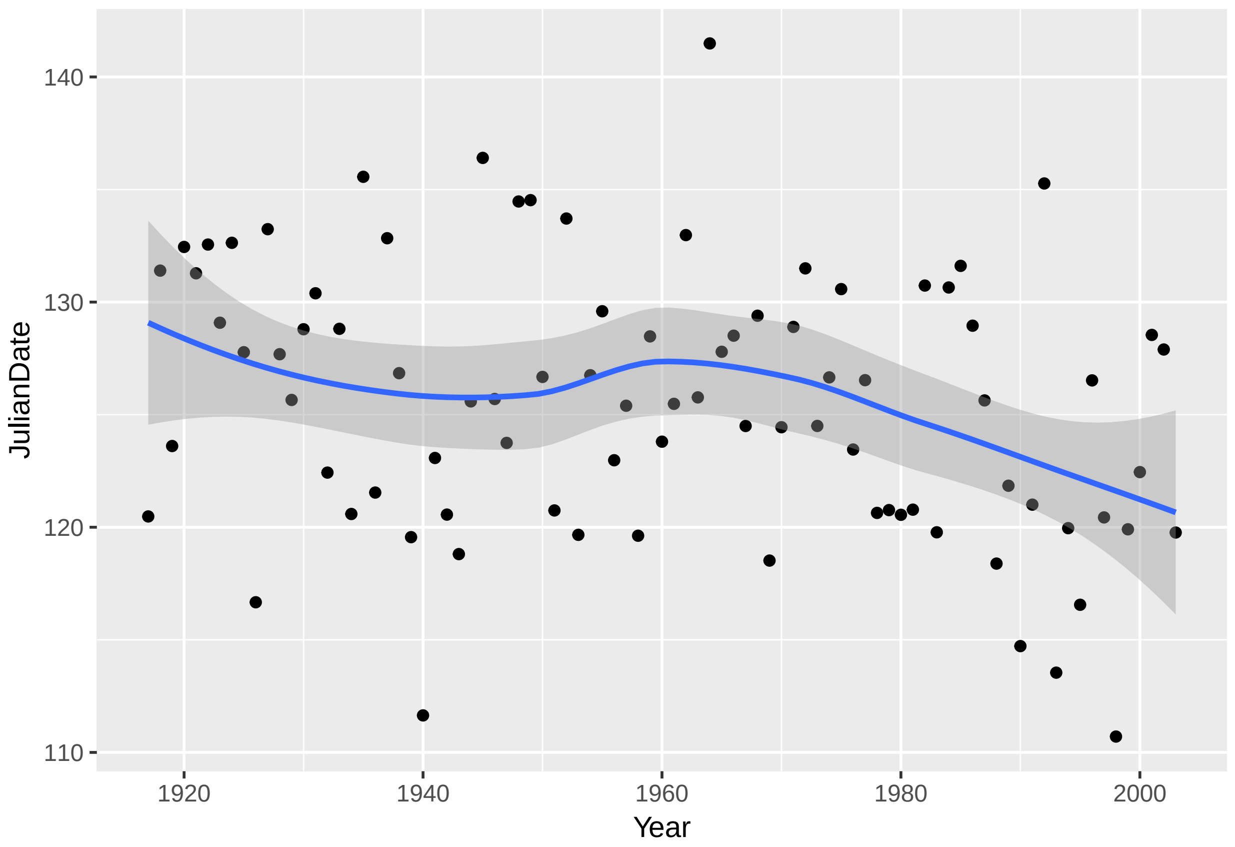


This is actually a small-but-real downward trend, especially since about 1960, but the large amount of variability makes it hard to see, so I'm good with either "no trend" or "weak downward trend" or anything roughly like that. There is definitely not much trend before 1960, but most of the really early 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.

Question title

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://www.utsc.utoronto.ca/~butler/c32/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.

```
url="http://www.utsc.utoronto.ca/~butler/c32/compatt.txt"
anxiety=read_delim(url," ")
```

```
## Parsed with column specification:
```

```
## cols(
##   gender = col_character(),
##   CAS = col_double(),
##   CARS = col_double()
## )
```

```
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
## # ... with 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`:

```
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 list out the whole data set with `print(n=Inf)` 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 (or, later, SAS) to count them for you.

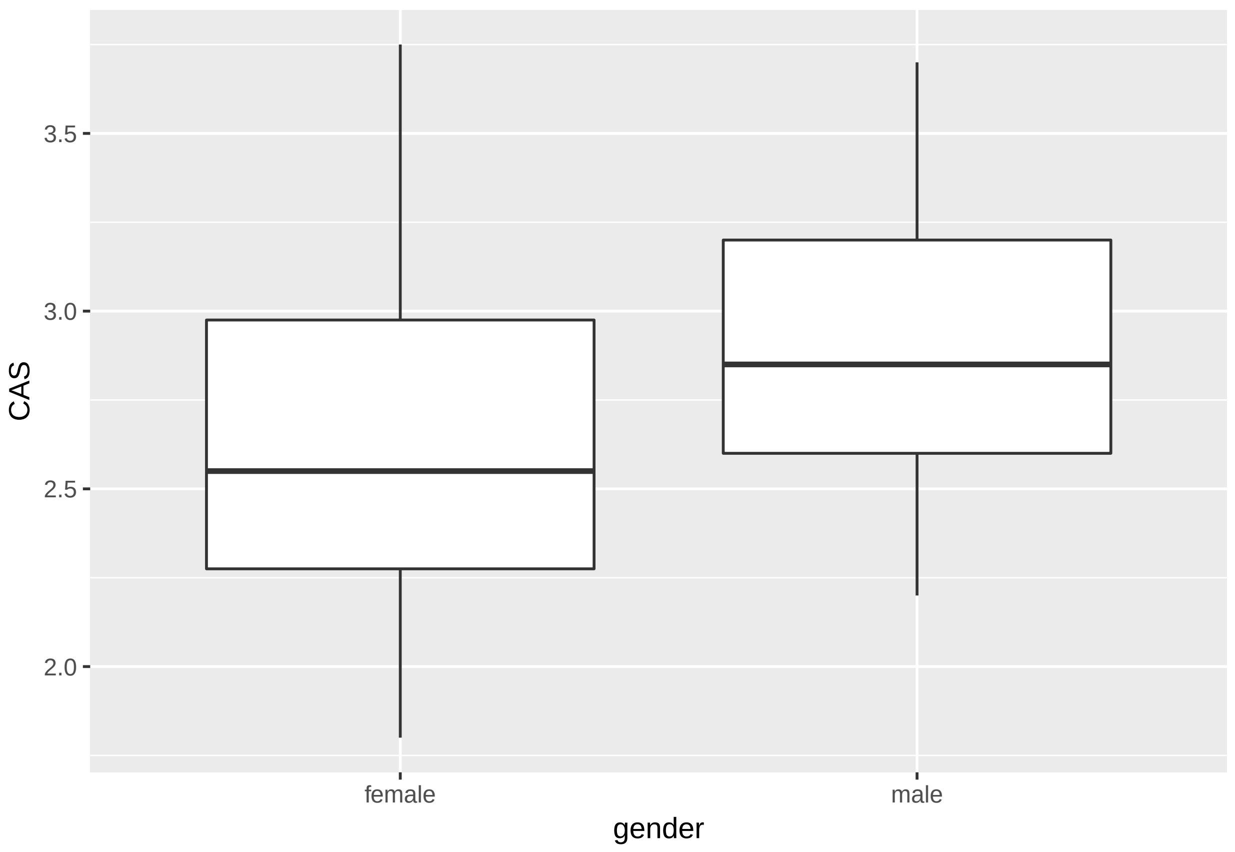
15 females and 20 males, *which you should say.* I made a point of *not* saying that it is enough to get the output with the answers on it, so you need to tell me what the answer is.

(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:

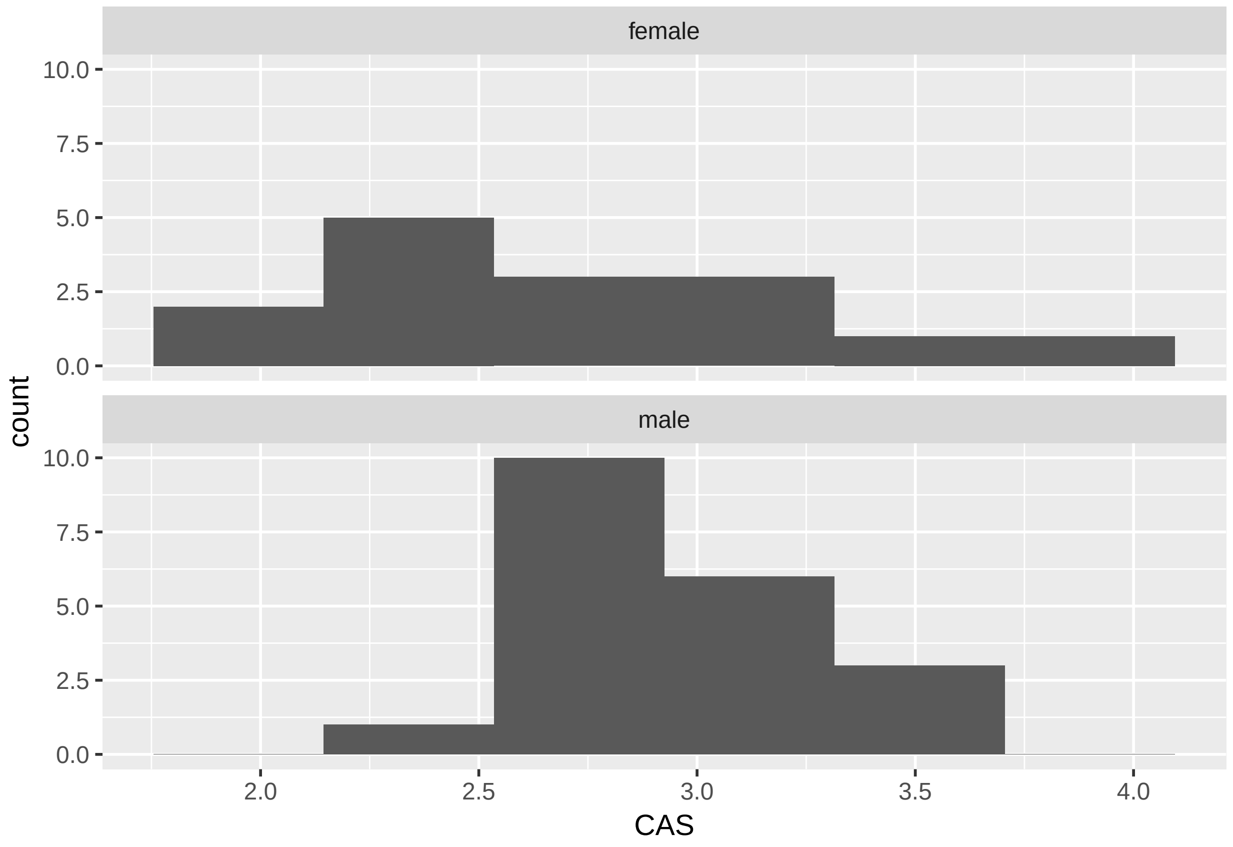
```
ggplot(anxiety,aes(x=gender,y=CAS))+geom_boxplot()
```



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:

```
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, either `summarize_if` or `summarize_at`. To do it the first way, ask what these two columns have in common: they are the only two numeric (quantitative) columns:

```
anxiety %>% summarize_if(is.numeric,funs(mean,sd))
```

```
## # A tibble: 1 x 4
##   CAS_mean CARS_mean CAS_sd CARS_sd
##      <dbl>     <dbl>  <dbl>   <dbl>
## 1     2.82      2.77  0.484   0.671
```

Or the second way, asking yourself what the *names* of those columns have in common: they start with C and the gender column doesn't:

```
anxiety %>% summarize_at(vars(starts_with("C")),funs(mean,sd))
```

```
## # A tibble: 1 x 4
##   CAS_mean CARS_mean CAS_sd CARS_sd
##      <dbl>     <dbl>  <dbl>   <dbl>
## 1     2.82      2.77  0.484   0.671
```

Either of these is good, or anything equivalent (like noting that the two anxiety scales both `ends_with` S):

```
anxiety %>% summarize_at(vars(ends_with("S")),funs(mean,sd))
```

```
## # A tibble: 1 x 4
##   CAS_mean CARS_mean CAS_sd CARS_sd
##      <dbl>     <dbl>  <dbl>   <dbl>
## 1     2.82      2.77  0.484   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. This uses two tools that we haven't seen yet:

```
anxiety %>% select(2:3) %>% summarize_all(funs(mean,sd))
```

```
## # A tibble: 1 x 4
##   CAS_mean CARS_mean CAS_sd CARS_sd
##      <dbl>     <dbl>  <dbl>   <dbl>
## 1     2.82      2.77  0.484   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_if(is.numeric,funs(mean,sd))
```

```
## # A tibble: 2 x 5
##   gender CAS_mean CARS_mean CAS_sd CARS_sd
##   <chr>      <dbl>     <dbl>  <dbl>   <dbl>
## 1 female      2.64      2.51  0.554   0.773
## 2 male        2.94      2.96  0.390   0.525
```

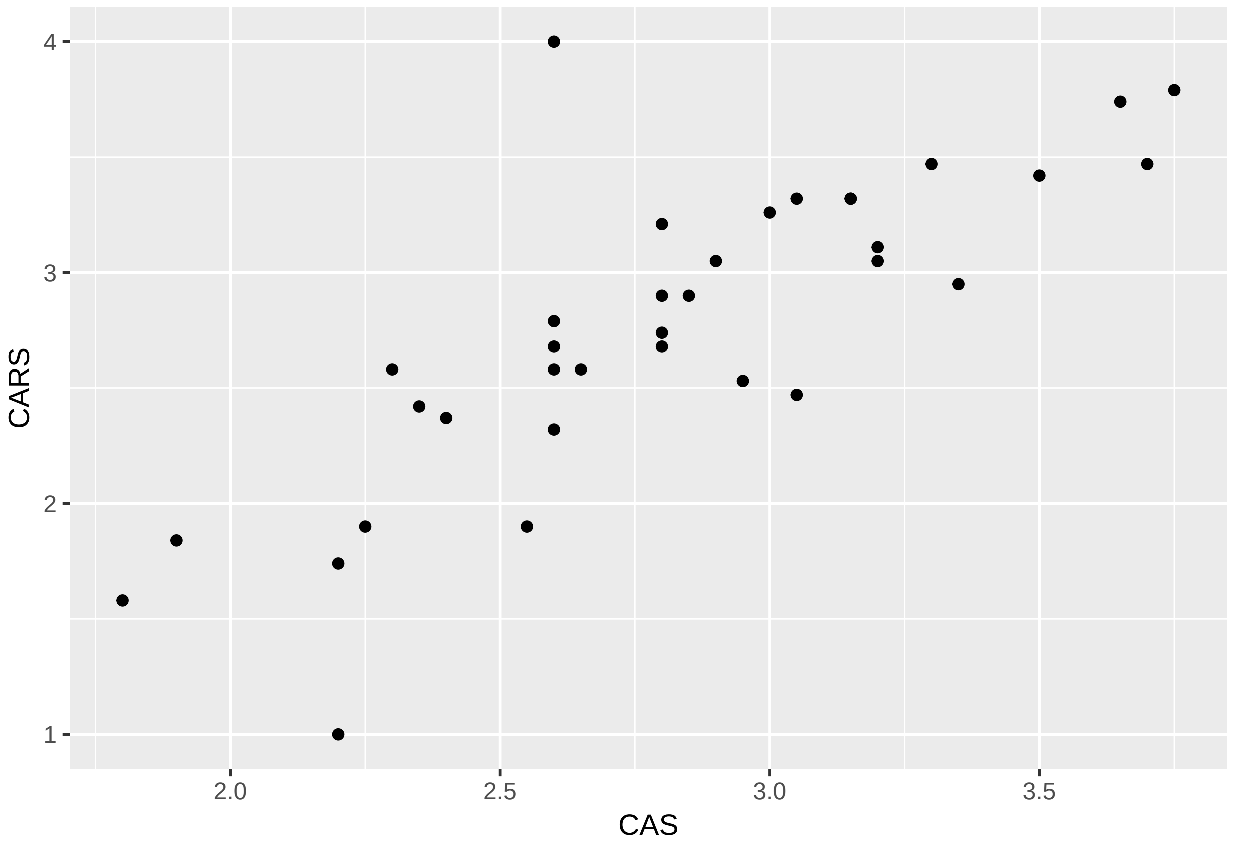
or

```
anxiety %>% group_by(gender) %>%
summarize_at(vars(starts_with("C")),funs(mean,sd))
```

```
## # A tibble: 2 x 5
##   gender CAS_mean CARS_mean CAS_sd CARS_sd
##   <chr>      <dbl>     <dbl>  <dbl>   <dbl>
## 1 female      2.64      2.51  0.554   0.773
## 2 male        2.94      2.96  0.390   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:

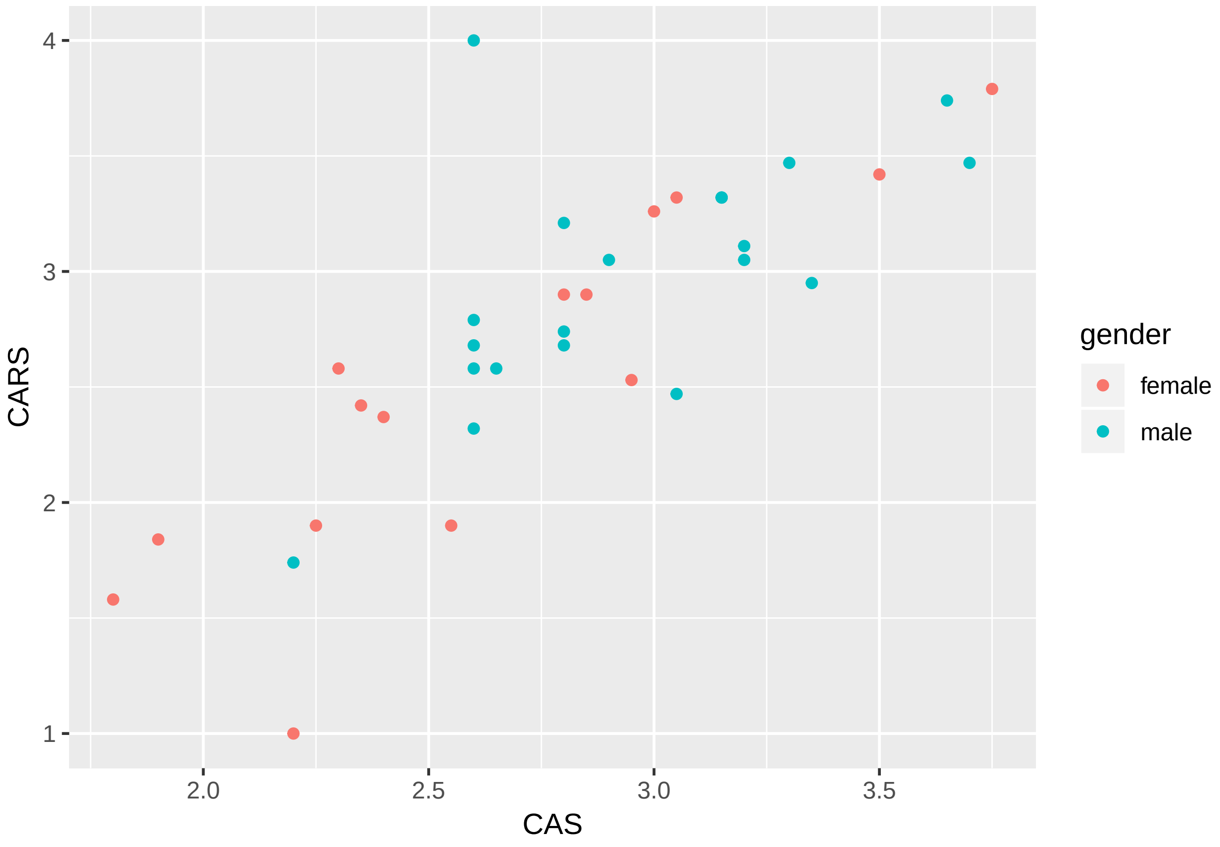
```
ggplot(anxiety,aes(x=CAS,y=CARS))+geom_point()
```



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.

Question title

A hunter-gatherer society is one where people get their food by hunting, fishing or foraging rather than by agriculture or by raising animals. Such societies tend to move from place to place. Anthropologists have studied hunter-gatherer societies in forest ecosystems across the world. The average population density of these societies is 7.38 people per 100 km^2. Hunter-gatherer societies on different continents might have different population densities, possibly because of large-scale ecological constraints (such as resource availability), or because of other factors, possibly social and/or historic, determining population density.

Some hunter-gatherer societies in Australia were studied, and the population density per 100 km^2 recorded for each. The data are in http://www.utsc.utoronto.ca/~butler/c32/hg.txt.

(a) Read the data into R. Do you have the correct variables? How many hunter-gatherer societies in Australia were studied? Explain briefly.

Solution

The data values are separated by (single) spaces, so `read_delim` is the thing:

```
url="http://www.utsc.utoronto.ca/~butler/c32/hg.txt"
societies=read_delim(url," ")
```

```
## Parsed with column specification:
## cols(
##   name = col_character(),
##   density = col_double()
## )
```

I like to put the URL in a variable first, because if I don't, the `read_delim` line can be rather long. But if you want to do it in one step, that's fine, as long as it's clear that you are doing the right thing.

Let's look at the data frame:

```
societies
```

```
## # A tibble: 13 x 2
##    name       density
##    <chr>        <dbl>
##  1 jeidji       17
##  2 kuku         50
##  3 mamu         45
##  4 ngatjan      59.8
##  5 undanbi      21.7
##  6 jinibarra    16
##  7 ualaria       9
##  8 barkindji    15.4
##  9 wongaibon     5.12
## 10 jaralde      40
## 11 tjapwurong   35
## 12 tasmanians   13.4
## 13 badjalang    13.4
```

I have the name of each society and its population density, as promised (so that is correct). There were 13 societies that were studied. For me, they were all displayed. For you, you'll probably see only the first ten, and you'll have to click Next to see the last three.

(b) The question of interest is whether these Australian hunter-gatherer societies are like the rest of the world in terms of mean population density. State suitable null and alternative hypotheses. *Define any symbols you use*: that is, if you use a symbol, you also have to say what it means.

Solution

The mean for the world as a whole ("average", as stated earlier) is 7.38. Let *mu* denote the population mean for Australia (of which these societies are a sample). Then our hypotheses are:

$$H_0 : \mu = 7.38$$

and

$$H_a : \mu \neq 7.38.$$

There is no reason for a one-sided alternative here, since all we are interested in is whether Australia is different from the rest of the world. *Expect to lose a point* if you use the symbol μ without saying what it means.

(c) Test your hypotheses using a suitable test. What do you conclude, in the context of the data?

Solution

A t-test, since we are testing a mean:

```
t.test(societies$density,mu=7.38)
```

```
##
##  One Sample t-test
##
```

```
## data:  societies$density
## t = 3.8627, df = 12, p-value = 0.002257
## alternative hypothesis: true mean is not equal to 7.38
## 95 percent confidence interval:
##  15.59244 36.84449
## sample estimates:
## mean of x
##  26.21846
```

The P-value is 0.0023, less than the usual *alpha* of 0.05, so we *reject* the null hypothesis and conclude that the mean population density is not equal to 7.38. That is to say, Australia is different from the rest of the world in this sense.

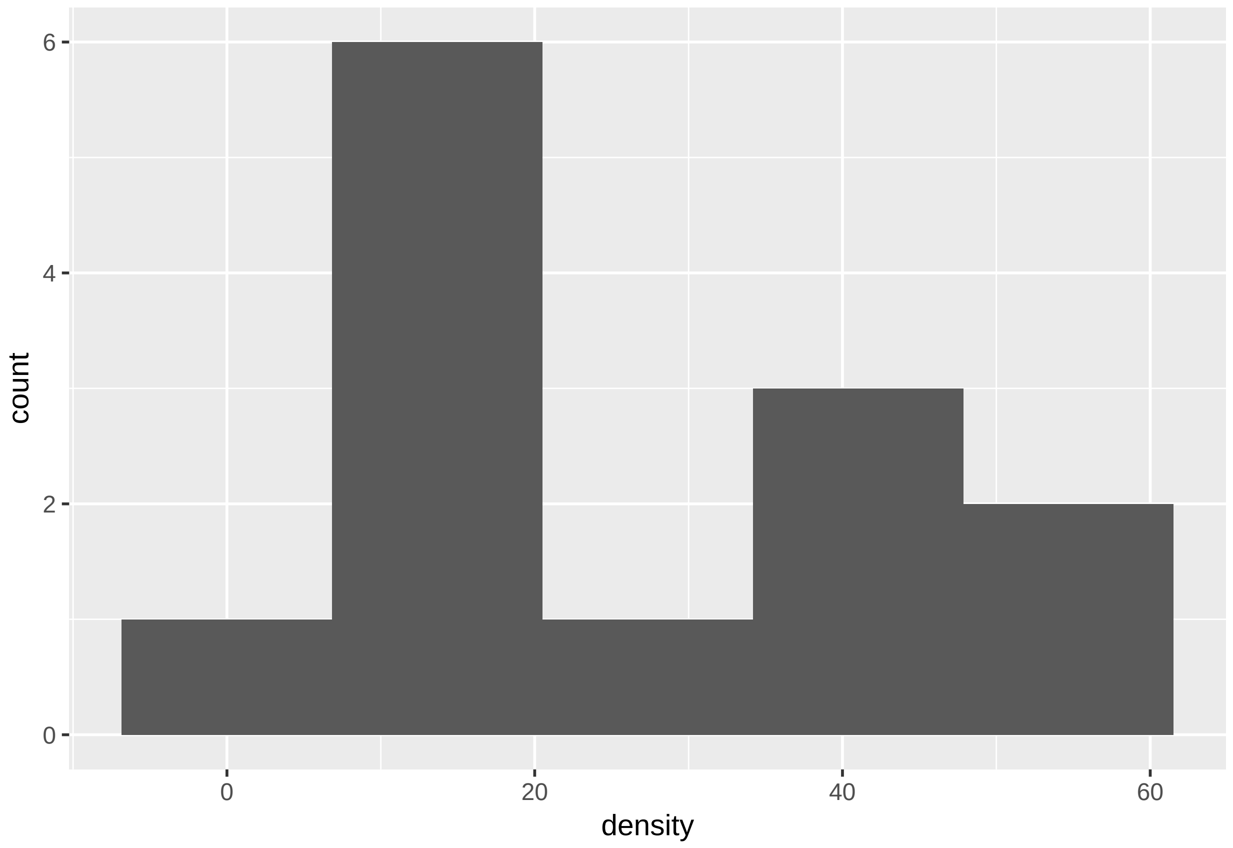
As you know, "reject the null hypothesis" is only part of the answer, so gets only part of the marks.

(d) Do you have any doubts about the validity of your test? Explain briefly, using a suitable graph to support your explanation.

Solution

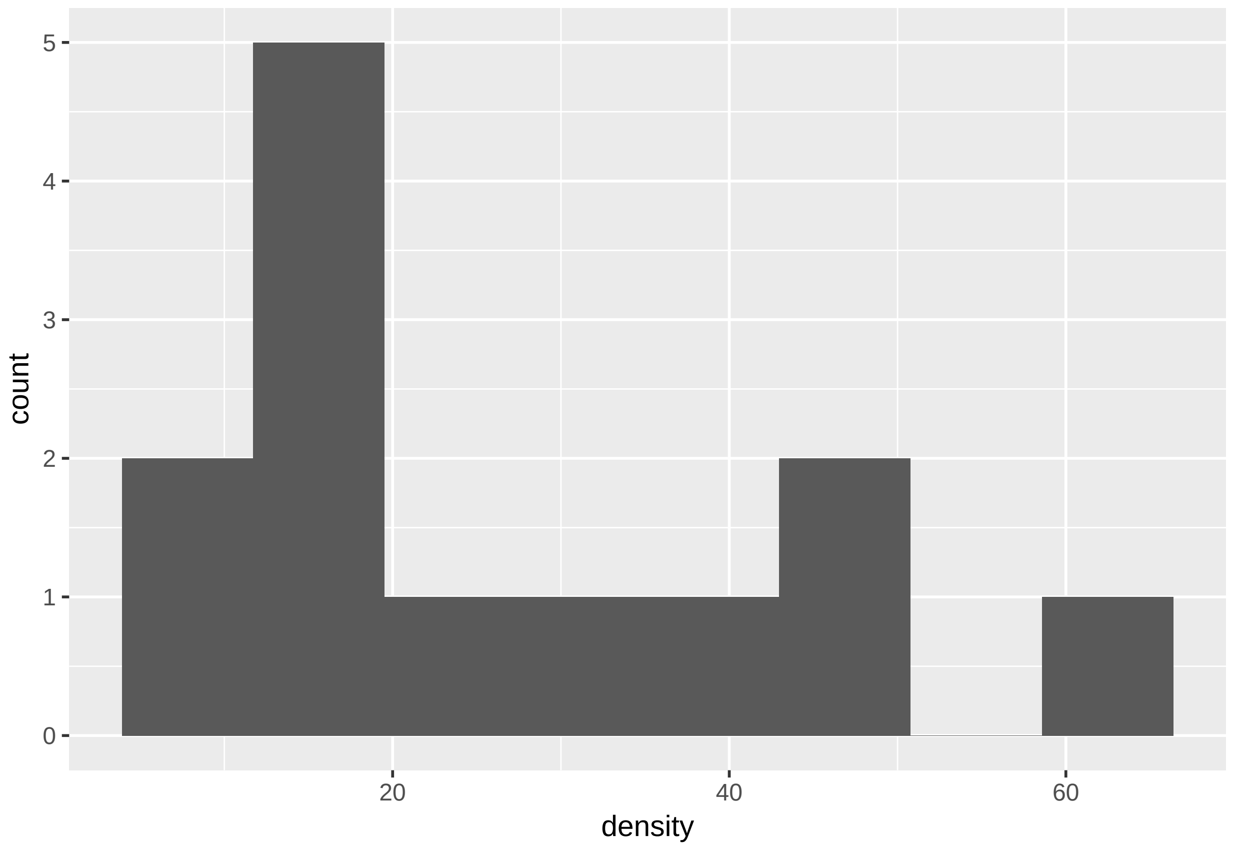
The assumption behind the *t*-test is that the data are approximately normal. We can assess that in several ways, but the simplest (which is perfectly acceptable at this point) is a histogram. You'll need to pick a suitable number of bins. This one comes from Sturges' rule:

```
ggplot(societies,aes(x=density))+geom_histogram(bins=5)
```



Your conclusion might depend on how many bins you chose for your histogram. Here's 8 bins (which is really too many with only 13 observations, but it actually shows the shape well):

```
ggplot(societies,aes(x=density))+geom_histogram(bins=8)
```



or you can get a number of bins from one of the built-in functions, such as:

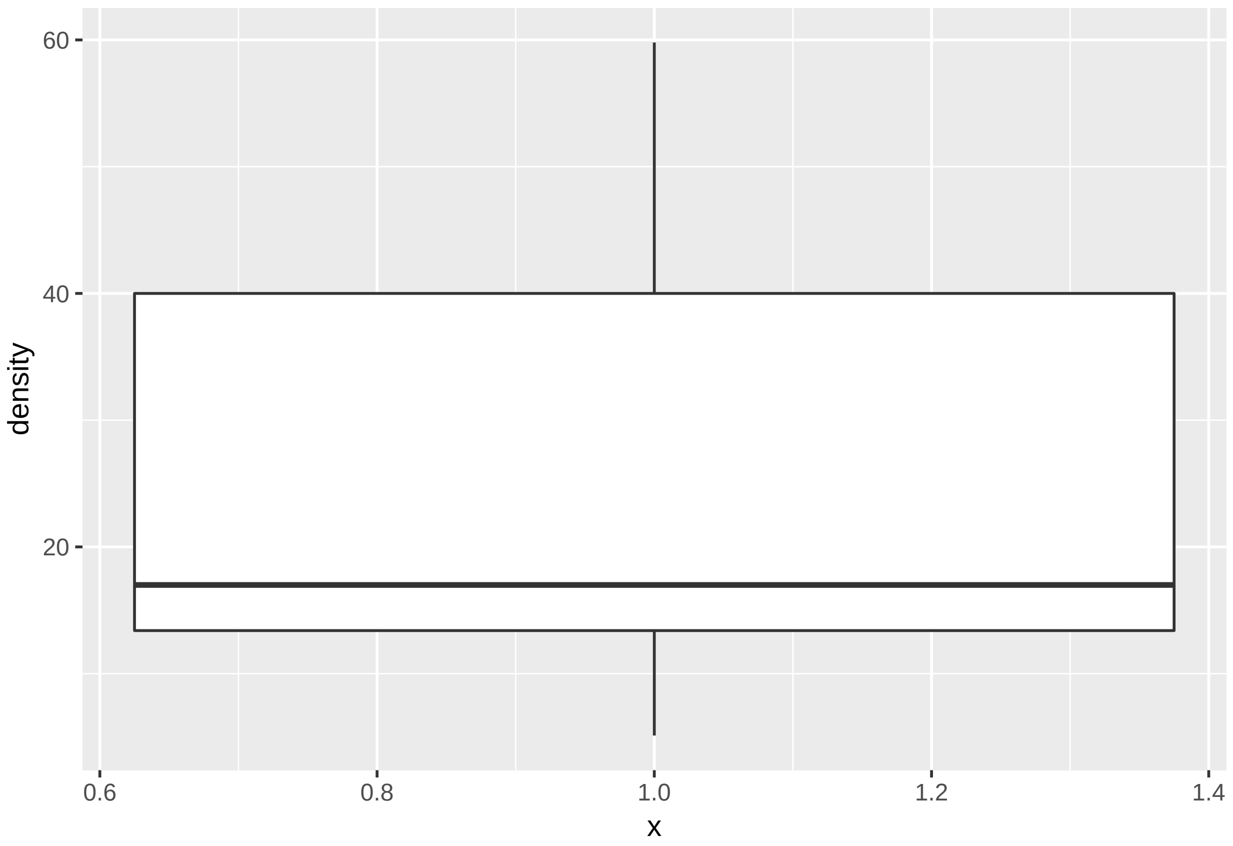
```
mybins=nclass.FD(societies$density)
mybins
```

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

This one is small. The interquartile range is large and n is small, so the binwidth will be large and therefore the number of bins will be small.

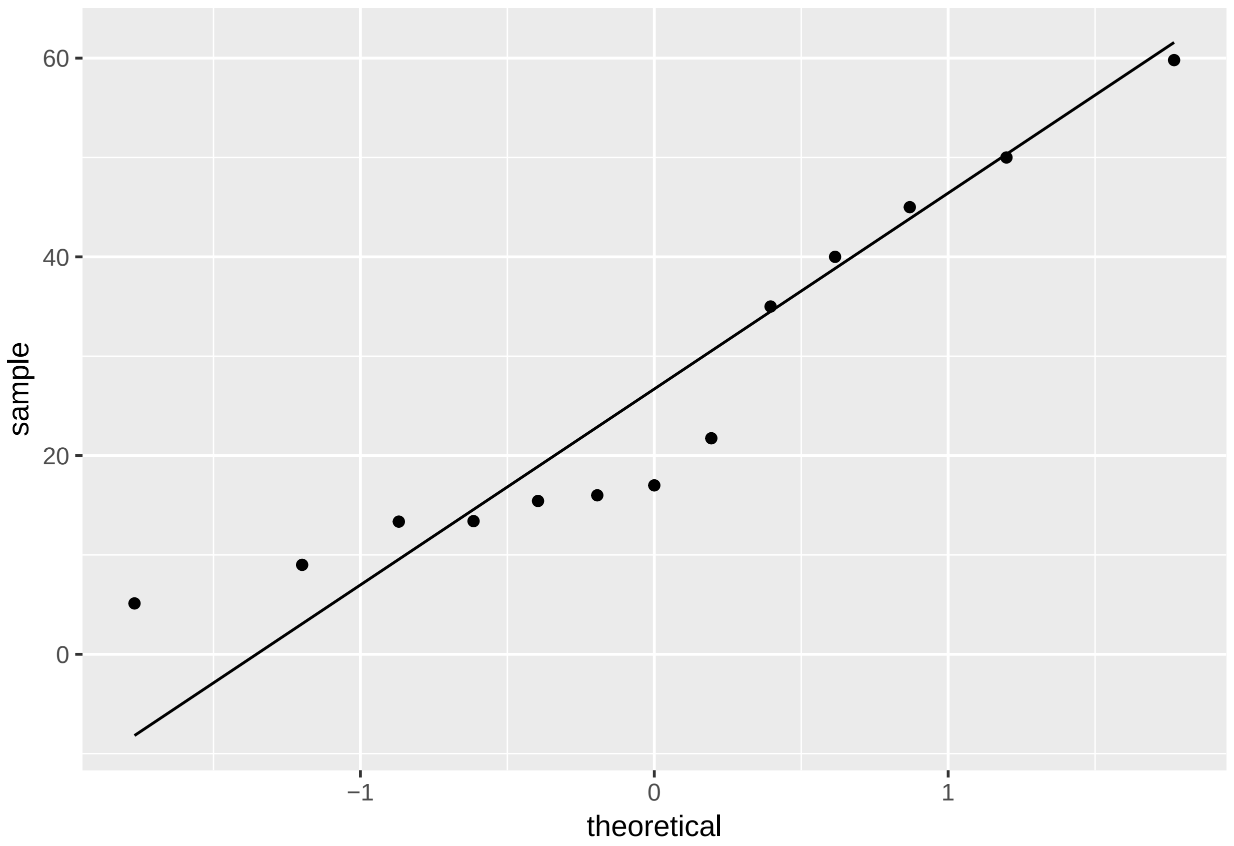
Other choices: a one-group boxplot:

```
ggplot(societies,aes(x=1,y=density))+geom_boxplot()
```



This isn't the best for assessing normality as such, but it will tell you about lack of symmetry and outliers, which are the most important threats to the *t*-test, so it's fine here. Or, a normal quantile plot:

```
ggplot(societies,aes(sample=density))+
stat_qq()+stat_qq_line()
```



This is actually the best way to assess normality, but I'm not expecting you to use this plot here, because we may not have gotten to it in class yet. (If you have read ahead and successfully use the plot, it's fine.)

After you have drawn your chosen plot (you need *one* plot), you need to say something about normality and thus whether you have any doubts about the validity of your t-test. This will depend on the graph you drew: if you think your graph is symmetric and outlier-free, you should have no doubts about your t-test; if you think it has something wrong with it, you should say what it is and express your doubts. My guess is that you will think this distribution is skewed to the right. Most of my plots are saying that.endnote{The normal quantile plot is rather interesting: it says that the uppermost values are approximately normal, but the *smallest* eight or so values are too bunched up to be normal. That is, normality fails not because of the long tail on the right, but the bunching on the left. Still right-skewed, though.}

On the website where I got these data, they were using the data as an example for another test, precisely *because* they thought the distribution was right-skewed. Later on, we'll learn about the sign test for the median, which I think is actually a better test here.

Chapter 5

One-sample inference

```
library(tidyverse)
```

```
## -- Attaching packages ---------------------------------------------------------------------------------
```

```
## v ggplot2 3.0.0     v purrr   0.2.5
## v tibble  1.4.2     v dplyr   0.7.6
## v tidyr   0.8.1     v stringr 1.3.1
## v readr   1.1.1     v forcats 0.3.0
```

```
## -- Conflicts ------------------------------------------------------------------------------------------
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

Question title

The data in file http://www.utsc.utoronto.ca/~butler/c32/ncbirths.csv 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. We have seen these data before.

(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 (rather than downloading the file, unless you are working offline):

```
myurl="http://www.utsc.utoronto.ca/~butler/c32/ncbirths.csv"
bw=read_csv(myurl)
```

```
## Parsed with column specification:
## cols(
##   `Father Age` = col_integer(),
##   `Mother Age` = col_integer(),
##   `Weeks Gestation` = col_integer(),
##   `Pre-natal Visits` = col_integer(),
##   `Marital Status` = col_integer(),
##   `Mother Weight Gained` = col_integer(),
##   `Low Birthweight?` = col_integer(),
##   `Weight (pounds)` = col_double(),
##   `Premie?` = col_integer(),
##   `Few Visits?` = col_integer()
## )
```

(b) Find a 95% confidence interval for the mean birth weight of all babies born in North Carolina (of which these babies are a sample). At the end, you should state what the confidence interval is. Giving some output is necessary, but *not* enough by itself.

If your variable name has a space or other special character (like a question mark) in it, remember that you have to surround its name with backticks, as discussed the first time we looked at these data.

Solution

This:

```
t.test(bw$`Weight (pounds)`)
```

```
##
##  One Sample t-test
##
## data:  bw$`Weight (pounds)`
## t = 104.94, df = 499, p-value < 2.2e-16
## alternative hypothesis: true mean is not equal to 0
## 95 percent confidence interval:
##  6.936407 7.201093
## sample estimates:
## mean of x
##   7.06875
```

or (the same, but remember to match your brackets):

```
with(bw,t.test(`Weight (pounds)`))
```

```
##
##  One Sample t-test
##
## data:  Weight (pounds)
## t = 104.94, df = 499, p-value < 2.2e-16
## alternative hypothesis: true mean is not equal to 0
## 95 percent confidence interval:
##  6.936407 7.201093
## sample estimates:
## mean of x
##   7.06875
```

The confidence interval goes from 6.94 to 7.20 pounds.

There is an annoyance about `t.test`. Sometimes you can use `data=` with it, and sometimes not. When we do a two-sample *t*-test later, there is a "model formula" with a squiggle in it, and there we can use `data=`, but here not, so you have to use the dollar sign or the `with` to say which data frame to get things from. The distinction seems to be that emph{if you are using a model formula}, you can use `data=`, and if not, not.

This is one of those things that is a consequence of R's history. The original `t.test` was without the model formula and thus without the `data=`, but the model formula got "retro-fitted" to it later. Since the model formula comes from things like regression, where `data=` is legit, that had to be retro-fitted as well. Or, at least, that's my understanding.

(c) Birth weights of babies born in the United States have a mean of 7.3 pounds. Is there any evidence that babies born in North Carolina are less heavy on average? State appropriate hypotheses, do your test, obtain a P-value and state your conclusion, in terms of the original data.

Solution

Null is that the population mean (the mean weight of all babies born in North Carolina) is $H_0 : mu = 7.3$ pounds, and the alternative is that the mean is less: $H_a : mu < 7.3$ pounds. This is a one-sided alternative, which we need to feed into `t.test`:

```
t.test(bw$`Weight (pounds)`,mu=7.3,alternative="less")
```

```
##
##  One Sample t-test
##
## data:  bw$`Weight (pounds)`
## t = -3.4331, df = 499, p-value = 0.0003232
## alternative hypothesis: true mean is less than 7.3
## 95 percent confidence interval:
##      -Inf 7.179752
## sample estimates:
## mean of x
##   7.06875
```

$ %$

Or with `with`. If you see what I mean.

The P-value is 0.0003, which is *less* than any *alpha* we might have chosen: we *reject* the null hypothesis in favour of the alternative, and thus we conclude that the mean birth weight of babies in North Carolina is indeed less than 7.3 pounds.

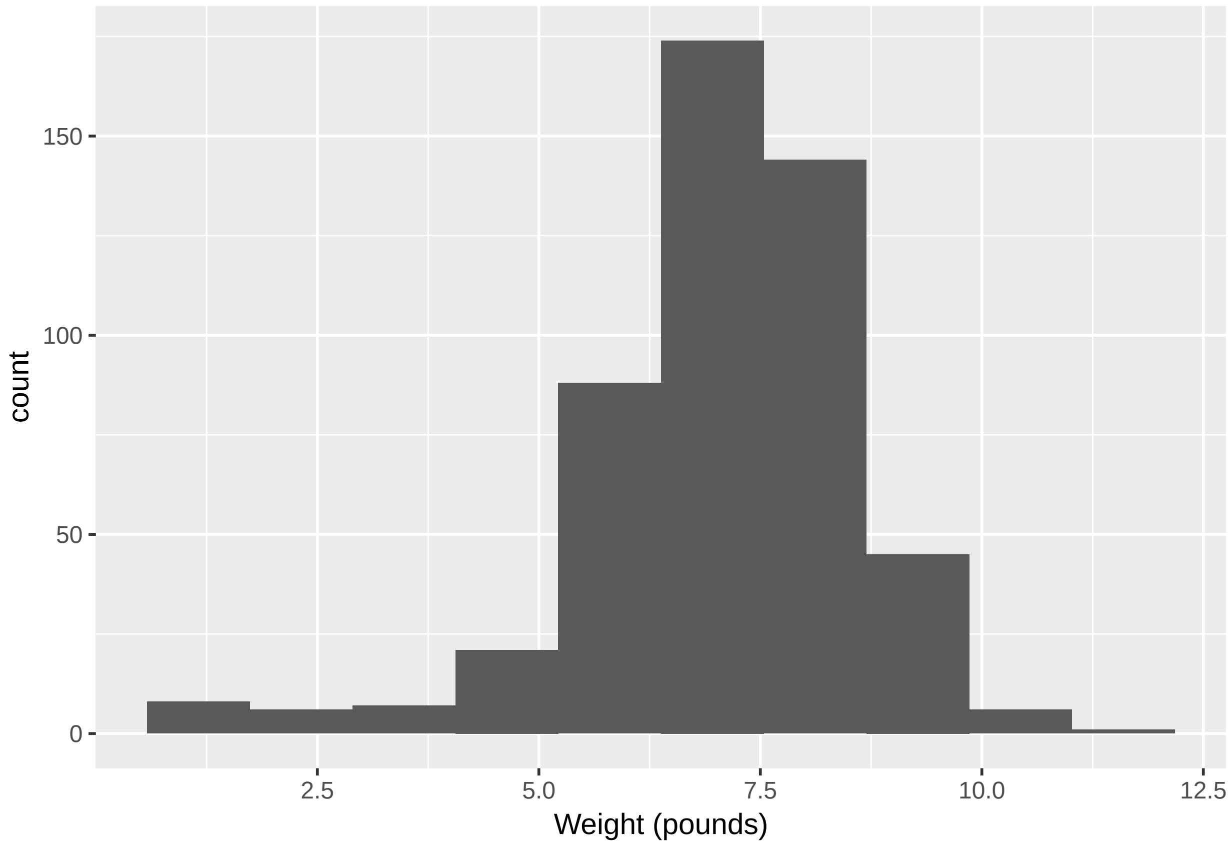
"Reject the null hypothesis" is *not* a complete answer. You need to say something about what rejecting the null hypothesis means *in this case*: that is, you must make a statement about birth weights of babies.

(d) The theory behind the t-test 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.)

Solution

We did this before (and discussed the number of bins before), so I'll just reproduce my 10-bin histogram (which is what I preferred, but this is a matter of taste):

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



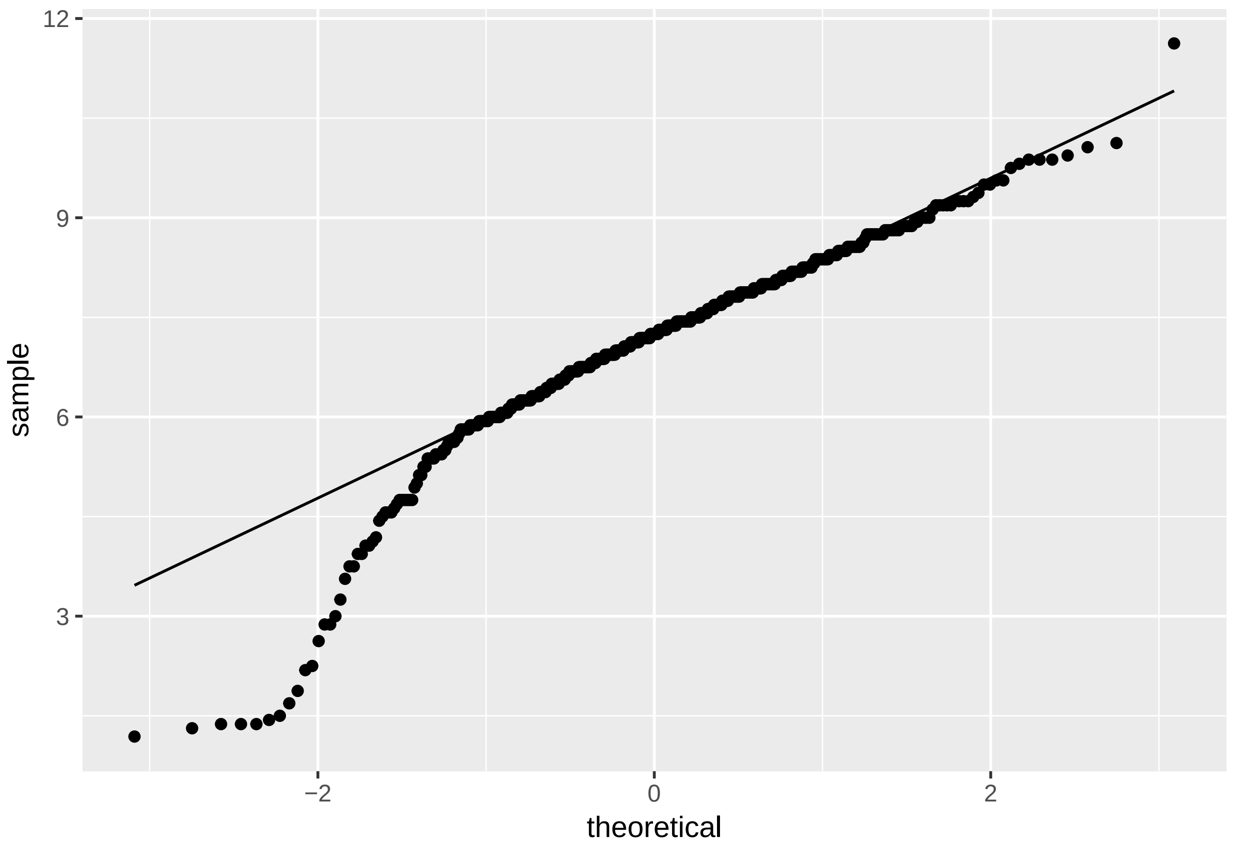
So, we were assessing normality. What about that?

It is mostly normal-looking, but I am suspicious about those *very* low birth weights, the ones below about 4 pounds. There are 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.

A normal quantile plot is better for assessing normality than a histogram is, but I won't make you do one until we have seen the idea in class. Here's the normal quantile plot for these data:

```
ggplot(bw,aes(sample=`Weight (pounds)`))+stat_qq()+stat_qq_line()
```



This is rather striking: the lowest birthweights (the ones below 5 pounds or so) are *way* too low for a normal distribution to apply. The top end is fine (except perhaps for that one very heavy baby), but there are too many low birthweights for a normal distribution to be believable. Note how much clearer this story is than on the histogram.

Having said that, the *t*-test, especially with a sample size as big as this (500), behaves *very* well when the data are somewhat non-normal (because it takes advantage of the Central Limit Theorem: that is, it's the *sampling distribution of the sample mean* whose shape matters). So, even though the data are definitely not normal, I wouldn't be too worried about our test.

This perhaps gives some insight as to why Freedman-Diaconis said we should use so many bins for our histogram. We have a lot of low-end outliers, so that the IQR is actually *small* compared to the overall spread of the data (as measured, say, by the SD or the range) and so FD thinks we need a lot of bins to describe the shape. Sturges is based on data being approximately normal, so it will tend to produce a small number of bins for data that have outliers.

Question title

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 http://www.utsc.utoronto.ca/~butler/c32/nenana.txt.

Yes, we saw these data before.

(a) Read the data into R, as before, or use the data frame that you read in before. 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://www.utsc.utoronto.ca/~butler/c32/nenana.txt"
nenana=read_tsv(myurl)
```

```
## Parsed with column specification:
## cols(
##   Year = col_integer(),
##   JulianDate = col_double(),
##   `Date&Time` = col_character()
## )
```

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.

(b) Obtain a 90% confidence interval for the mean `JulianDate`. What interval do you get? Looking back at your histogram, do you have any doubts about the validity of what you have just done?

Solution

This is a matter of using `t.test` and pulling out the interval. Since we are looking for a non-standard interval, we have to remember `conf.level` as the way to get the confidence level that we want. I'm going with `with` this time, though the dollar-sign thing is equally as good:

```
with(nenana,t.test(JulianDate,conf.level=0.90))
```

```
##
##  One Sample t-test
##
## data:  JulianDate
## t = 197.41, df = 86, p-value < 2.2e-16
## alternative hypothesis: true mean is not equal to 0
## 90 percent confidence interval:
##  124.4869 126.6018
## sample estimates:
## mean of x
##  125.5443
```

Between 124.5 and 126.6 days into the year. Converting that into something we can understand (because I want to), there are $31 + 28 + 31 + 30 = 120$ days in January through April (in a non-leap year), so this says that the mean breakup date is between about May 4 and May 6.

The t-test is based on an assumption of data coming from a normal distribution. The histogram we made earlier looks pretty much normal, so there are no doubts about normality and thus no doubts about the validity of what we have done, on the evidence we have seen so far. (I have some doubts on different grounds, based on another of the plots we did earlier, which I'll explain later, but all I'm expecting you to do is to look at the histogram and say "Yep, that's normal enough". Bear in mind that the sample size is 87, which is large enough for the Central Limit Theorem to be pretty helpful, so that we don't need the data to be more than "approximately normal" for the sampling distribution of the sample mean to be very close to t with the right df.)

(c) An old-timer in Nenana strokes his grey beard and says "When I were young, I remember the tripod used to fall into the water around May 10". In a non-leap year, May 10 is Julian day

130. Test the null hypothesis that the mean `JulianDay` is 130, against the alternative that it is less. What do you conclude? What practical implication does that have (assuming that the old-timer has a good memory)?

Solution

The test is `t.test` again, but this time we have to specify a null mean and a direction of alternative:

```r
with(nenana,t.test(JulianDate,mu=130,alternative="less"))
```

```
##
##  One Sample t-test
##
## data:  JulianDate
## t = -7.0063, df = 86, p-value = 2.575e-10
## alternative hypothesis: true mean is less than 130
## 95 percent confidence interval:
##      -Inf 126.6018
## sample estimates:
## mean of x
##  125.5443
```

For a test, look first at the P-value, which is 0.0000000002575: that is to say, the P-value is very small, definitely smaller than 0.05 (or any other *alpha* you might have chosen). So we *reject* the null hypothesis, and conclude that the mean `JulianDate` is actually *less* than 130.

Now, this is the date on which the ice breaks up on average, and we have concluded that it is *earlier* than it used to be, since we are assuming the old-timer's memory is correct.

This is evidence in favour of global warming; a small piece of evidence, to be sure, but the ice is melting earlier than it used to all over the Arctic, so it's not just in Nenana that it is happening. You don't need to get to the "global warming" part, but I *do* want you to observe that the ice is breaking up earlier than it used to.

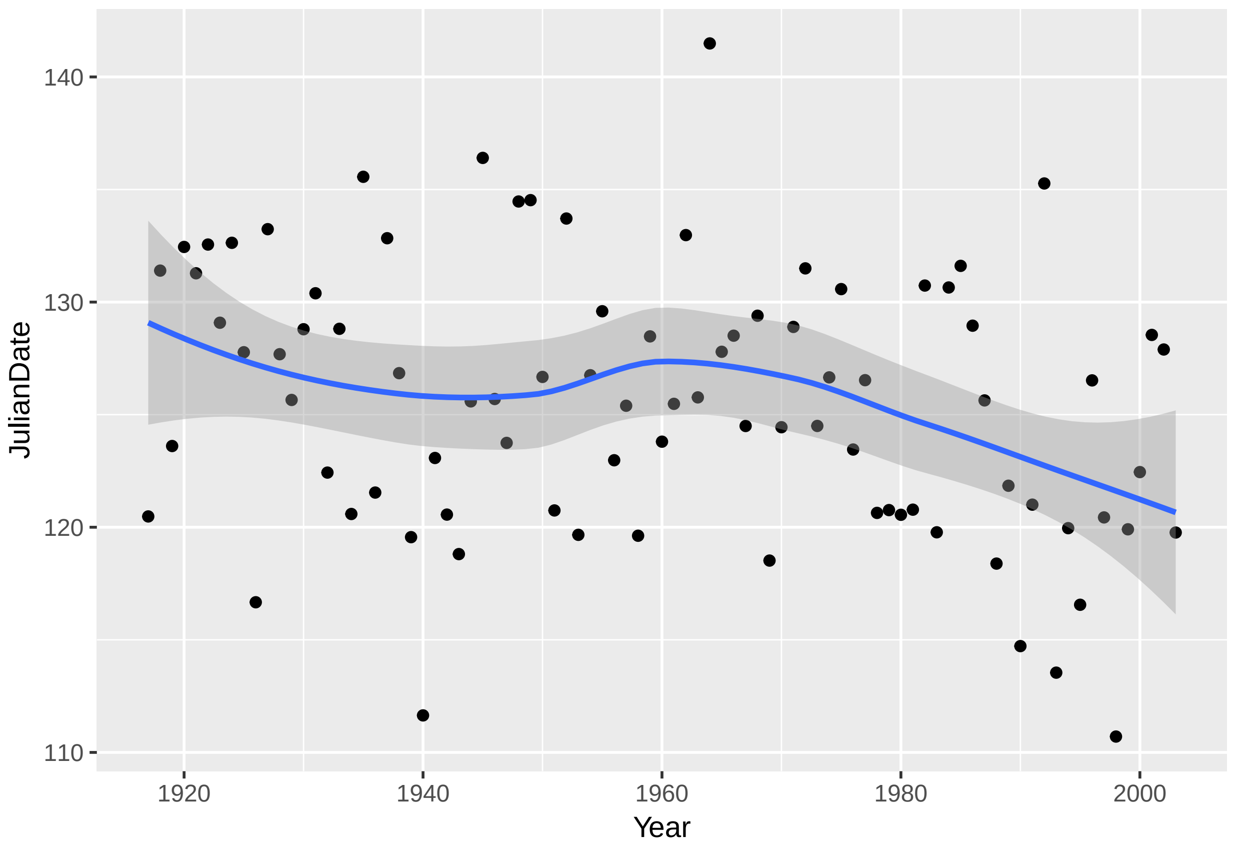
(d) Plot `JulianDate` against `Year` on a scatterplot. What recent trends, if any, do you see? Comment briefly. (You did this before, but I have some extra comments on the graph this time, so feel free to just read this part.)

Solution

I liked the `ggplot` with a smooth trend on it:

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

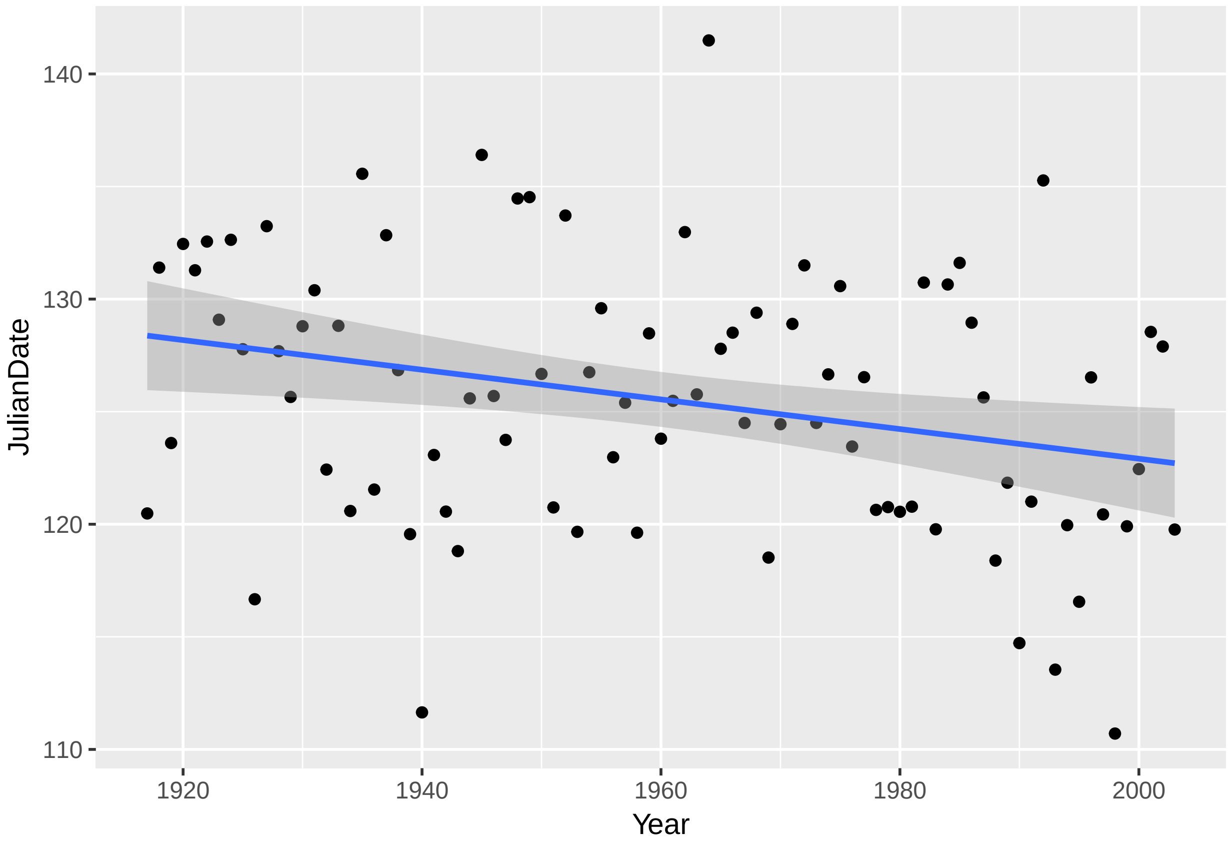
```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```



There was 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 (and consistent with the test we did earlier). Note that the old-timer's value of 130 is the kind of `JulianDate` we would typically observe around 1920, which would make the old-timer over 90 years old.

All right, why did I say I had some doubts earlier? Well, because of this downward trend, the mean is not actually the same all the way through, so it doesn't make all that much sense to estimate it, which is what we were doing earlier by doing a confidence interval or a hypothesis test. What would actually make more sense is to estimate the mean `JulianDate` *for a particular year*. This could be done by a regression: predict `JulianDate` from `Year`, and then get a '`confidence interval for the mean response'' (as you would have seen in B27 or will see in C67). The trend isn't really linear, but is not that far off. I can modify the previous picture to give you an idea. Putting in`method="lm"`fits a line; as we see later,`lm' does regressions in R:

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



Compare the confidence interval for the mean `JulianDate` in 1920: 126 to 131 (the shaded area on the graph), with 2000: 121 to 125. A change of about 5 days over 80 years. And with the recent trend that we saw above, it's probably changing faster than that now. Sobering indeed.